Uniwersytet Warszawski
Wydział Fizyki

Marcin Polkowski
Nr albumu: 251328

Local seismic events in area of Trans European Suture Zone based on data from PASSEQ experiment

Praca magisterska
na kierunku Fizyka
w zakresie Geofizyki

Praca wykonana pod kierunkiem
dr Moniki Wilde-Piórko
Zakład Fizyki Litosfery
Instytut Geofizyki
Wydział Fizyki UW

Warszawa, czerwiec 2012

Oświadczenie kierującego pracą

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem i stwierdzam, że spełnia ona warunki do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

data *Podpis kierującego pracą*

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

data *Podpis autora pracy*

Abstract

Two years of continuous seismic recordings from over 80 stations in Poland of the PASSEQ 2006-2008 experiment is analyzed to locate unknown local seismic events. Paper presents detection (Carl Johnson's STA/LTA algorithm) and location procedure (grid search method). Four unknown events, three in Baltic Sea and one near Jarocin, are detected and localized.

Słowa kluczowe

local seismicity, PASSEQ 2006-2008, passive seismic experiment, grid search, automatic detection

Dziedzina pracy (kody wg programu Socrates-Erasmus)

(13.2) Fizyka

Tytuł pracy w języku angielskim

Local seismic events in area of Trans European Suture Zone based on data from PASSEQ experiment

Contents

List of Figures

Listings

Chapter 1

Introduction

1.1 Tectonics of Poland

Poland is located in the junction of three major tectonics units: the **Precambrian East European Craton**, the **Palaeozoic Platform** of Central and Western Europe, and the **Alpine orogen** represented by the Carpathian Mountains.

Transition zone between the East European Craton and Palaeozoic Platform is called the **Trans European Suture Zone** (TESZ). Location of the TESZ [Pharaoh, 1999] in Poland is shown in Figure 1.1. The TESZ is a deep pocket of sediments, where the surface of Crystaline Basement is sometimes deeper than 20 km. The geological structure between north-east and south-west Poland varies in terms of thickness of sedimentary cover and in terms of seismic velocities. This transition zone between two geologically different units contains series of faults which can be considered as a source of seismic activity.

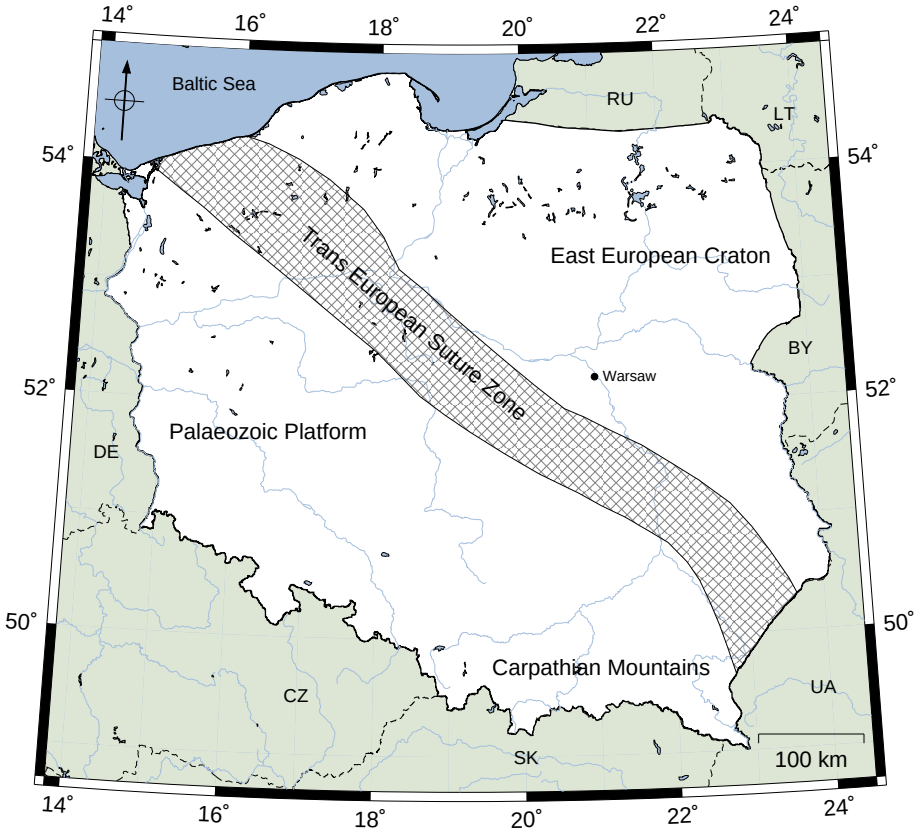

Figure 1.1
Simplified tectonics of territory of Poland.

1.2 Local seismicity in Poland

Poland is usually classified as a seismic free (aseismic) territory however every year there are multiple seismic events induced by mining industry in Lubin-Głogów Copper Basin, Upper Silesia, Rybnik Coal District and Bełchatów open-pit coal mine [Guterch, 2009]. These induced earthquakes have usually small magnitudes of M2.0 - M2.5. Small tectonic events occur regularly in Carpathian Mountains. Figure 1.2 shows earthquakes in Poland from the European-Mediterranean Seismological Centre on-line catalog from 2004 to 2012 and almost no seismic activity around the Trans European Suture Zone is observed. Reason for that is a short time of observation and significant distance to permanent seismic stations, so small local events could not be detected.

More information about activity in the area of the TESZ is given by historical catalogs of seismic events [Guterch, 2009]. There were 69 known events from 1496 to 2005. For most of them there are no instrumental data so their localization was done according to intensity data. The epicenters of historical earthquakes are shown in Figure 1.3. Accumulation of seismic epicenters is well visible in Carpathians and Sudets in south Poland, while other events are located (approximately) around the TESZ.

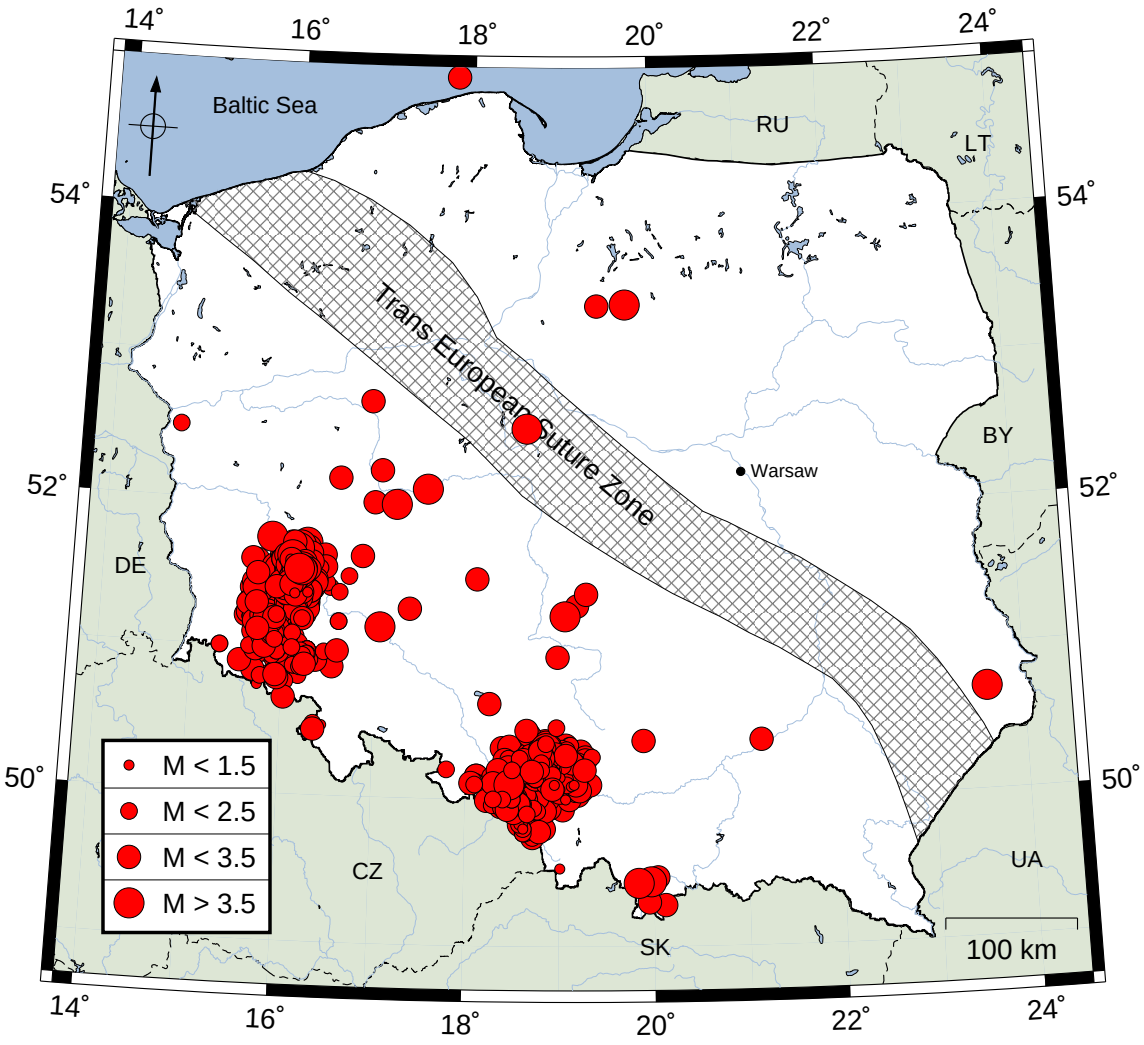

Figure 1.2
Location of epicenters of 2909 local earthquakes stored in European-Mediterranean Seismological Centre on-line catalog (from 2004 to 2012). Two large event aggregations are in Lubin-Głogów Copper Basin, Upper Silesia and Rybnik Coal District.

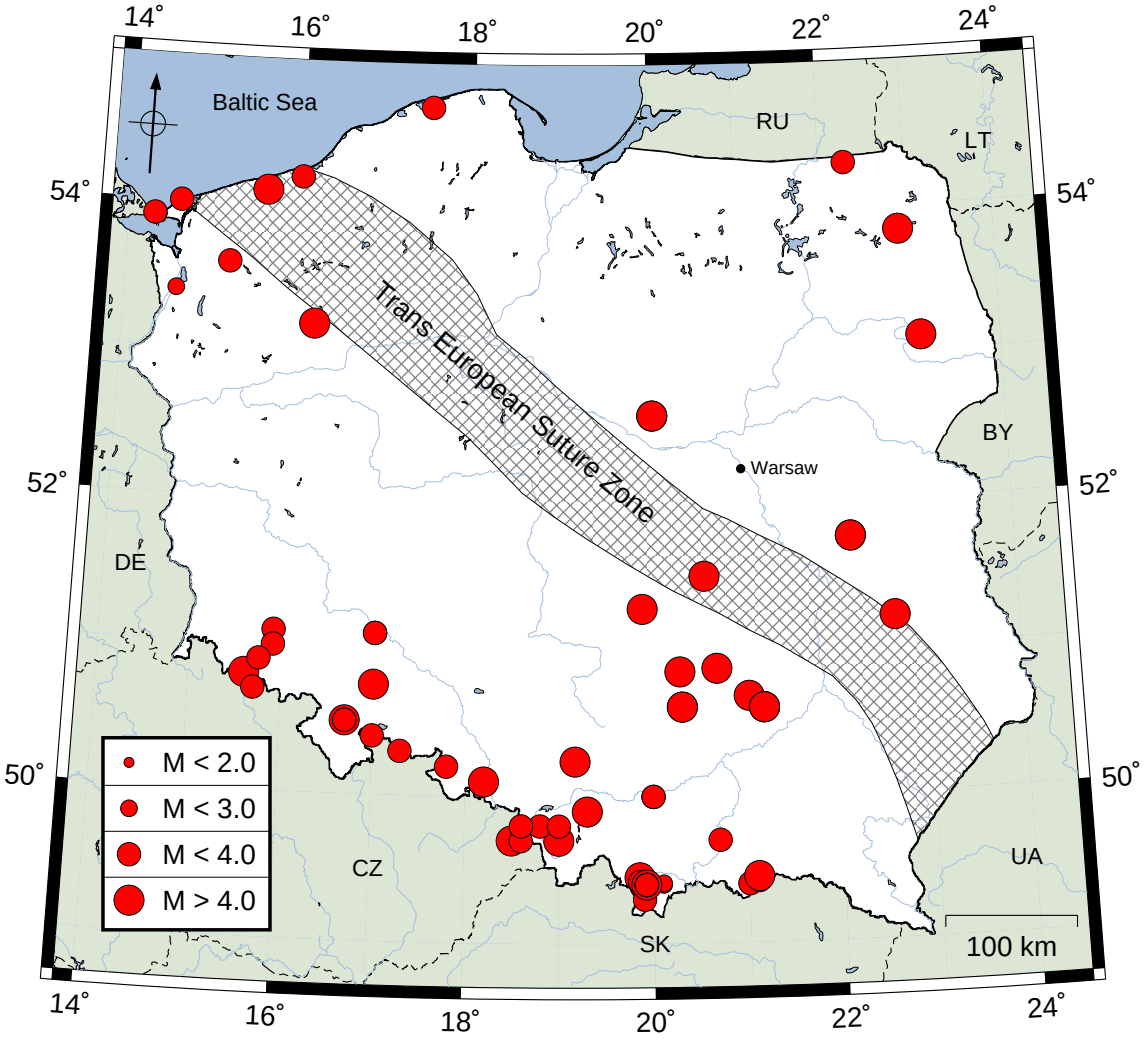

Figure 1.3
Location of epicenters of 69 local earthquakes from 1496 to 2005.

1.3 Polish seismological network

In period from 2006 to 2008 Polish permanent seismological network managed by Institute of Geophysics P.A.S. included 6 broadband seismic stations. Station localization is shown in Figure 1.4. Station WAR was located in the middle of Warsaw city where urban noise level was very significant. For that reason WAR station was closed in October 2007, moved south-west and renamed. New seismic station (BEL) stated operation in January 2009. A small number of stations (WAR station can be omitted because of noise level) allowed recording and tracking larger local and teleseismic events but station distribution was not sufficient to observe any local activity in central and east Poland.

KSP and OJC observatory Station Bulletins of local and teleseismic events are regularly prepared and published on-line. Every recorded wave (phase) is noted for known events. KSP station is located close to Lubin-Głogów Copper Basin and Upper Silesia and Rybnik Coal District and many local events from these areas are noted in KSP station bulletin. Local events in station bulletin have specified origin time, regional location (Lubin, Silesia, etc.) and approximate magnitude. Note that for single station event detection and exact localization is not possible. Example of station bulletin of KSP station is shown in Appendix A (Listing A.1).

At the end of each year Local Bulletin is published at Institute of Geophysics P.A.S. It contains a summary of local seismicity in Poland during whole year. List of larger events, recorded at multiple stations, is given with origin time and coordinates of epicenter. Example of Local Bulletin is shown in Appendix A (Listing A.2).

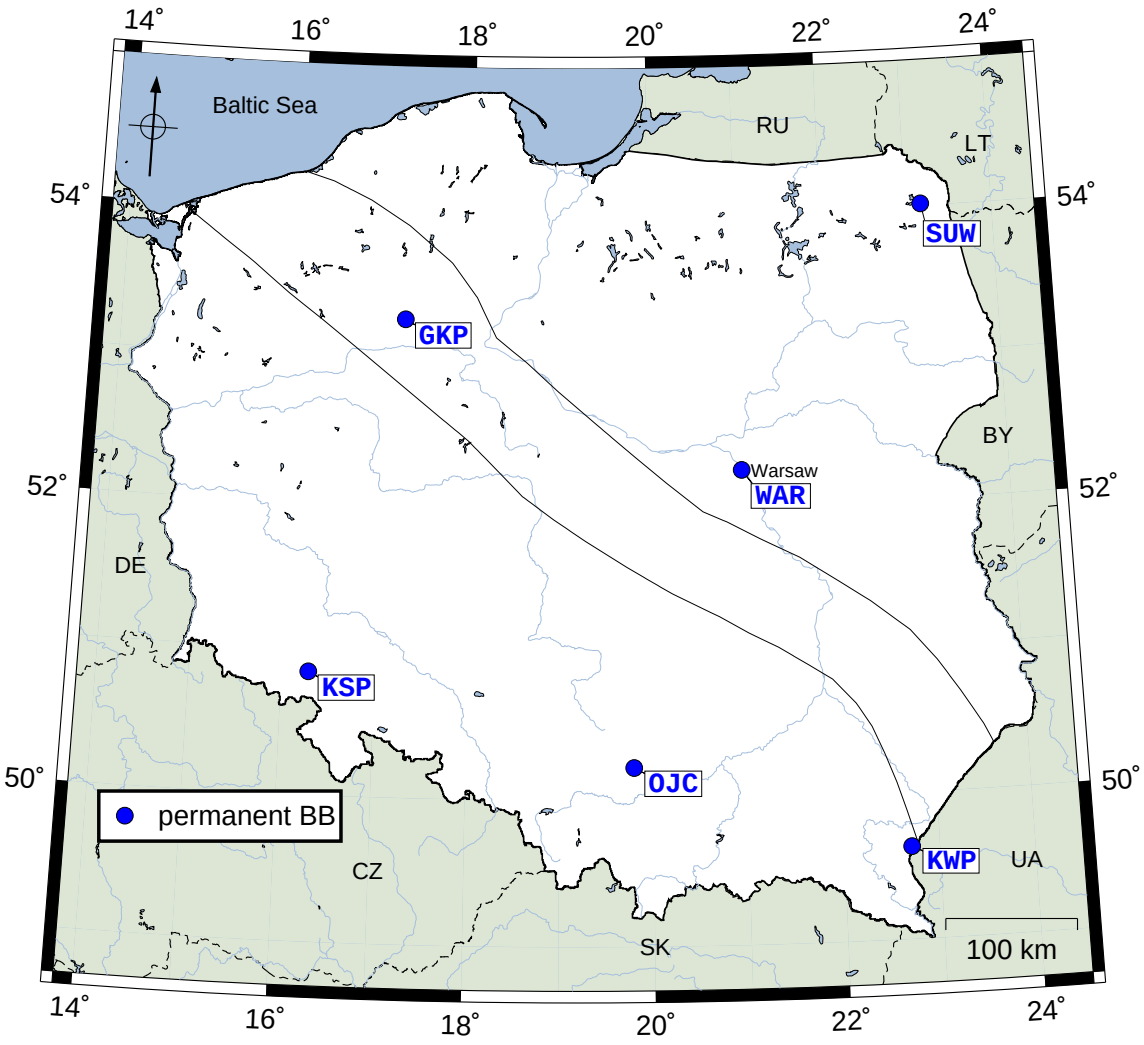

Figure 1.4
Location of permanent broadband seismic stations in Poland operating from 2006 to 2008.

Chapter 2

PASSEQ 2006-2008 experiment

PASSEQ 2006-2008 (PASsive Seismic Experiment in TESZ) was passive seismic experiment that took place in Central Europe in Germany, Czech Republic, Poland and Lithuania [Wilde-Piórko et al., 2008]. The main aim of the project was to study a deep structure of lithosphere and astenosphere around the Trans European Suture Zone. Almost 200 temporary broadband and short-period stations were located in four countries - most of them in Poland.

Stations were deployed along the P4 refraction profile and symmetrically around it. Position

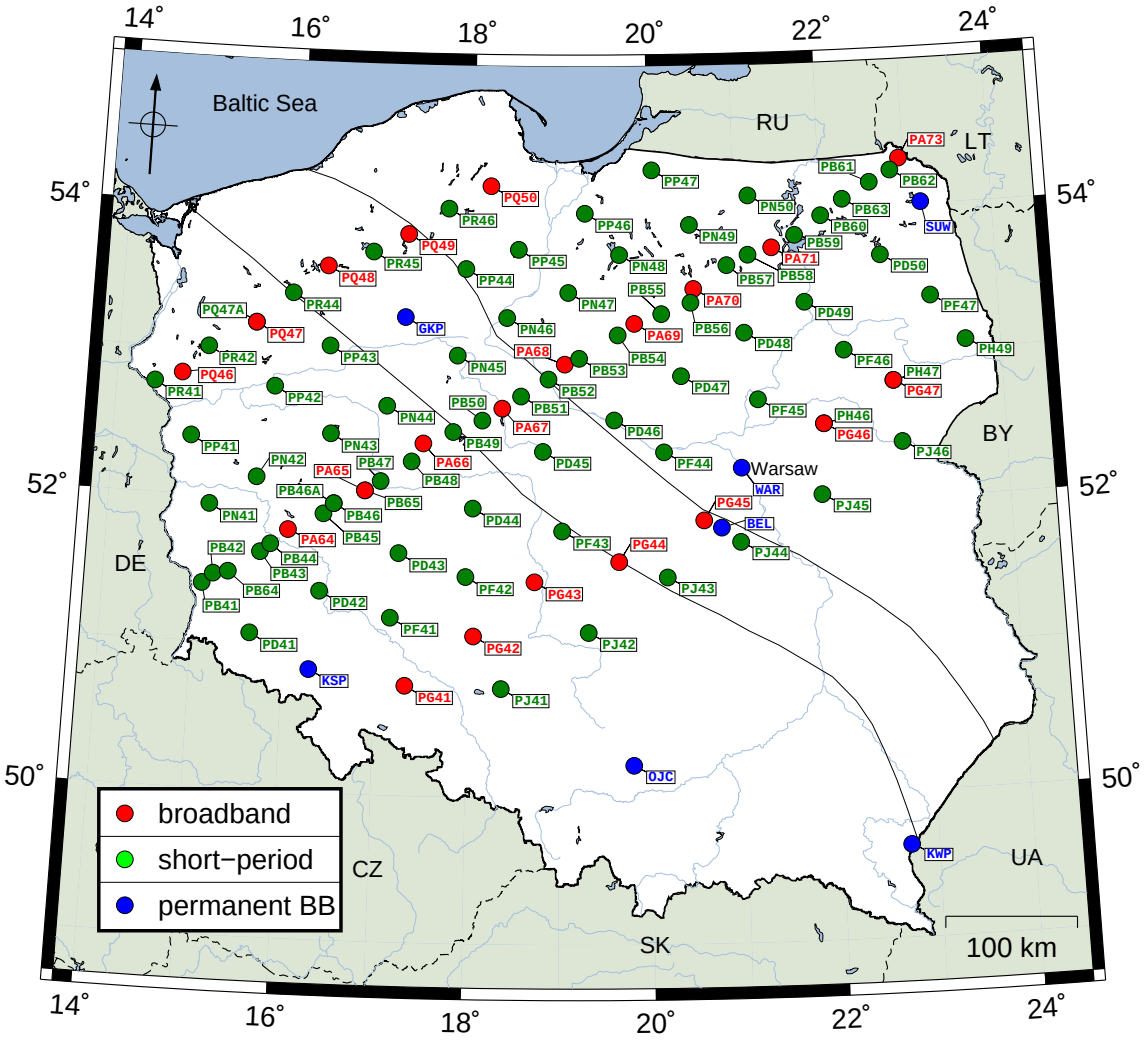

Figure 2.1
Position of PASSEQ 2006-2008 and permanent stations in Poland. Note that not all station were operational at the same time.

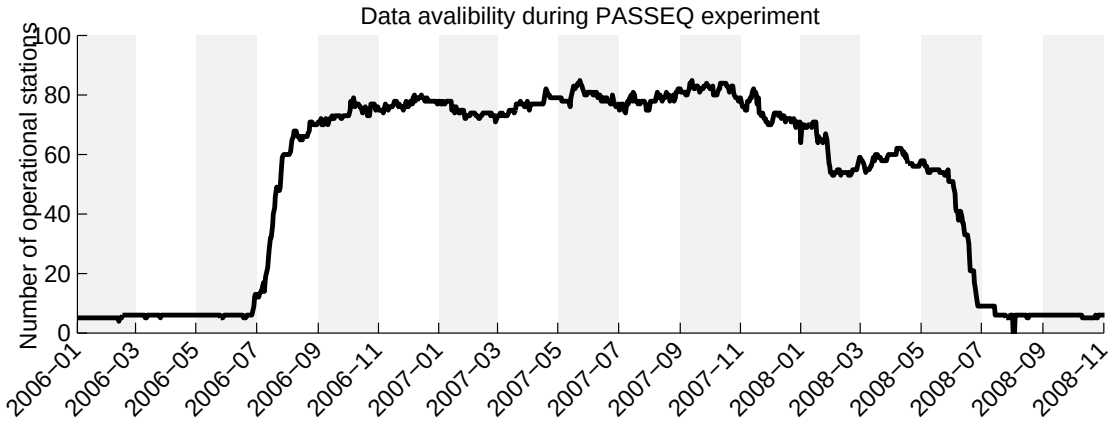

Figure 2.2
Number of stations (PASSEQ 2006-2008 + permanent) operating in territory of Poland in years 2006, 2007 and 2008.

of stations in Poland is shown in Figure 2.1.

PASSEQ 2006-2008 was planned to use mostly teleseismic data for various analysis methods including seismic tomography, receiver function, surface waves, SKS splitting, ambient noise and others. Local events detection wasn't primary goal of this project.

Due to limited founds and limited equipment availability PASSEQ 2006-2008 consisted of different types of seismic sensors and digitizers. Different stations in different time periods have recorded with different data sampling rate (20Hz, 50Hz, 100Hz and 125Hz).

Stations were deployed in July 2006 and removed in August 2008, but for numerous reasons (including technical) not all stations had continuous recordings during experiment. Figure 2.2 shows number of working stations in time and Figure 2.3 shows example of local event from Lubin in August 18th 2006 recorded by PA64 seismic station.

All mentioned problems and project limitations made usage of PASSEQ 2006-2008 data a challenging task. All raw data from stations was processed in GFZ in Potsdam, Germany. Data in miniseed format was then available to all project participants. Unfortunately after data set was processed in GFZ, some problems with data was acknowledged due to file formats and missing data. Next chapter describes steps that was necessary to unify data before further analysis.

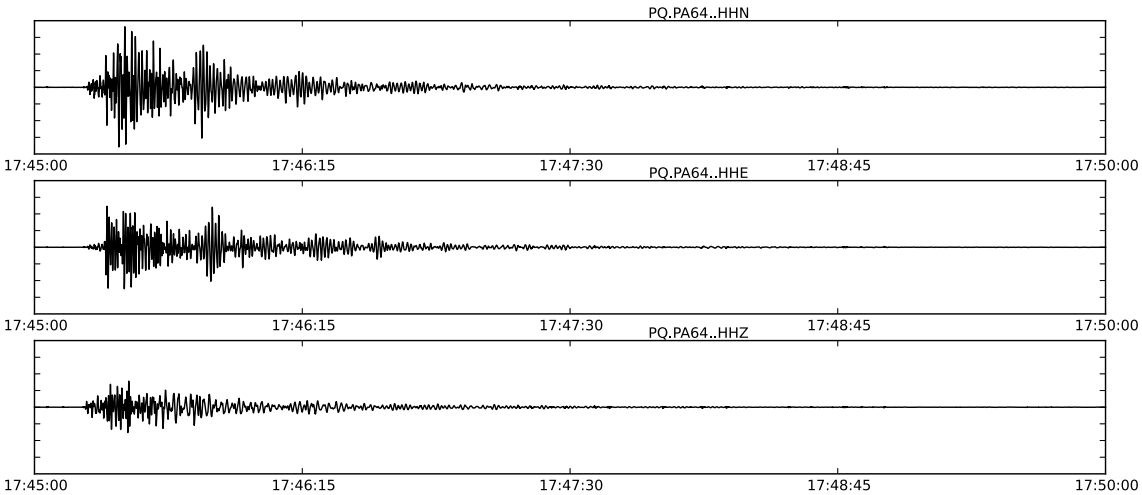

Figure 2.3
Example of local event from Lubin (M3.5) in August 18th 2006 recorded by PASSEQ 2006-2008 seismic station PA64. Three components are shown (north-south, east-west and vertical). Recording is passed by bandpass filter with corner frequencies of 4Hz and 9.5Hz.

Chapter 3

Data analysis

In this chapter data analysis, resulting in a list of automatically calculated local events will be described. Data analysis is divided into multiple steps and for each step a computer program (or script) was prepared. All significant source codes are provided so reader can analyze, understand, repeat and improve whole process.

All scripts were prepared in cross platform languages such as python, MATLAB, php and C++ (Figure 3.1). Calculations were also done step by step and information between steps was exchanged mostly using ASCII text files for better process debugging. ObsPy - a Python Toolbox for seismology/seismological observatories was used for seismic data processing (reading, writing, etc.) [Megies et al., 2011, Beyreuther et al., 2010].

Each step of procedure can be modified to meet any special requirements. Programs are more or less optimized. Total calculation (for whole PASSEQ 2006-2008 data set in Poland) took about five days on four modern desktop computers. Further code optimizations are possible (migrate everything to single C++ program) but not necessary for calculation presented in this paper.

Figure 3.1
Software used for analysis.

3.1 Data preparation

The PASSEQ 2006-2008 data was downloaded from GFZ in 24-hour miniseed files. Over 2.5 TB of data was stored locally. Unfortunately most data files were corrupted in one or more of following ways:

- some files contained multiple overlapping traces in one file,
- some day files did not start at midnight and did not cover whole 24-hours,
- files had different size of data blocks,
- some data samples were missing,
- some files were big endian while other were little endian.

Before any further processing data set had to be unified and cleaned. For that purposes ObsPy and special Python script was created. Its aim was to:

- clean overlaps,
- unify block size to 4096 bits,
- select only Z-component,
- save cleaned data to 60-minute files starting every 30 minutes,
- file with samples missing were omitted.

30-minute overlapping of data files was useful during detection and will be described in next sections.

A source code of data preparation script is shown in Listing 3.1. Details about script are given in in-line comments.

Listing 3.1
Python script for data preparation

```
#!/usr/bin/env python

# Import all necesary tools and toolboxes:
import glob
from obspy.core import read, Stream, Trace
from obspy.core import UTCDateTime
from time import strftime
import os
import sys

# Set time range of data preparation:
global_time_start   = 1136116800    #2006-01-01 12:00
global_time_end     = 1230638400    #2008-12-30 12:00

# Station name is given as command line parameter:
station = sys.argv[1]

# Set input and output paths for data files:
path        = '/INPUT_PATH/' + station + '/'
output_path = '/OUTPUT_PATH/' + station + '/'

# Check if output path exists. If not create directory:
if not os.path.exists(output_path):
    os.makedirs(output_path)

# Generate array of days in time range
step = 24*3600
global_times = range(int(global_time_start),int(global_time_end), int(step))

# Loop through all days and process them:
for start_time in global_times:
    # Change date format
    UTC_start_time = UTCDateTime(start_time).getDateTime()

    # Calculate two dates: current day and next day:
```

```
    UTC_file1 = UTCDateTime(start_time).getDateTime()
    UTC_file2 = UTCDateTime(start_time+24*3600).getDateTime()

    # Inform user about progress:
    print 'Processing station ' + station + ' @ ' + UTC_file1.strftime("%Y-%m-%d %H
        :%M:%S") + ' to ' + UTC_file2.strftime("%Y-%m-%d %H:%M:%S")

    # Prepare paths of two data files - current and next day:
    file1 = path + UTC_file1.strftime(station + '_%Y-%m-%d.mseed')
    file2 = path + UTC_file2.strftime(station + '_%Y-%m-%d.mseed')

    # Get file sizes:
    size1 =  os.path.getsize(file1)
    size2 =  os.path.getsize(file2)

    # check if first file exists. If yes load it to memory. If not create empty
        trace:
    if size1 > 0:
        A = read(file1)
    else:
        A=Stream(Trace())

    # check if second file exists. If yes load it to memory. If not create empty
        trace:
    if size2 > 0:
        B = read(file2)
    else:
        B=Stream(Trace())

    # Combine two loaded files:
    DATA = A + B

    # Select only Z-component
    DATAZ = DATA.select(channel="BHZ")

    # Clean overlaps in data
    DATAZ._cleanup()

    # Create array of start times of hour blocks every 30 minutes:
    step2 = 3600
    local_times = range(int(start_time),int(start_time+step), int(step2/2))

    # Process every hour between 12:00 current day and 12:00 next day:
    # Every trace in combined file is analyzed separatly:
    for trace_num in range(len(DATAZ)):

        # Get current trace:
        TRACE = DATAZ[trace_num]

        # Process every hour:
        for local_start_time in local_times:

            # Create local instance of trace and trim it to current hour:
            LOCAL_TRACE = TRACE.copy()
            LOCAL_TRACE.trim(UTCDateTime(local_start_time),UTCDateTime(
                local_start_time)+step2)

            # Check if there is no samples missing in trimmed trace:
            if LOCAL_TRACE.stats.npts > step2 * LOCAL_TRACE.stats.sampling_rate -
                1:

                # Calculate trace start time:
                UTC_trace_time = UTCDateTime(local_start_time)

                # Write trace to file in output path
                LOCAL_TRACE.write(output_path + UTC_trace_time.strftime(station + "
                    _%Y-%m-%d_%H-%M-%S") + '.mseed', format="MSEED")
```

3.2 Signal detection

A detection goal was to analyze continues time series (seismic record) and provide list of moments where possible signal (seismic wave) was recorded. Main python function was prepared to return a list of signal detections for a given file name (already prepared one-hour, one station, one component miniseed file).

The analysis is done in three steps: data loading, filtration and detection of signal.

Data loading. Data file was loaded from repository.

Filtration. Data was filtered with zero-phase bandpass Butterworth filter (from 4.0Hz to 9.5Hz). After filtration mean value of trace was removed and amplitude has been normalized to constant value (the same value for all stations). An example of filtration of broadband trace is shown in Figure 3.2.

Detection of signal. The simplest detector is amplitude threshold: if recorded signal has amplitude higher then threshold detection is noted. This type of detector is of course of poor quality because of noisiness of the recording. A detector has to adapt to current conditions (eg. noise level). The simplest way to create such a detector is calculating two moving averages over signal with different windows: short and long. Most of seismic detectors use relation between short term average (STA) and long term average (LTA).

In this paper Carl Johnson's STA/LTA detection algorithm was used. This algorithm calculates four moving averages and takes two parameters:

$$\mathbf{eta} = \mathbf{star} - (\mathit{ratio} * \mathbf{ltar}) - \mathrm{abs}(\mathbf{sta} - \mathbf{lta}) - \mathit{quiet},$$

where:

- **eta** - detector response - value over 0 means detection,
- **sta** - short term moving average of signal,
- **lta** - long term moving average of signal,
- **star** - short term moving average of absolute value of signal and **lta** difference,
- **ltar** - long term moving average of **star**,
- **ratio** and **quiet** - sensitivity parameters.

It is recommended for long term average to be 8 times longer then short time average. In this paper short term average had 4 seconds window and long term average had 32 seconds window. Both **ratio** and **quiet** parameters had value 2.

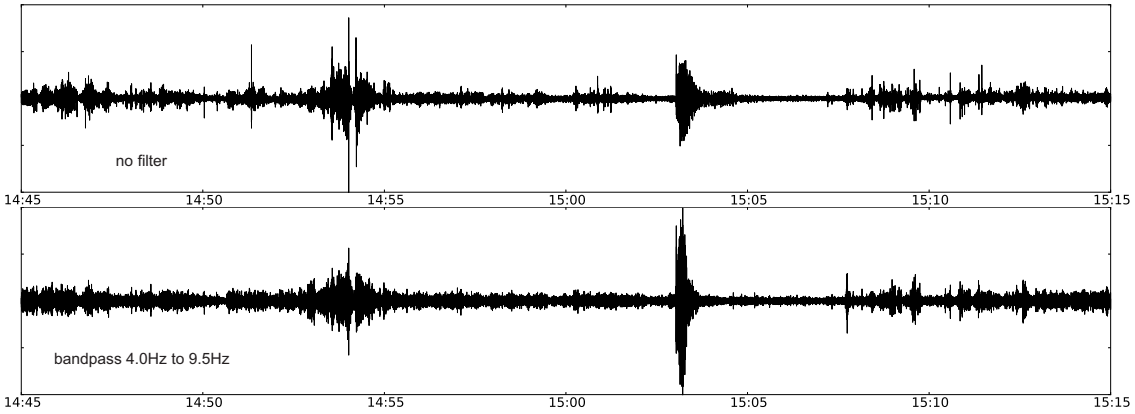

Figure 3.2
30 minute recording of station PA64 in August 24th 2006. Event of magnitude M2.5 was recorded at 15:03.

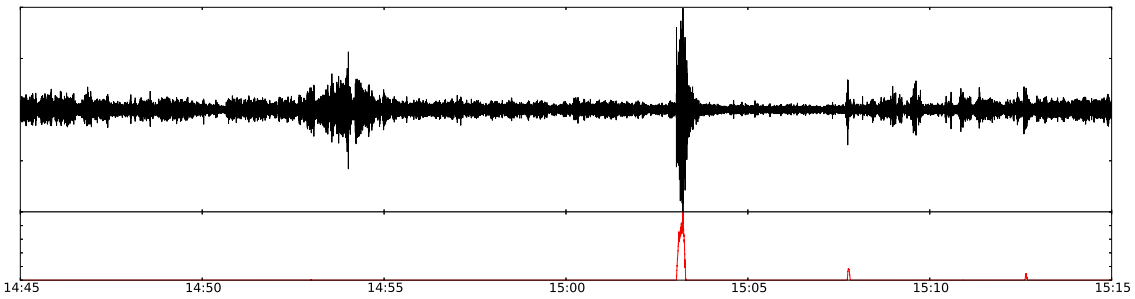

Figure 3.3
30 minute recording of station PA64 in August 24th 2006. M2.5 event was recorded at 15:03. Upper, black curve is recorder signal. Bottom, red curve is detector response.

Carl Johnson's STA/LTA was originally implemented in ObsPy toolbox but it's performance was extremely poor: over 20 seconds was required to calculate detection in one hour data block. Carl Johnson's STA/LTA algorithm was optimized by author and implemented in C++ as Python module. Performance increase to 0.1 second per one hour block was achieved. Both implementations are shown and compared in Appendix B.

Values of moving averages have to stabilize in every processed file, and for that reason only detection from middle 30 minutes of every file were taken for further analysis. Example of detection for broadband station is shown in Figure 3.3.

Detector response to signal is time series of the same size like signal itself. It has to be analyzed to provide list of points in time where detection occurred. Every value above 0 means detection, so for providing list of detection times it was necessary to loop through detector response over time and locate points where detection curve (**eta**) changed value from negative to positive (detection time) and positive to negative (to calculate duration of detection). Additionally maximum and mean value of **eta** was noted for every detection.

PASSEQ 2006-2008 data set of Poland is almost 2.5 million one-hour data files (note that data set is doubled due to overlapping). A list of files was stored in MySQL database which kept track of which data file was already processed. The same database was used for storing information about detections (finally there were over 40 million detections). Using database for information storage allowed parallel processing of data files on multi core computer and multiple computers (in the same network). The signal detection for whole PASSEQ 2006-2008 data set took about three days on standard 8-core desktop computer. Full source code used for signal detection is shown in Listing 3.2. See in-line comments for more information.

Listing 3.2
Python script for signal detection

```
#!/usr/bin/env python

# Import all necesary tools and toolboxes:
import glob
import matplotlib
matplotlib.use('Agg')
from pylab import *
import matplotlib.pyplot as plt
from obspy.core import read
from obspy.core import UTCDateTime
from obspy.signal.trigger import *
from time import strftime
import numpy as np
import time
import urllib2
import string
import sys
import string
from matplotlib import dates
from obspy.taup.taup import *
from obspy.signal import cornFreq2Paz
import _mysql
from multiprocessing import Pool
import carlStaTrigC

```

```
26
27 # Definition of ProcessFile function.
28 # This function runs detection process over one data file (1-hour)
29 def ProcessFile(x):
30
31     # Start clock - performance monitoring"
32     t0 = time.time();
33
34     # Set input and output paths for data files:
35     input_dir   = '/INPUT_PATH/' + station + '/'
36     output_dir  = '/OUTPUT_PATH/' + station + '/'
37
38     # Connect to database:
39     db = _mysql.connect(host="***", user="***", passwd="***", db="***")
40
41     # Get number of files needed to be processed
42     db.query("SELECT count(*) FROM traces WHERE done = 0;")
43     mr = db.use_result().fetch_row()
44     db.close()
45
46     # Check if there are files to be processed:
47     if int(mr[0][0]) > 0:
48
49         # For performance reason new database connection is opened:
50         db = _mysql.connect(host="***", user="***", passwd="***", db="***")
51
52         # Lock tables in database. It is not possible for two threds to analyze the
               same data file at once:
53         db.query("LOCK TABLES traces WRITE;")
54
55         # Select data file to be analyzed:
56         db.query("SELECT uid, stacja, plik FROM traces WHERE done = 0 order by date
               LIMIT 1;")
57         mr = db.use_result().fetch_row()
58
59         # Flag selected file as "in progress"
60         db.query("UPDATE traces SET done = -1 WHERE uid = " + str(mr[0][0]) + ";")
61
62         # Unlock tables - other thread can now select their files:
63         db.query("UNLOCK TABLES;")
64
65         # Compose path of file to be analyzed:
66         file = input_dir + mr[0][1] + '/' + mr[0][2]
67
68         # Process file name:
69         date_name = mr[0][2];
70         date_name = string.replace(date_name,mr[0][2] + '_','')
71         date_name = string.replace(date_name,'.mseed','')
72
73         # Load data file into memory:
74         data = read(file)
75
76         # Cleanup data file (just in case):
77         data._cleanup();
78
79         # Select main trace in loaded file:
80         trace = data[0]
81
82         # Get loading time and start new timer:
83         load_time = str(time.time()-t0)
84         t0 = time.time();
85
86         # Declare step variable in seconds
87         step = 3600;
88
89         # Declare variables describing trace:
90         # Start time:
91         ts = trace.stats.starttime
92         # End time:
93         te = trace.stats.endtime
94         # Time between samples:
95         df = trace.stats.delta
96         # sampling rate:
97         sr = trace.stats.sampling_rate
98
99         # Only middle 30 minutes of trace are analyzed:
```

```
        hide = 15*60;
        okno = 30*60;
        xlim_max = (hide + okno)/60
        xlim_min = hide/60

        # Apply bandpass filter to trace:
        trace.filter('bandpass',freqmin=4.0, freqmax=9.5, corners=2, zerophase=True
            )

        # Remove mean value from trace
        trace.data = trace.data - trace.data.mean()

        # Normalize trace:
        trace.data = trace.data / max(np.absolute(trace.data))
        trace.data = trace.data * 100000

        # Compute Carl Johnson's STA/LTA detector response:
        cft = np.array(carlStaTrigC.compute(trace.data.tolist(), int(4 * sr), int
            (32 * sr), 2, 2))

        # Get only parts of detector response greater than 0:
        cft = np.where(cft>0,cft,0)

        # Process all detector triggers:
        for on_of in triggerOnset(cft, 0.0, 0.0):

            # Check if detection starts in middle 30 minutes:
            if trace.stats.starttime.timestamp + on_of[0]/sr > trace.stats.
                starttime.timestamp + hide:
                if trace.stats.starttime.timestamp + on_of[0]/sr < trace.stats.
                    starttime.timestamp + hide + okno:

                    # Calculate time of trigger start
                    trigger_start_time = (trace.stats.starttime.timestamp + on_of
                        [0]/sr)

                    # Calculate duration of trigger:
                    trigger_length = (on_of[1] - on_of[0]) / sr

                    # Process trigger:
                    cft_triggered = cft[on_of[0]:on_of[1]]
                    if cft_triggered.any():

                        # Calculate trigger max value
                        trigger_max = max(cft_triggered)

                        # Calculate trigger mean value
                        trigger_mean = sum(cft_triggered)/(on_of[1]-on_of[0])

                        # Insert trigger parameters to database:
                        msg = "'" + trace.stats.station + "'," + str(
                            trigger_start_time) + "," + str(trigger_length) + "," +
                             str(trigger_max) + "," + str(trigger_mean) + ""
                        db.query("INSERT INTO `trigger` VALUES(NULL, " + msg + ");"
                            )

        # Flag file as processed:
        db.query("UPDATE traces SET done = 1 WHERE uid = " + str(mr[0][0]) + ";")

        # Close database connection:
        db.close()

        # Display information about loading and processing time:
        print '\tData loaded in: ' + load_time + "\t" + 'End: ' + str(time.time()-
            t0)


# Detection script is prepared to run on multicore systems.
# Check if current thread is main. Only main thred can invoke other threads:
if __name__ == '__main__':

    # Connect to database:
    db = _mysql.connect(host="***", user="***", passwd="***", db="***")

    # Get number of files needed to be processed
    db.query("SELECT count(*) FROM traces WHERE done = 0;")
```

```
    mr = db.use_result().fetch_row()
    db.close()
    count = int(mr[0][0])

    # Start threads pool for 8-core system:
    pool = Pool(processes=8)

    # Supporting variable optimize is used later in loop:
    optimize = 0

    # Work while number of files to process reaches 0:
    while count > 0:

        # For performance reason new database connection is opened:
        db = _mysql.connect(host="***", user="***", passwd="***", db="***")

        # Get number of files needed to be processed
        db.query("SELECT count(*) FROM traces WHERE done = 0;")
        mr = db.use_result().fetch_row()
        count = int(mr[0][0])

        # Optimize database table every 800 (100 * 8 cores) files processed
        if optimize == 100:
            db.query("OPTIMIZE TABLE traces;");
            print "optimized"
            optimize = 0;
        optimize = optimize + 1

        # Close database connection:
        db.close()

        # Inform user about progress
        print count

        # Start 8 parallel instances of ProcessFile function:
        pool.map(ProcessFile, range(8))
```

3.3 Event grid searching

In previous section a process of detection was described. Unfortunately single station detection cannot be used for seismic event finding. A detector responses to seismic waves but also responses to noise. Additional analysis of detection signals is required to recognize local seismic events.

All methods of such analysis are based on coincidence of detection signals on different stations. In this paper grid search method was used. Grid searching is simply a method of testing if given locations in given times could have been a source of seismic wave [Sambridge and Kennett, 1986]. Area that is being analyzed should be divided into grid. In case of this paper grid of size 0.05 by 0.05 degrees was taken.

Analysis were done one day at the time. A list of detections from all stations for that day was taken. For every grid cell number of operating stations in 150 km radius was calculated. If there was at least 15 working stations traveltime between analyzed grid cell and all stations in radius was calculated using 1-D reference *iasp91* model [Kennett and Engdahl, 1991] (Figure 3.4). All detections were moved back in time by calculated traveltime (different times for different stations!). Then a number of stations in radius with detection signals for every second of analyzed day was calculated. Detections were considered with 3 second tolerance. When number of stations giving detection exceeded 50% of total stations in radius event was noted (time, cell coordinates and number of stations).

After processing all grid cells, a list of times of possible events was generated. A location of possible events was calculated as a weighted average of coordinates of all grid cells that detected at this time. A weight of average equaled to percent of stations giving detection. This is a main advantage of grid search method - a location of detected event is easy to determine.

For better method understanding illustrated step-by-step example is given.

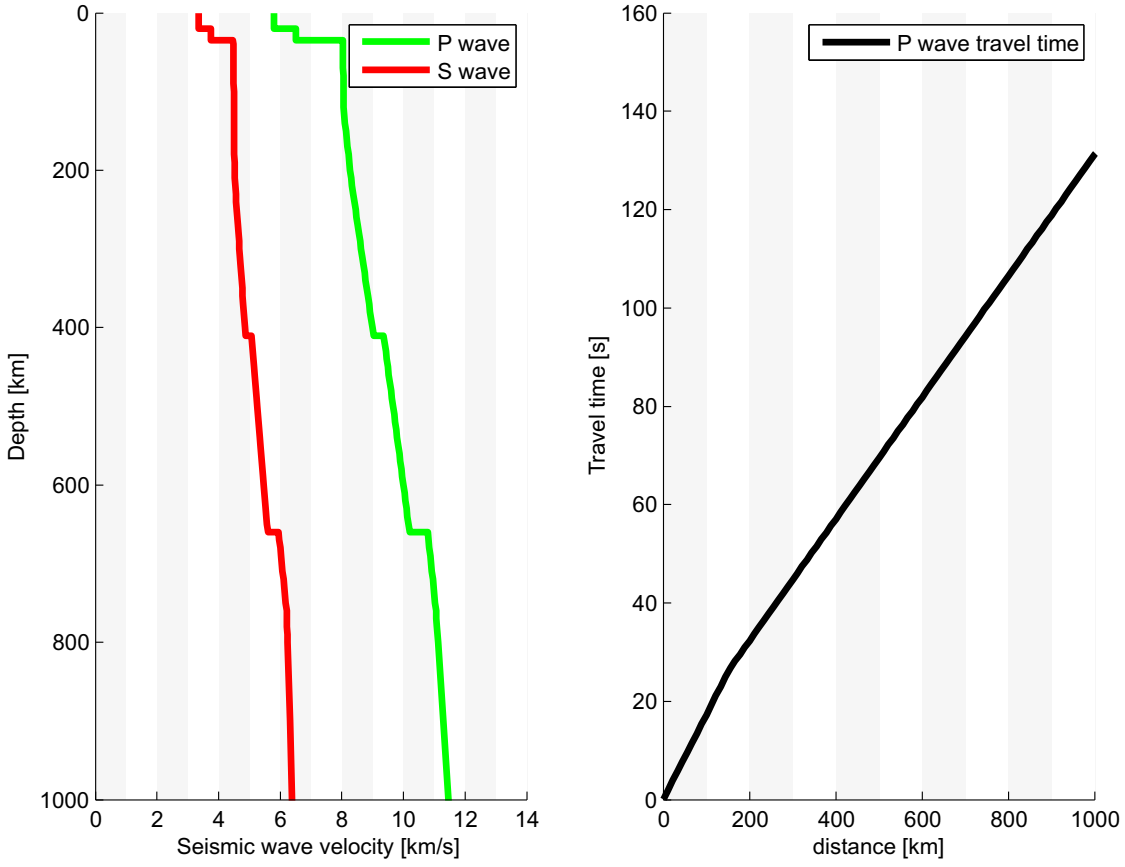

Figure 3.4
P- and S-wave velocity in *iasp91* model up to 1000 km depth (left), traveltime of P-wave in *iasp91* model up to 1000 km distance (right).

3.3.1 Step-by-step example

An example is based on confirmed M3.5 event in Lubin area. Figure 3.5 shows stations being analyzed and location of two test grid cells (A, B) and Figure 3.6 shows recordings from these stations at their original time. It's clear that some event was recorded at every station but there is no indication if it was the same event and what was its location.

Time shift procedure was preformed assuming event location A and B. Shifted records for location A are shown in Figure 3.8 and for location B in Figure 3.7. It's well noticeable that shifted recordings for location A are aligned to approximately the same time - the time of origin of event. In that case it can be assumed that the event occurred near to location A.

Because detection times are taken with 3 second tolerance the closer analyzed cell is to possible event location, the more stations are taking part in coincidence. Figure 3.9 shows map of coincidence level (number of station with detection to total number of stations) for analyzed event calculated in every grid cell in Poland.

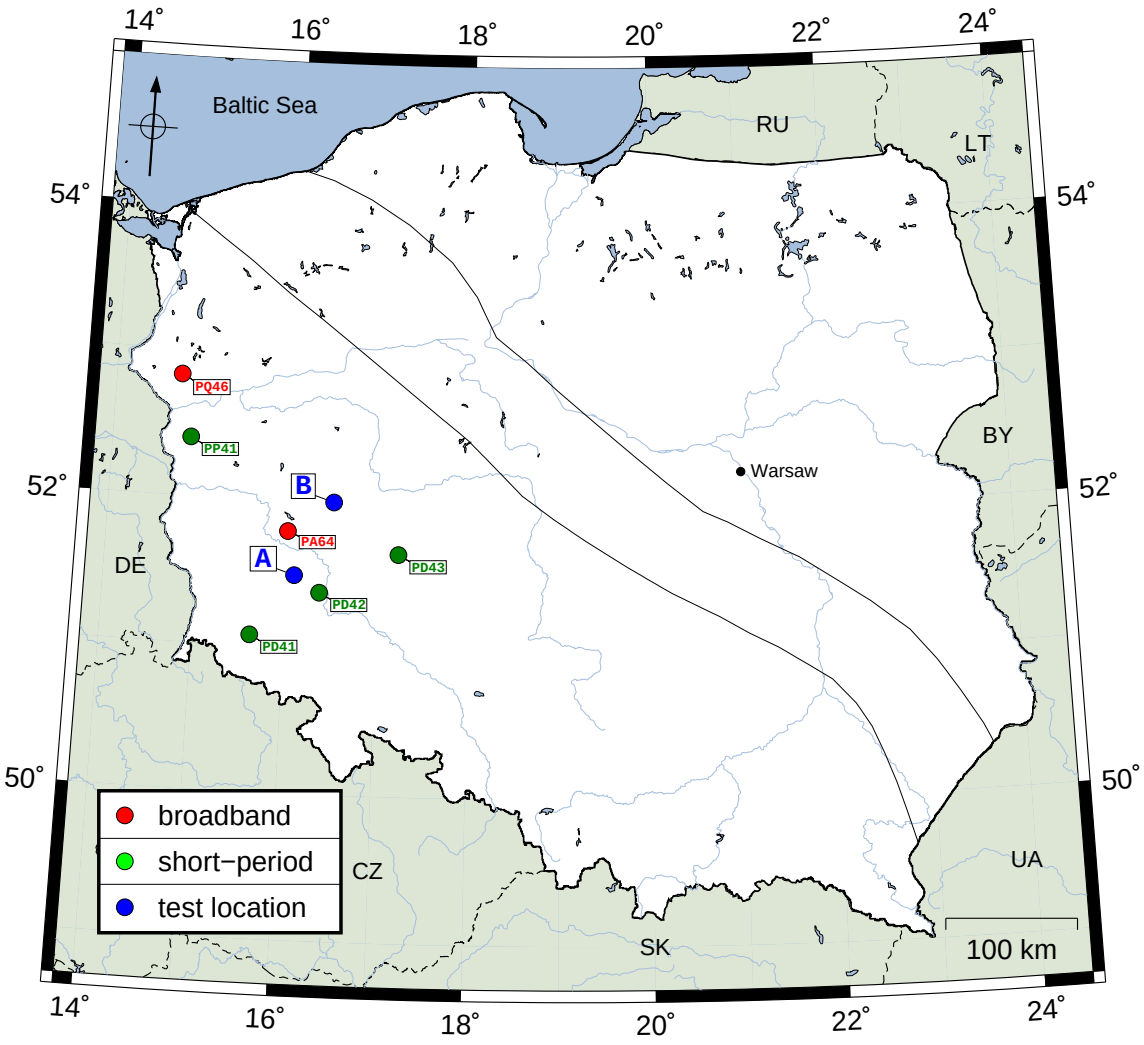

Figure 3.5
Grid search example: location of six PASSEQ 2006-2008 stations (green and red dots) and two test locations (A, B).

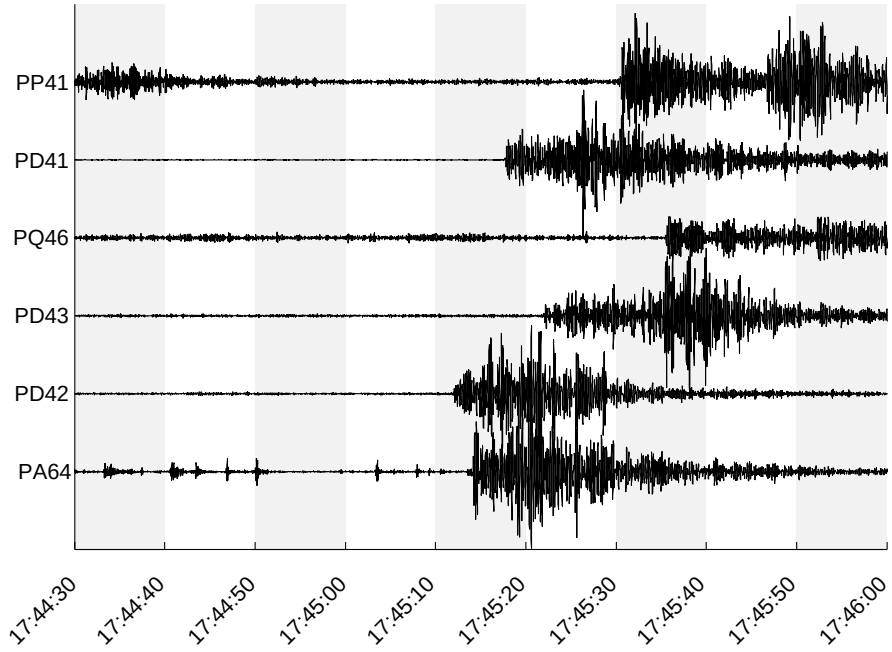

Figure 3.6
Grid search example: recordings of six PASSEQ 2006-2008 stations. X-axis shows original time of recording.

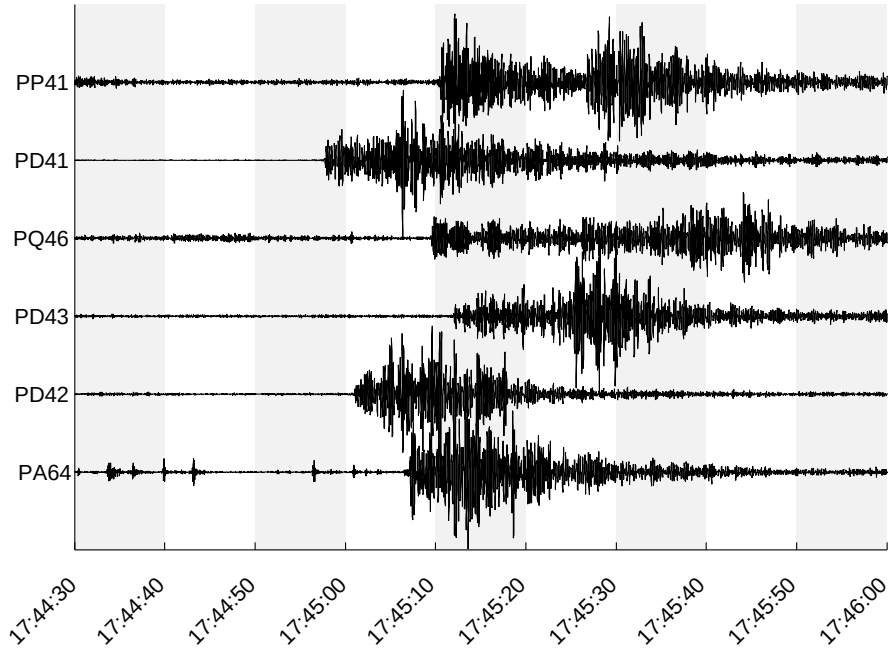

Figure 3.7
Grid search example: recordings for location B of six PASSEQ 2006-2008 stations. X-axis shows shifted time of recording.

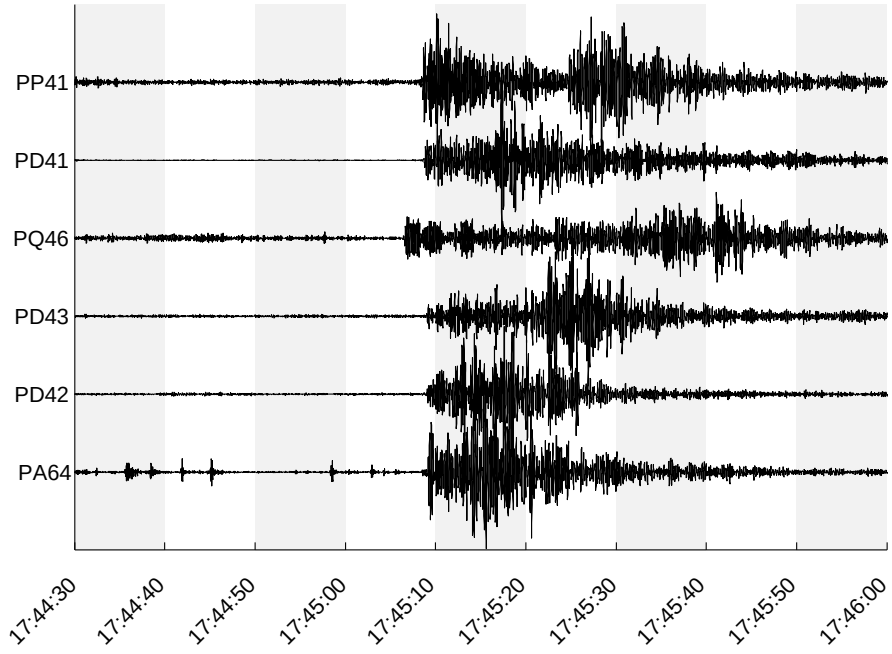

Figure 3.8
Grid search example: recordings for location A. Other explanations like in Figure 3.7.

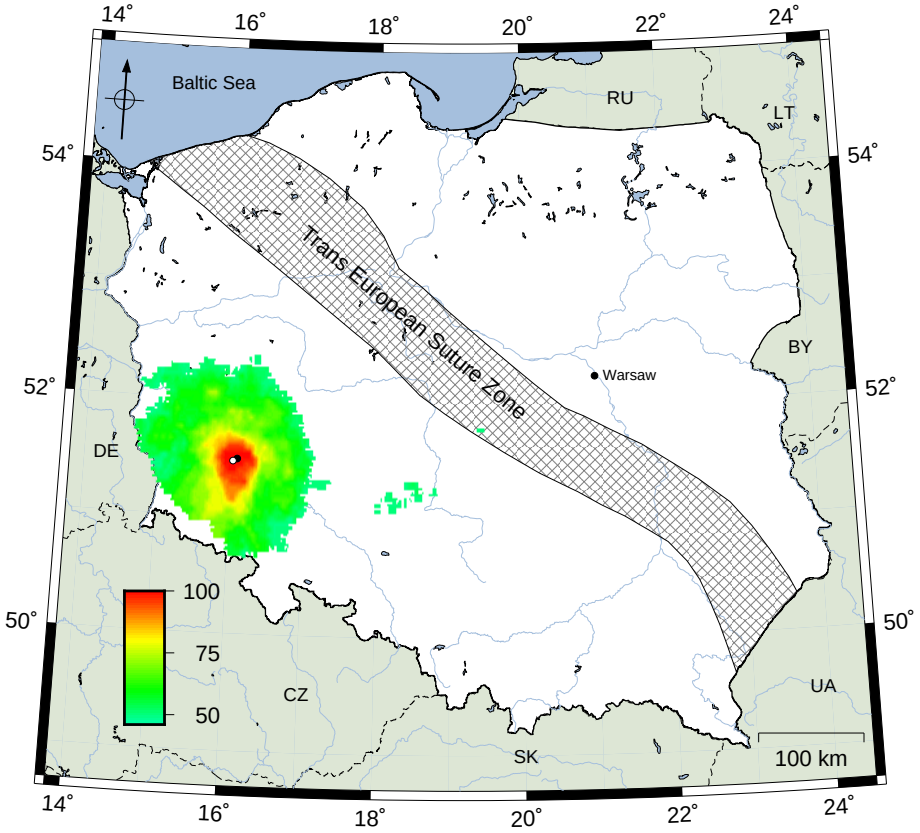

Figure 3.9
Grid search example: a probability map of the grid search for analyzed M3.5 event from Lubin. Color represents percent of stations in 150 km radius that recorded signal at the same time. Black dot shows event epicenter from Local Bulletin (IGF P.A.S.) and white dot shows calculated epicenter.

Chapter 4

Results

4.1 Event selection

After analyzing whole PASSEQ 2006-2008 data set for Poland over 3500 events were automatically detected. This list was then filtered to eliminate noise-only detections. Filtration was done by eliminating events visible in less than 20 grid cells.

New list consisted of 1206 events was analyzed. Events in the area of Lubin and Śląsk were not analyzed due to high and known level of seismicity in this areas. These events were used for determining method accuracy (detection and location). Only 46 events from other areas of Poland were on the list.

Figure 4.1 shows all 1206 possible events. 46 events out of Lubin where analyzed separately.

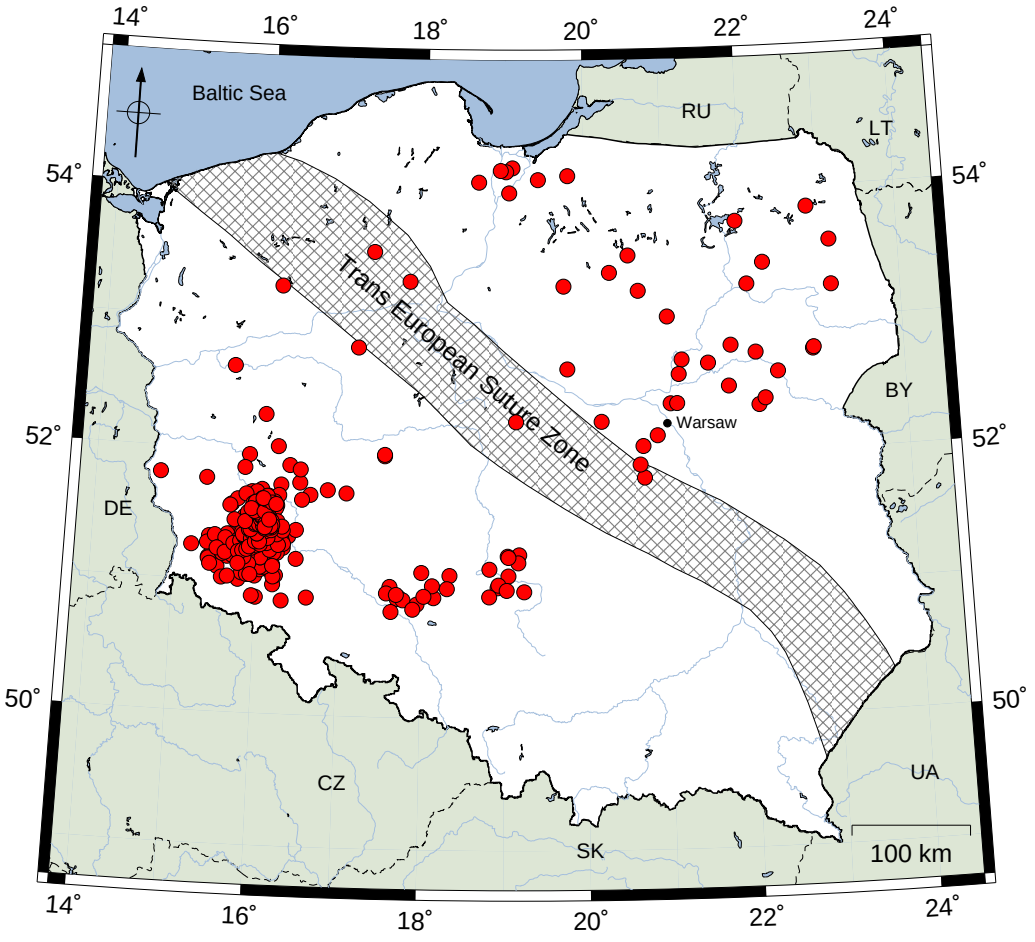

Figure 4.1
Location of epicenters of possible 1206 events based on automated detection and grid search method.

4.2 Event detection and location accuracy

Two aspects of accuracy were discussed for selected time period from August 1st, 2008 to October 31st, 2008. First one is a detection efficiency: how many events are detected out of known events from station and local bulletins. Periods where number of operating stations was insufficient for grid search algorithm were omitted.

Expected level of 80% efficiency was achieved for this period of time. 20% of events were not detected for multiple reasons including:

- high noise on station,
- occurring at the same time as bigger teleseismic event,
- small magnitude of event (under M1.9),
- other, unknown.

Location of bigger events (M>2.8) from Local Bulletins was compared with location given by grid search algorithm. The location difference varies from 1 to 28 km with mean value of 10 km. A histogram of location differences for 53 events is shown in Figure 4.2.

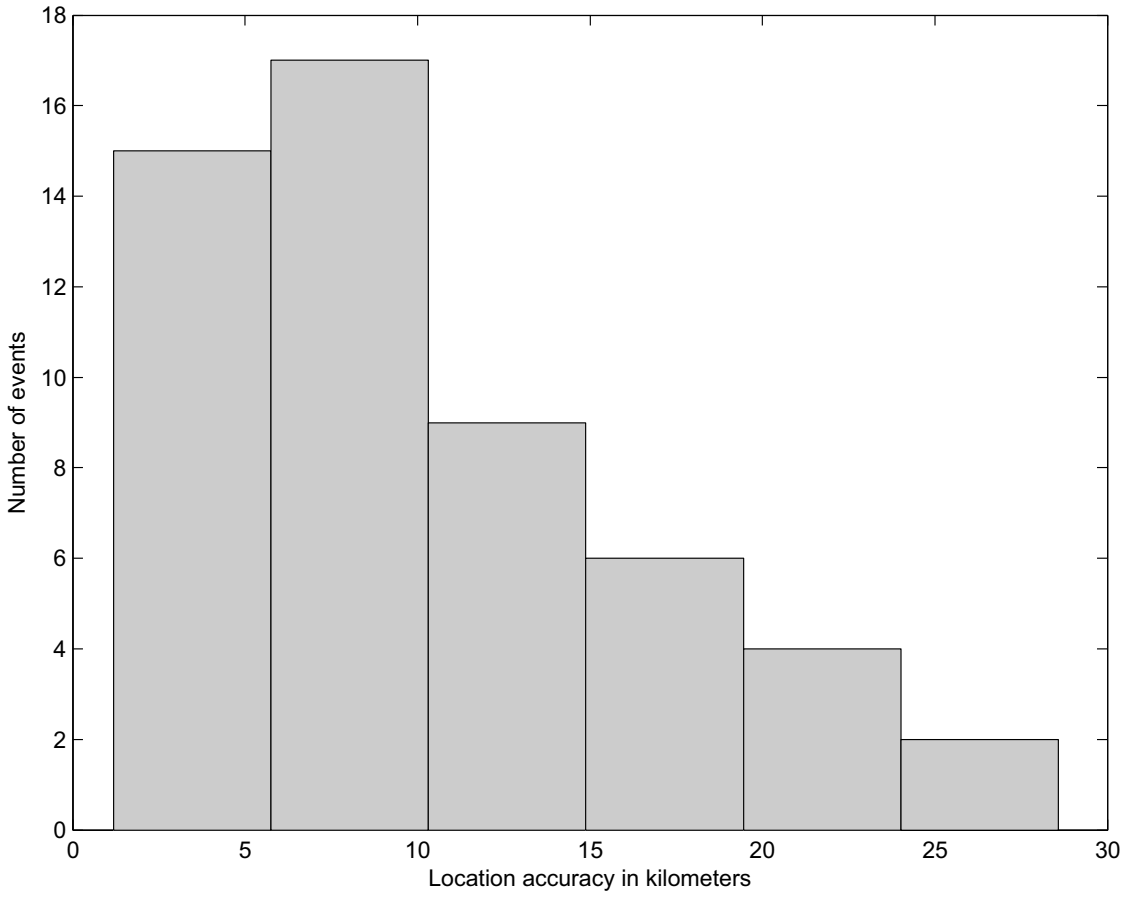

Figure 4.2
Histogram of event location accuracy (difference between bulletin and grid search location). Total of 53 Lubin events were analyzed from August 1st 2006 to October 31st 2006.

4.3 Detected events

46 possible events out of Lubin were manually analyzed:

- 30 were accidental detection of bigger known event,
- 12 were just accidental noise coincidence,
- 4 were real seismic events.

4.3.1 Event near Jarocin, May 6th 2007

Unlisted, strong seismic event was located near Jarocin (52.02°N, 17.48°E). Its magnitude was estimated to 2.5 by comparison to Lubin events. Event occurred in May 6th, 2007 around 07:32:30 UTC.

There was a strong event of M3.9 (52.026°N, 17.529°E) almost in the same place in January 6th, 2012. Both events were probably caused by tectonic stress. Jarocin is located about 100 km from the TESZ border, so this evens can be connected to this area. Grid search result is shown in Figure 4.3. Figures 4.4 and 4.5 show seismograms of event at neighbor stations. See figure descriptions for more information.

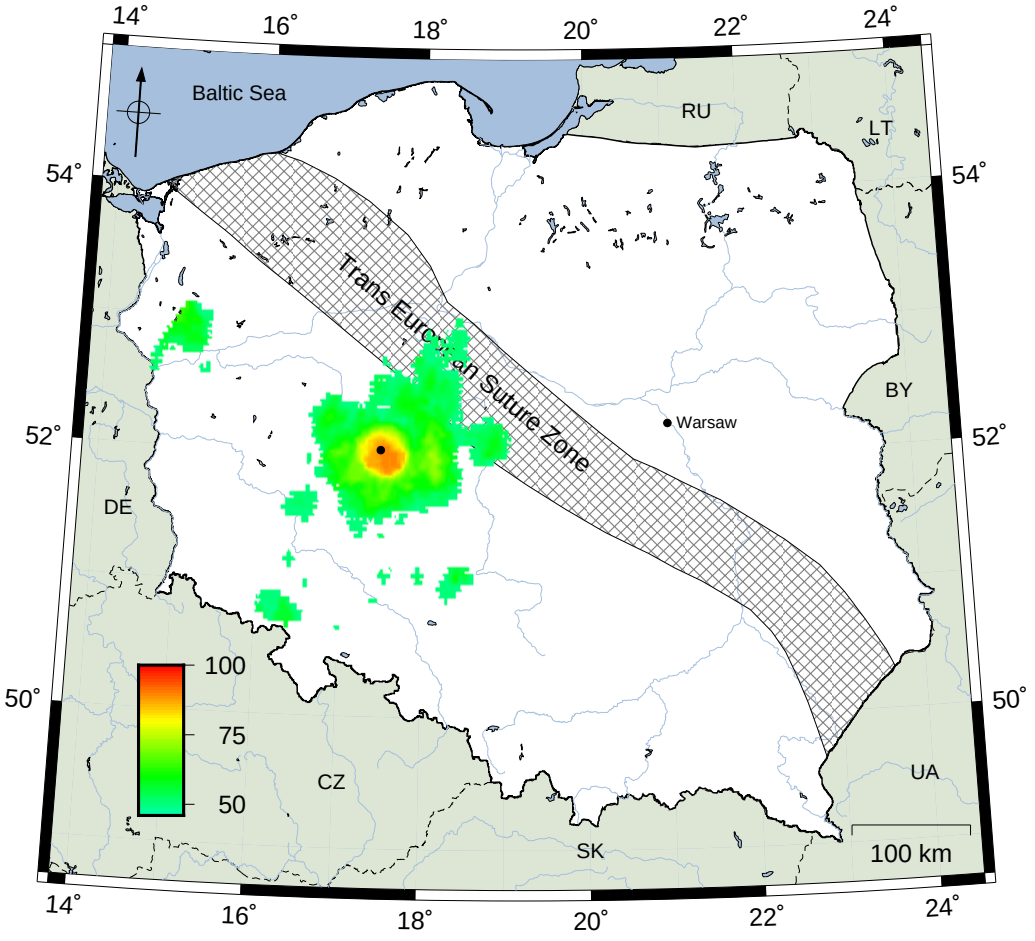

Figure 4.3
A grid probability map of Jarocin event, May 6th, 2007. Black dot shows calculated coordinates of epicenter.

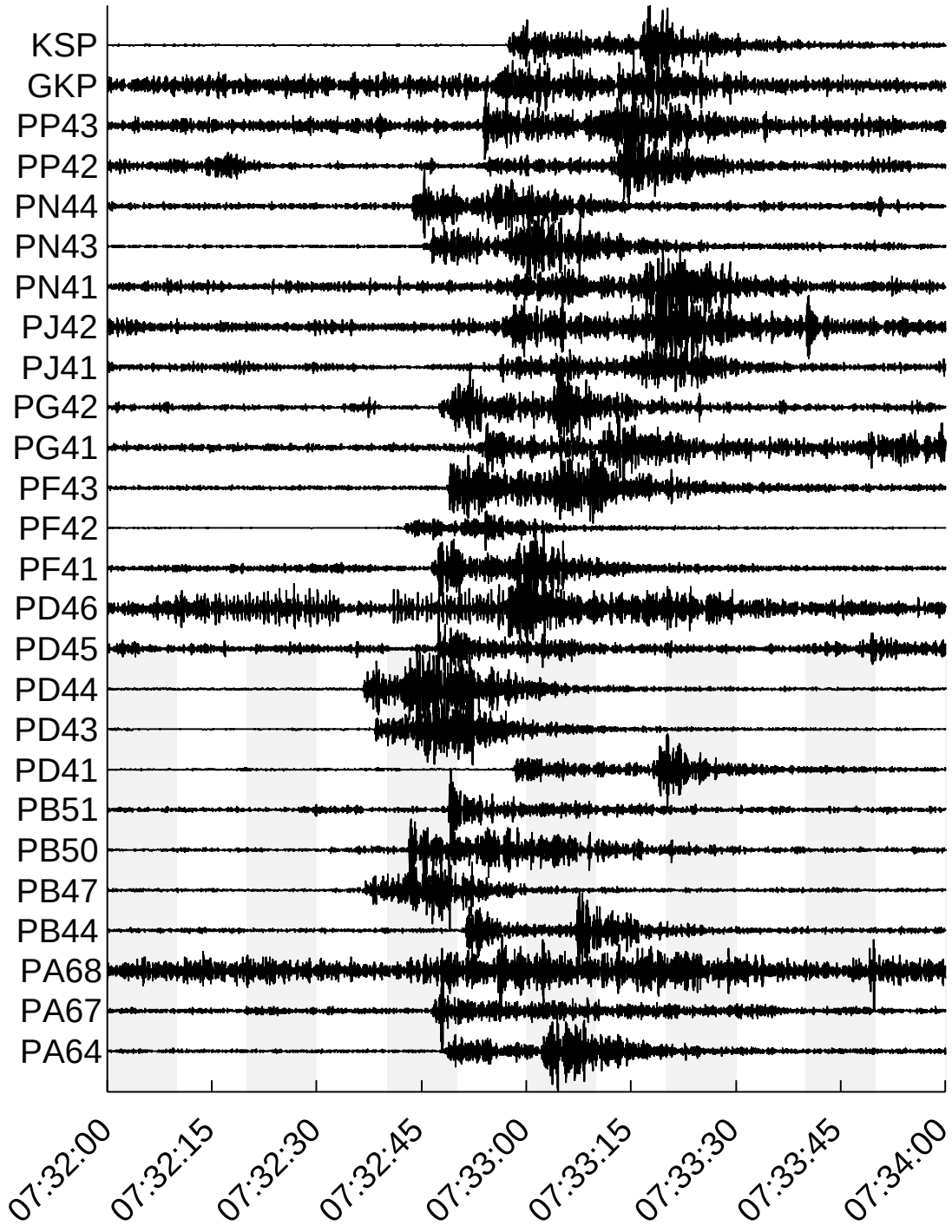

Figure 4.4
Vertical seismic traces of Jarocin event, May 6th, 2007 with original time at stations. All stations in radius of 150km from source are shown (codes of stations on the left).

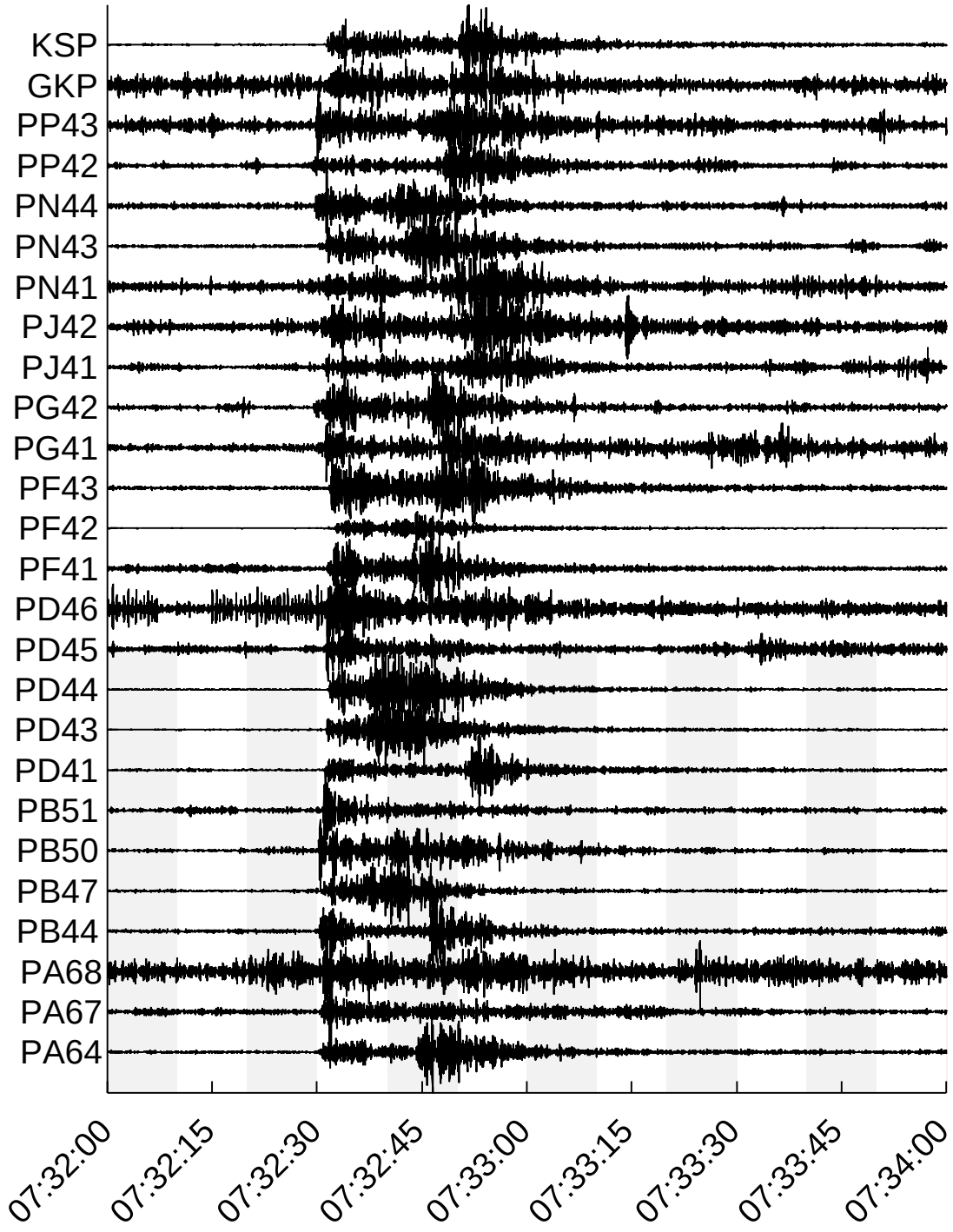

Figure 4.5
Vertical seismic traces of Jarocin event, May 6th, 2007 with shifted time for calculated source coordinates (52.02°N, 17.48°E). All detection stations in radius of 150km from source are shown (codes of stations on the left).

4.3.2 Event in Baltic Sea, September 12th 2006

Unknown event was detected in the Bay of Gdańsk. Grid search was performed only on land, so location of this event was found manually (manual grid searching): 54.55°N, 19.32°E. Time of event was determined to 15:12:14 UTC.

Figure 4.6 shows event location and Figure 4.7 shows time shifted traces of PASSEQ 2006-2008 stations around.

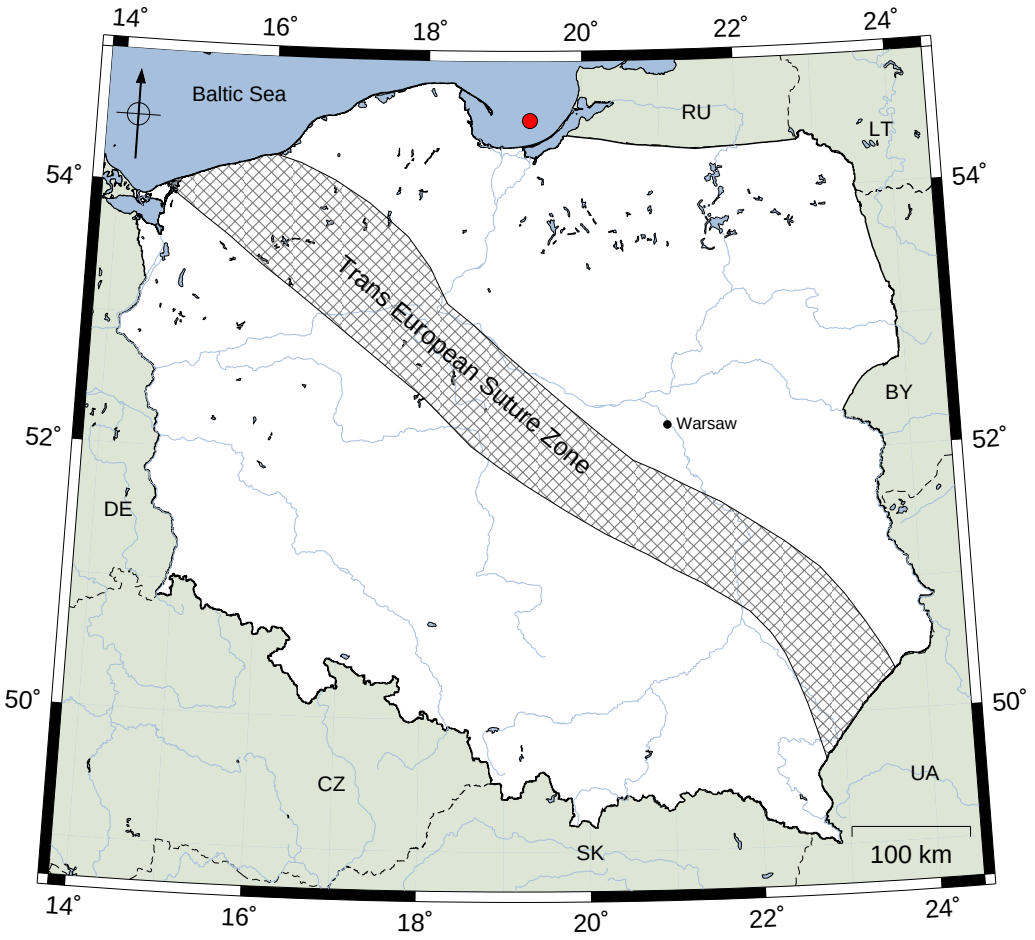

Figure 4.6
Location of event in Baltic Sea (Bay of Gdańsk), September 12th 2006.

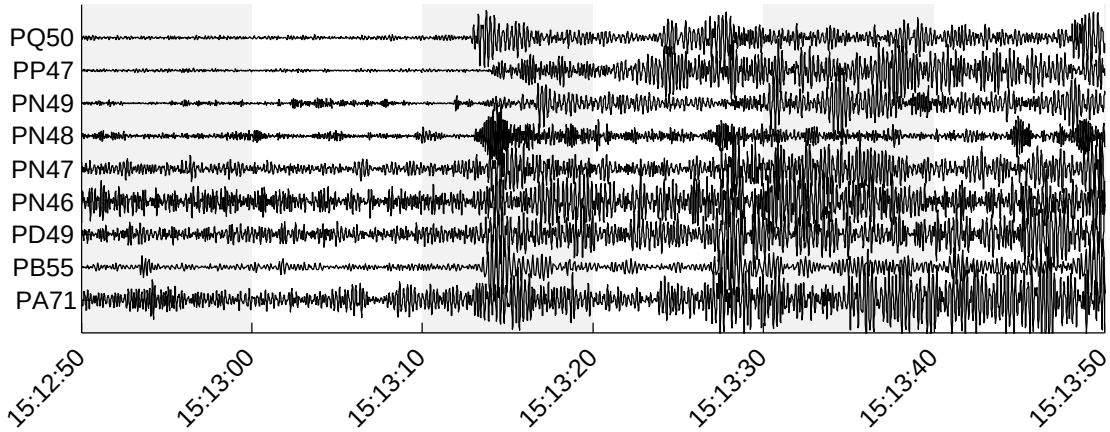

Figure 4.7
Vertical seismic recording of event in Baltic Sea (Bay of Gdańsk), September 12th 2006.

4.3.3 Event in Baltic Sea, March 20th 2007

Third event was also detected in the Bay of Gdańsk. Grid search was performed only on land, so location of this event was found manually (manual grid searching): 54.60°N, 18.75°E. Time of event was determined to 23:08:51 UTC. This event was listed in emsc-csem.org catalog but was located in Gotland, Sweden. Magnitude was not estimated.

Figure 4.8 shows event location and Figure 4.9 shows time shifted seismic records of PASSEQ 2006-2008 stations around.

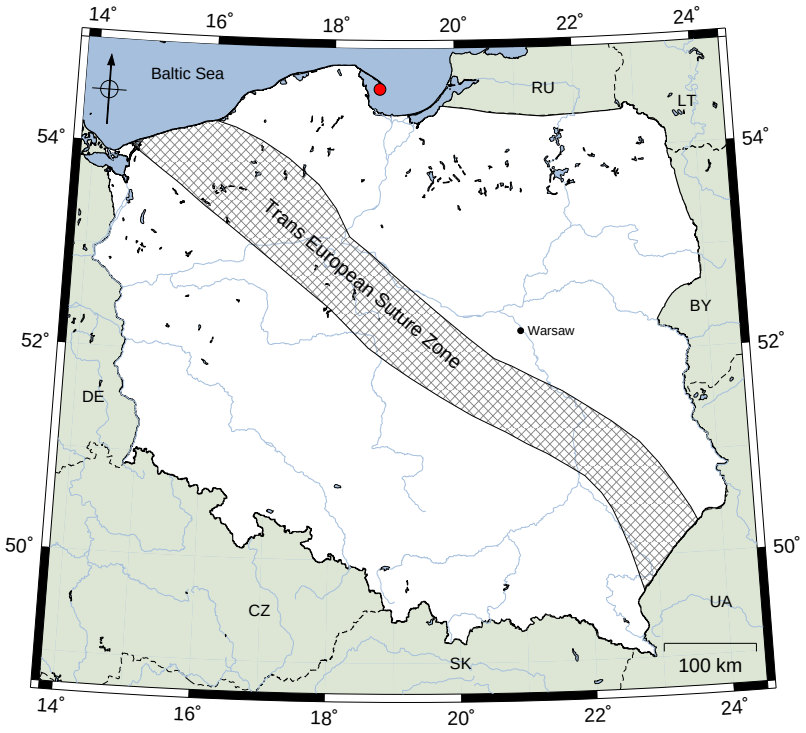

Figure 4.8
Location of event in Baltic Sea (Bay of Gdańsk), March 20th 2007.

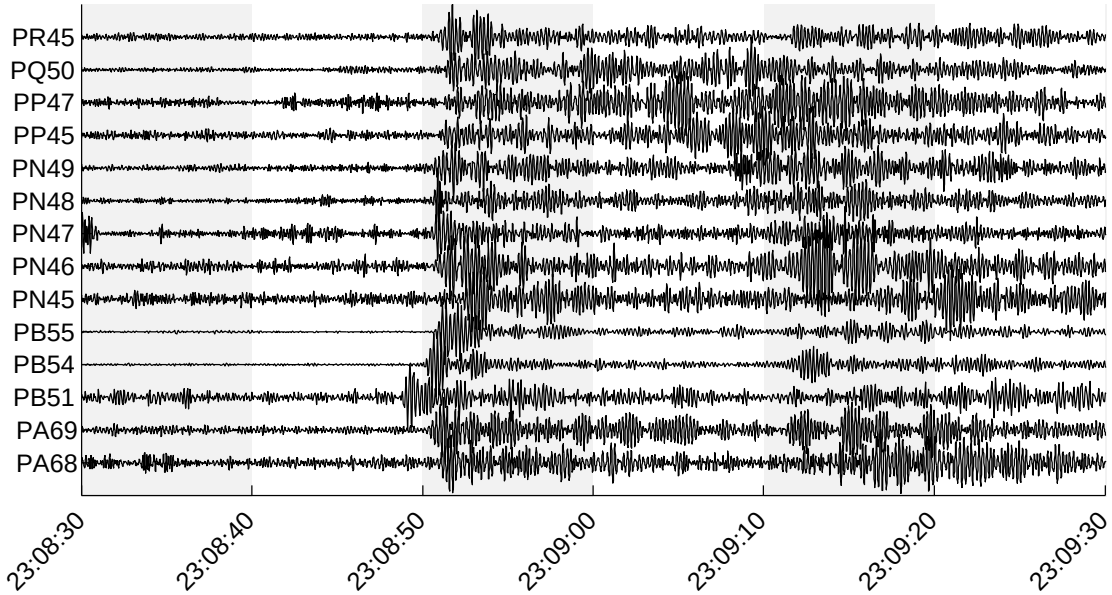

Figure 4.9
Vertical seismic recording of event in Baltic Sea (Bay of Gdańsk), March 20th 2007.

4.3.4 Event in Baltic Sea, May 2nd 2007

Last event was also detected in the Bay of Gdańsk. Grid search was performed only on land, so location of this event was found manually (manual grid searching): 54.69°N, 19.17°E. Time of event was determined to 07:08:23 UTC. This event was listed in emsc-csem.org catalog but it was located in the middle of Baltic Sea. Magnitude was not estimated.

Figure 4.10 shows event location and Figure 4.11 shows time shifted seismic records of PASSEQ 2006-2008 stations around.

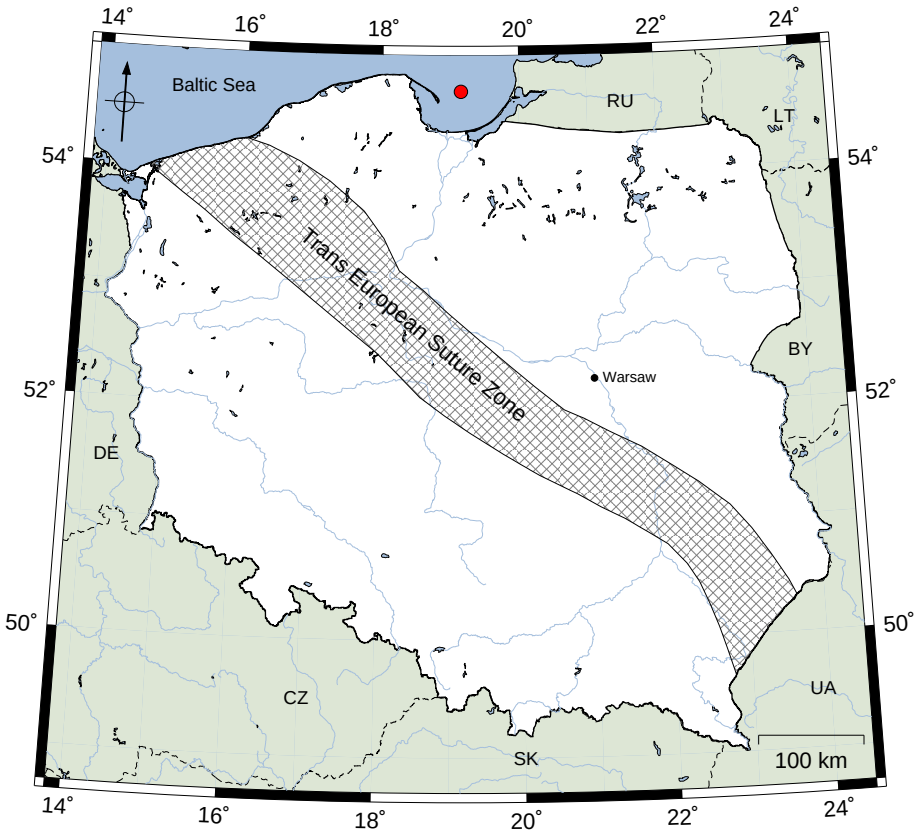

Figure 4.10
Location of event in Baltic Sea (Bay of Gdańsk), May 2nd 2007.

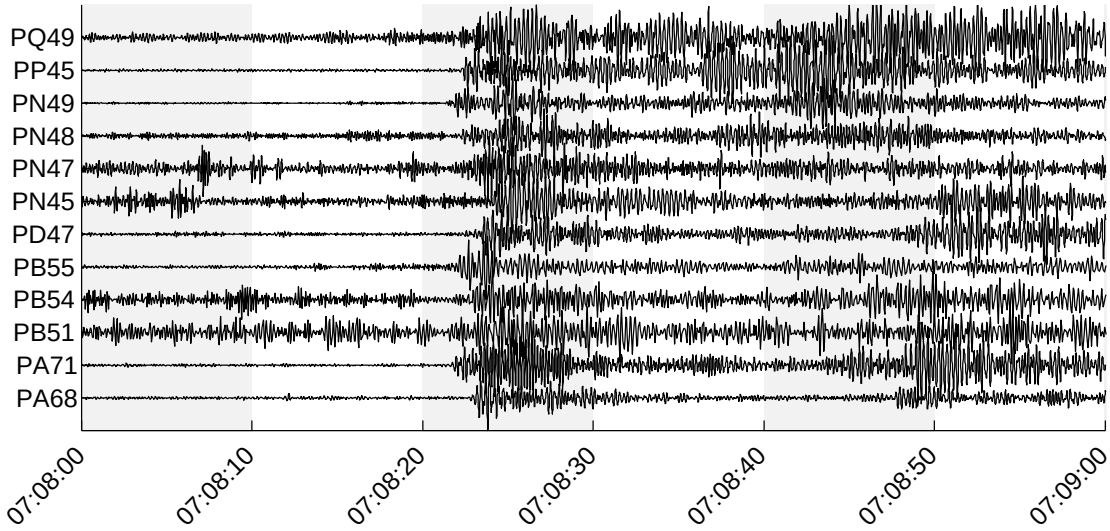

Figure 4.11
Vertical seismic recording of event in Baltic Sea (Bay of Gdańsk), May 2nd 2007.

Chapter 5

Summary

PASSEQ 2006-2008 data set for Poland was processed to detect local seismic events in Poland. Carl Johnson's STA/LTA detection algorithm was used with grid search method. All significant programs and script required for this analysis were prepared specially for purposes of this paper. The list of 1206 possible seismic events were prepared (Figure 4.1). 46 events from regions with unknown seismicity were selected for detailed analysis. Finally 4 events were confirmed detection (others were effect of bigger events or seismic noise, Figure 5.1):

- Event in Baltic Sea, September 12th 2006 15:12:14 UTC, 54.55°N, 19.32°E,
- Event in Baltic Sea, March 20th 2007 23:08:51 UTC, 54.60°N, 18.75°E,
- Event in Baltic Sea, May 2nd 2007 07:08:23 UTC, 54.69°N, 19.17°E,
- Event near Jarocin, May 6th 2007 07:32:30 UTC, 52.02°N, 17.48°E.

Event near Jarocin was clearly caused by tectonic stress (similar, stronger event happened in this area in 2012) while three events in Baltic Sea may have been caused by military exercises of Polish Navy [Meyer and Kulhanek, 1981, Wiejacz and Debski, 2001].

No events in the area of TESZ were detected. Recording quality and distribution of stations made detection of small events (under M1.8) impossible. This allows to formulate paper conclusion:

There was no significant seismic activity in the area of the Trans European Suture Zone during PASSEQ 2006-2008 experiment.

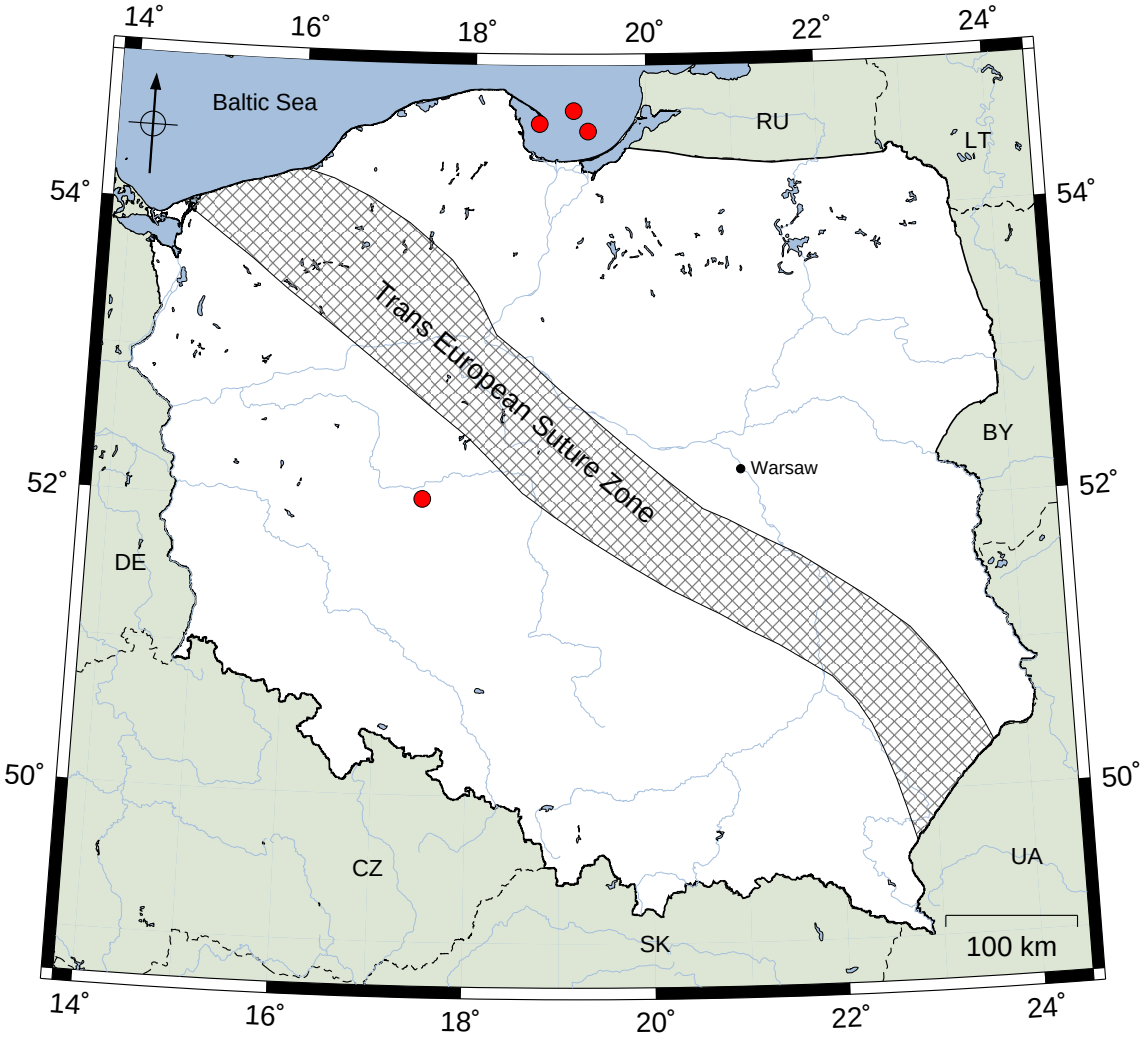

Figure 5.1
Location of four detected and analyzed seismic events.

Acknowledgments

I would like to express my gratitude to the following people:

- Dr Monika Wilde-Piórko for her excellent supervision, patience, support and for allowing me to work and expand my knowledge at Lithospheric Physics Department,
- Prof. Marek Grad for profound review,
- Dr Timo Tiira for excellent ideas,
- Prof. Andrzej Wysmołek for great patience presented to all students,
- Prof. Katarzyna Chałasińska-Macukow for support
- and anybody I missed who deserves a mention.

I would also thank people who created great and free programs and tools:

- ObsPy - a Python Toolbox for seismology/seismological observatories [Megies et al., 2011, Beyreuther et al., 2010] - set of python based functions that helped me master PASSEQ 2006-2008 data set,
- Generic Mapping Toolbox GMT5 [Wessel and Smith, 1991] - set of tools that made my maps look so nice,
- The TauP Toolkit - seismic travel time calculator [Crotwell et al., 1999] - fast, easy and useful tool.

Appendix A

Bulletins of Polish seismic observatories

Two types of data sources about local seismic events exist in Poland:

1. Station Bulletins - contain lists of most events with determined time of seismic phases, but does not provide information about event locations. Example of Station Bulletin is shown in Listing A.1,
2. Local bulletins - contain lists of bigger local events in Poland with their location. Example of Local Bulletin is shown in Listing A.2.

Both bulletins are prepared by Institute of Geophysics P.A.S. in Warsaw. They are available at institute web page: http://igf.edu.pl/.

Listing A.1
Fragment of KSP Station Bulletin in August 1st 2006

```
KSP Aug 2006

    Aug01                           Mm = 2.2  Lubin          d=79km
         ePg        Z  02 41 02.7
         eSg       E       41 12.2

    Aug01                           Mm = 2.1  Lubin          d=63km
         ePg        Z  03 02 11.3
         eSg       E       02 18.9

    Aug01                           Mm = 1.6  Walbrzych Reg. d=41km
         ePg       E   05 09 20.6
         eSg      N        09 25.7

    Aug01                           Mm = 2.5  Silesia        d=204km
         ePg      N    05 24 24.5
         eSg       E       24 48.4

    Aug01                           Mm = 1.9  Walbrzych Reg. d=66km
         ePg        Z  08 27 43.2
         eSg       E       27 51.3

    Aug01                           Mm = 2.1  Walbrzych Reg. d=34km
         ePg        Z  08 39 43.5
         eSg      N        39 47.6

    Aug01                           Mm = 2.2  Silesia        d=174km
         ePg        Z  08 51 57.2
         eSg      N        52 17.7

    Aug01                           Mm = 1.9                 d=150km
         ePg        Z  09 11 52.1
         eSg      N        12 10.2
```

Listing A.2
Fragment of Local Bulletin in January 17th 2007

```
JAN 17
         fi = 51.481°N, la = 16.101°E
         H = 17:27:39.7 , M = 3.1

   KSP   d=  72.2 km
         Pg eZ         17 27 51.84
         Sg eE         17 28 00.54
             Z         17 28 03.9

   OJC   d= 295.0 km
         Pg eZ         17 28 29.8
         Sg eN         17 29 04.9
   NIE   d= 376.1 km
         P  eZ         17 28 42.8
         S  eN         17 29 27.1
   RAC   d= 214.0 km
         Pg eZ         17 28 14.4
         Sg eE         17 28 40.3
```

Appendix B

Carl Johnson's STA/LTA performance

ObsPy implementation of Carl Johnson's STA/LTA is available as standard python function in ObsPy.signal toolbox. Its source code is shown in Listing B.1. In this implementation moving average is calculated by repeating summation of all elements in window. If trace is 1-hour long with 100Hz sampling and average window is 32 seconds, average is calculated 356800 times and each time 3200 values are summed.

To increase performance other method of calculating moving average was used: a sum of values in window is kept in memory and for every window move one value is subtracted (went out of window) and one value is added (new in window).

This combined with switching to C++ allowed to increase speed of detection from 20 second per trace to 0.1 seconds per trace.

Source of new implementation of Carl Johnson's STA/LTA is shown in Listing B.2. This python module should be compiled for every machine separately (32 and 64 bit).

Listing B.1
Source code of original Carl Johnson's STA/LTA from ObsPy toolbox

```
def carlSTATrig(a, nsta, nlta, ratio, quiet):
    """
    Computes the carlSTATrig characteristic function.

    eta = star - (ratio * ltar) - abs(sta - lta) - quiet

    :type a: NumPy ndarray
    :param a: Seismic Trace
    :type nsta: Int
    :param nsta: Length of short time average window in samples
    :type nlta: Int
    :param nlta: Length of long time average window in samples
    :type ration: Float
    :param ratio: as ratio gets smaller, carlSTATrig gets more sensitive
    :type quiet: Float
    :param quiet: as quiet gets smaller, carlSTATrig gets more sensitive
    :rtype: NumPy ndarray
    :return: Characteristic function of CarlStaTrig
    """
    m = len(a)
    #
    sta = np.zeros(len(a), dtype='float64')
    lta = np.zeros(len(a), dtype='float64')
    star = np.zeros(len(a), dtype='float64')
    ltar = np.zeros(len(a), dtype='float64')
    pad_sta = np.zeros(nsta)
    pad_lta = np.zeros(nlta)  # avoid for 0 division 0/1=0
    #
    # compute the short time average (STA)
    for i in xrange(nsta):  # window size to smooth over
        sta += np.concatenate((pad_sta, a[i:m - nsta + i]))
    sta /= nsta
    #
    # compute the long time average (LTA), 8 sec average over sta
```

```
    for i in xrange(nlta):  # window size to smooth over
        lta += np.concatenate((pad_lta, sta[i:m - nlta + i]))
    lta /= nlta
    lta = np.concatenate((np.zeros(1), lta))[:m]  # XXX ???
    #
    # compute star, average of abs diff between trace and lta
    for i in xrange(nsta):  # window size to smooth over
        star += np.concatenate((pad_sta, abs(a[i:m - nsta + i] - lta[i:m - nsta + i
            ])))
    star /= nsta
    #
    # compute ltar, 8 sec average over star
    for i in xrange(nlta):  # window size to smooth over
        ltar += np.concatenate((pad_lta, star[i:m - nlta + i]))
    ltar /= nlta
    #
    eta = star - (ratio * ltar) - abs(sta - lta) - quiet
    eta[:nlta] = -1.0
    return eta
```

Listing B.2
Source code of C++ implementation (Python module) of modified Carl Johnson's STA/LTA algorithm

```
#include <Python.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double *trace;
double *sta;
double *lta;
double *star;
double *ltar;
double *tmp;
double *eta;


// Function for calculating average using new algorithm:
void srednia(double *input, double *output, int length, int window)
{
    double sum = 0;
    int i, j;
    for(j=0;j<window;j++)
    {
        sum += input[window-j-1];
    }
    output[window-1]=sum/(double)window;

    for(i = window; i < length; i++)
    {
        sum = sum + input[i] - input[i-window];
        output[i]=sum/(double)window;
    }
}

// Main function
static PyObject* compute(PyObject* self, PyObject* args)
{
    PyObject * listObj;
    int nsta;
    int nlta;
    double ratio;
    double quiet;
    double sum = 0.0;
    if (! PyArg_ParseTuple( args, "O!iidd", &PyList_Type, &listObj, &nsta, &nlta, &
        ratio, &quiet)) return NULL;
    int N = PyList_Size(listObj);

    trace = (double*)malloc(N*sizeof(double));
    sta = (double*)malloc(N*sizeof(double));
    lta = (double*)malloc(N*sizeof(double));
    star = (double*)malloc(N*sizeof(double));
    ltar = (double*)malloc(N*sizeof(double));
    eta = (double*)malloc(N*sizeof(double));
    tmp = (double*)malloc(N*sizeof(double));
```

```
52
53      int i;
54      for(i = 0; i<N; i++)
55      {
56          sta[i] = 0;
57          lta[i] = 0;
58          star[i] = 0;
59          ltar[i] = 0;
60          eta[i] = 0;
61          tmp[i] = 0;
62      }
63
64      for(i=0;i<N;i++)
65      {
66          trace[i] = (double) PyFloat_AsDouble(PyList_GetItem(listObj, i));
67          sum += trace[i];
68      }
69
70
71
72      srednia(trace, sta, N, nsta);
73      srednia(sta, lta, N, nlta);
74      for(i = 0; i < N; i++)
75      {
76          if(i < nlta)
77              lta[i]=0;
78          tmp[i] = abs(trace[i]-lta[i]);
79      }
80      srednia(tmp,star,N,nsta);
81      srednia(star, ltar, N, nlta);
82
83      for(i = 0; i < N; i++)
84      {
85          if(i > nlta)
86          {
87              eta[i] = star[i] - (ratio * ltar[i]) - abs(sta[i] - lta[i]) - quiet;
88          }
89          else
90          {
91              eta[i] = -1;
92          }
93      }
94
95
96
97
98
99      PyObject *result=PyList_New(N);
100     for(i=0;i<N;i++)
101     {
102         PyObject *op = PyFloat_FromDouble((double)eta[i]);
103         PyList_SetItem(result,i,op);
104     }
105
106     return Py_BuildValue("N", result);
107 }
108
109 static char carl_docs[] = "carlStaTrigC( ): no documentation!!\n";
110
111 static PyMethodDef carl_funcs[] = {{"compute", (PyCFunction)compute, METH_VARARGS ,
        carl_docs},{NULL}};
112
113 void initcarlStaTrigC(void)
114 {
115     Py_InitModule3("carlStaTrigC", carl_funcs, "Carl Sta Trig Module!");
116 }
```

Bibliography

M. Beyreuther, R. Barsch, L. Krischer, T. Megies, Y. Behr, and J. Wassermann. ObsPy: A Python Toolbox for Seismology. *SRL*, 81(3):530–533, may 2010. doi: 10.1785/gssrl.81.3.530.

H. P. Crotwell, T. J. Owens, and J. Ritsema. The taup toolkit: Flexible seismic travel-time and ray-path utilities. *Seismological Research Letters*, 70(2):154–160, 1999. doi: 10.1785/gssrl.70.2.154. URL `http://srl.geoscienceworld.org/content/70/2/154.short`.

B. Guterch. Sejsmiczność polski w świetle danych historycznych. *Przegląd Geologiczny*, 57:513–520, 2009.

B. L. N. Kennett and E. R. Engdahl. Traveltimes for global earthquake location and phase identification. *Geophysical Journal International*, 105(2):429–465, 1991. ISSN 1365-246X. doi: 10.1111/j.1365-246X.1991.tb06724.x. URL `http://dx.doi.org/10.1111/j.1365-246X.1991.tb06724.x`.

T. Megies, M. Beyreuther, R. Barsch, L. Krischer, and J. Wassermann. Obspy – what can it do for data centers and observatories? *Annals of Geophysics*, 54(1), 2011. ISSN 2037-416X. URL `http://www.annalsofgeophysics.eu/index.php/annals/article/view/4838`.

K. Meyer and O. Kulhanek. The gulf of gdańsk seismic event sequence of june 25 - july 3, 1980. *Acta Geophysica Polonica*, 29:315–320, 1981.

T.C. Pharaoh. Palaeozoic terranes and their lithospheric boundaries within the trans-european suture zone (tesz): a review. *Tectonophysics*, 314(1–3):17 – 41, 1999. ISSN 0040-1951. doi: 10.1016/S0040-1951(99)00235-8. URL `http://www.sciencedirect.com/science/article/pii/S0040195199002358`.

M. S. Sambridge and B. L. N. Kennett. A novel method of hypocentre location. *Geophysical Journal of the Royal Astronomical Society*, 87(2):679–697, 1986. ISSN 1365-246X. doi: 10.1111/j.1365-246X.1986.tb06644.x. URL `http://dx.doi.org/10.1111/j.1365-246X.1986.tb06644.x`.

P. Wessel and W. H. F. Smith. Free software helps map and display data. *Eos Trans. AGU*, 72 (41):441–446, 1991. ISSN 0096-3941. URL `http://dx.doi.org/10.1029/90EO00319`.

P. Wiejacz and W. Debski. New observations of gulf of gdansk seismic events. *Physics of the Earth and Planetary Interiors*, 123(2–4):233 – 245, 2001. ISSN 0031-9201. doi: 10.1016/S0031-9201(00)00212-0. URL `http://www.sciencedirect.com/science/article/pii/S0031920100002120`. <ce:title>Tomographic Imagining of 3-D Velocity Structure and Accurate Earthquake Location,</ce:title>.

M. Wilde-Piórko, W. H. Geissler, J. Plomerová, M. Grad, V. Babuška, E. Brückl, J. Cyziene, W. Czuba, R. England, E. Gaczyński, R. Gazdova, S. Gregersen, A. Guterch, W. Hanka, E. Hegedüs, B. Heuer, P. Jedlička, J. Lazauskiene, G. R. Keller, R. Kind, K. Klinge, P. Kolinsky, K. Komminaho, E. Kozlovskaya, F. Krüger, T. Larsen, M. Majdański, J. Málek, G. Motuza, O. Novotný, R. Pietrasiak, Th Plenefisch, B. Růžek, S. Sliaupa, P. Środa, M. Świeczak, T. Tiira, P. Voss, and P. Wiejacz. Passeq 2006-2008: Passive seismic experiment in trans-european suture zone. *Studia Geophysica et Geodaetica*, 52(3):439–448, 2008. URL `www.scopus.com`. Cited By (since 1996): 6.