Realistic Seismic Signal Detection Simulation for GMS Testing

CONTENTS

Realistic Seismic Signal Detection Simulation for GMS Testing

Abstract

This paper presents a comprehensive framework for generating realistic seismic signal detection data for testing Geophysical Monitoring System (GMS) software. We define mathematical models for waveform generation, signal detection algorithms, and data structures that match the actual SNL-GMS implementation. The simulator provides continuous waveform data at configurable sample rates, realistic signal detections with phase identification, and event associations following seismic analysis standards.

Keywords: seismic simulation, signal detection, GMS, waveform generation, testing framework

1. Introduction

1.1 Motivation

Testing seismic monitoring systems requires realistic data that exercises all components of the signal processing pipeline. Current mock implementations provide only basic metadata without actual waveforms, limiting the effectiveness of integration testing. This work defines a realistic simulation framework that generates:

1. Continuous seismic waveforms at configurable sample rates
2. Signal detection objects with phase identification
3. Event associations and location estimates
4. Processing configuration parameters

1.2 Scope

This paper covers: - Seismic waveform generation - Signal detection algorithms (STA/LTA) - Phase identification - Event association methodology - Data structure specifications - Implementation guidelines

2. Seismic Waveform Generation

2.1 Continuous Waveform Model

2.1.1 Mathematical Model

The seismic waveform (w(t)) is modeled as:

[w(t) = s(t) + n(t)]

where: - (s(t)) = seismic signal (P-wave, S-wave, surface waves) - (n(t)) = background noise

2.1.2 Seismic Signal Components

The seismic signal is composed of:

[s(t) = s_P(t) + s_S(t) + s_{surface}(t)]

P-wave (Primary wave):

[s_P(t) = A_P e^{-_P (t - t_P)} (2f_P (t - t_P)) H(t - t_P)]

where: - (A_P) = P-wave amplitude - (_P) = P-wave attenuation coefficient (0.5-2.0 s(^{-1})) - (f_P) = P-wave dominant frequency (0.5-5 Hz) - (t_P) = P-wave arrival time - (H(t)) = Heaviside step function

S-wave (Secondary wave):

[s_S(t) = A_S e^{-_S (t - t_S)} (2f_S (t - t_S)) H(t - t_S)]

where: - (A_S) = S-wave amplitude (typically 1.5-3× P-wave amplitude) - (_S) = S-wave attenuation coefficient (0.3-1.5 s(^{-1})) - (f_S) = S-wave dominant frequency (0.2-2 Hz) - (t_S = t_P + -) (S-wave arrival time)

Surface waves (Love and Rayleigh):

[s_{surface}(t) = A_L e^{-_L (t - t_L)} (2f_L (t - t_L)) H(t - t_L) + A_R e^{-_R (t - t_R)} (2f_R (t - t_R)) H(t - t_R)]

2.1.3 Background Noise Model

Background noise follows a colored Gaussian process:

[n(t) = _{i=1}^{N} a_i (2f_i t + _i)]

with: - (f_i) = frequencies from 0.01 to 20 Hz - (a_i) = amplitudes from noise spectrum - (_i) = random phase in [0, 2π)

The noise power spectral density follows:

[P(f) = P_0 ()^{-}]

where: - (P_0) = reference power level - (f_0) = reference frequency (1 Hz) - ( ) = spectral slope (typically 2-4)

2.2 Station-Specific Parameters

Each station has unique characteristics:

```
        {
          "stationId": "STA001",
          "networkType": "IU",
          "location": {
            "latitude": 35.9275,
            "longitude": -106.4572,
            "elevation": 1845.0
          },
          "channels": [
            {
              "name": "BHZ",
              "type": "BROADBAND",
              "orientation": "VERTICAL",
              "sampleRate": 40.0,
              "response": {
                "poles": [-0.03661, -7.549, -7.549],
                "zeros": [0.0, 0.0, 0.0],
                "gain": 2946.6
              },
              "sensitivity": 1.589e9
            },
            {
              "name": "BHN",
              "type": "BROADBAND",
              "orientation": "NORTH",
              "sampleRate": 40.0
            },
            {
              "name": "BHE",
```

```
            "type": "BROADBAND",
            "orientation": "EAST",
            "sampleRate": 40.0
          }
        ],
        "noiseModel": {
          "type": "PetersonNLNM",
          "level": "LOW_NOISE"
        }
      }
```

2.3 Waveform Generation Algorithm

```
        func GenerateWaveform(station Station, event Event, dur
time.Duration, sampleRate float64) []float64 {
            // Calculate travel times
            distance := haversine(station.Location, event.Locat
            tP := distance / vP  // P-wave velocity
            tS := distance / vS  // S wave velocity

            // Calculate amplitudes
            AP := event.Magnitude * stationGain(station, event,
            AS := AP * sWaveRatio(station, event)

            // Generate samples
            nSamples := int(duration.Seconds() * sampleRate)
            samples := make([]float64, nSamples)

            for i := 0; i < nSamples; i++ {
                t := float64(i) / sampleRate

                // P-wave
                if t >= tP {
                    samples[i] += AP * math.Exp(-alphaP*(t-tP))
                                  math.Sin(2*math.Pi*fP*(t-tP))
                }

                // S-wave
                if t >= tS {
```

```
                samples[i] += AS * math.Exp(-alphaS*(t-tS)) *
                              math.Sin(2*math.Pi*fS*(t-tS))
            }

            // Surface waves
            tL := tS + distance/vL
            tR := tS + distance/vR
            if t >= tL {
                samples[i] += AL * math.Exp(-alphaL*(t-tL)) *
                              math.Sin(2*math.Pi*fL*(t-tL))
            }
            if t >= tR {
                samples[i] += AR * math.Exp(-alphaR*(t-tR)) *
                              math.Sin(2*math.Pi*fR*(t-tR))
            }

            // Add noise
            samples[i] += generateNoise(frequency, t)
        }

        return samples
    }
```

3. Signal Detection Algorithms

3.1 STA/LTA (Short-Term Average / Long-Term Average)

3.1.1 Algorithm Description

The STA/LTA algorithm detects signal onsets by comparing short-term and long-term averages of the waveform.

3.1.2 Mathematical Formulation

Short-Term Average (STA):

[STA(t) = *{i=0}^{N*{STA}-1} |x(t - i t)|]

where: - (N_{STA}) = number of samples in STA window (typically 0.5-2 seconds) - (t) = sample interval - (x(t)) = waveform sample at time (t)

Long-Term Average (LTA):

[LTA(t) = *{i=0}^{N*{LTA}-1} |x(t - N_{STA} t - i t)|]

where: - (N_{LTA}) = number of samples in LTA window (typically 10-60 seconds)

Characteristic Function:

[CF(t) =]

Detection Trigger:

[Detection(t) =

]

where: - (*{on}) = trigger threshold (typically 3-5)* - *(* {off}) = detrigger threshold (typically 2-3)

3.1.3 Implementation

```
        func STALTA(waveform []float64, sampleRate float64,
staWindow, ltaWindow time.Duration, thresholdOn,
thresholdOff float64) []Detection {
            nSTA := int(float64(staWindow.Seconds()) * sampleRa
            nLTA := int(float64(ltaWindow.Seconds()) * sampleRa

            detections := make([]Detection, 0)
            inDetection := false

            for i := nLTA + nSTA; i < len(waveform); i++ {
                // Calculate STA
                sta := 0.0
                for j := 0; j < nSTA; j++ {
                    sta += math.Abs(waveform[i-j])
                }
                sta /= float64(nSTA)

                // Calculate LTA
                lta := 0.0
                for j := 0; j < nLTA; j++ {
                    lta += math.Abs(waveform[i-nSTA-j])
                }
                lta /= float64(nLTA)

                // Calculate characteristic function
                cf := sta / lta
```

```
                // Detect onset
                if cf > thresholdOn && !inDetection {
                    inDetection = true
                    detections = append(detections, Detection{
                        Time:       time.Unix(0,
int64(i)*int64(time.Second)/int64(sampleRate)),
                        Type:       "ONSET",
                        STA:        sta,
                        LTA:        lta,
                        Ratio:      cf,
                        Confidence: math.Min(cf/thresholdOn, 1
                    })
                }

                // Detect end
                if cf < thresholdOff && inDetection {
                    inDetection = false
                }
            }

            return detections
        }
```

3.2 Phase Identification

3.2.1 P-Wave Identification

P-waves are identified by:

1. **First arrival** at each station
2. **Higher frequency content** (0.5-5 Hz)
3. **Shorter duration** than S-waves
4. **Particle motion** primarily vertical

P-wave polarity determination:

[Polarity = sign(w_P(t_{onset}) u_z)]

where: - (w_P(t_{onset})) = first P-wave sample - (u_z) = vertical unit vector

3.2.2 S-Wave Identification

S-waves are identified by:

1. **Arrival after P-wave** with time delay: [t = d (-)] where:
    - (d) = source-station distance
    - (v_P) = P-wave velocity (5.5-8.5 km/s)
    - (v_S) = S-wave velocity (3.0-4.5 km/s)
2. **Lower frequency content** (0.2-2 Hz)
3. **Larger amplitude** than P-wave (1.5-3×)
4. **Particle motion** perpendicular to propagation

3.2.3 Phase Picking Algorithm

```
        func IdentifyPhase(waveform []float64, sampleRate float
onsetTime time.Time) Phase {
            // Extract window around onset
            windowStart := onsetTime.Add(-0.5 * time.Second)
            windowEnd := onsetTime.Add(5.0 * time.Second)

            // Calculate frequency content
            spectrum := fft.FFT(waveform)

            // Find dominant frequency
            dominantFreq := findDominantFrequency(spectrum,
sampleRate)

            // Calculate polarization
            polarization := calculatePolarization(waveform)

            // Classify phase
            if dominantFreq > 0.5 && polarization.Vertical > 0.
                return PhaseP
            } else if dominantFreq < 2.0 && polarization.Horizo
> 0.6 {
                return PhaseS
            }

            return PhaseUnknown
        }
```

3.3 Signal-to-Noise Ratio (SNR)

3.3.1 Definition

[SNR(t) =]

3.3.2 SNR Calculation

```
        func CalculateSNR(waveform []float64, detectionTime
time.Time, sampleRate float64) float64 {
            // Pre-signal window (noise)
            noiseStart := detectionTime.Add(-10 * time.Second)
            noiseEnd := detectionTime.Add(-1 * time.Second)

            // Signal window
            signalStart := detectionTime
            signalEnd := detectionTime.Add(5 * time.Second)

            // Calculate RMS noise
            noiseRMS := calculateRMS(waveform, noiseStart, nois
sampleRate)

            // Calculate RMS signal
            signalRMS := calculateRMS(waveform, signalStart,
signalEnd, sampleRate)

            // SNR in dB
            return 20 * math.Log10(signalRMS / noiseRMS)
        }
```

4. Signal Detection Data Structure

4.1 Detection Object

```
        {
          "id": "SD-2024031400001",
          "stationId": "STA001",
          "channel": "BHZ",
          "phase": "P",
```

```
      "detectionTime": "2024-03-14T00:00:15.500Z",
      "detectionTimeUncertainty": 0.05,

      "amplitude": {
        "value": 5.2e-7,
        "units": "meters",
        "type": "PEAK",
        "confidence": 0.95
      },

      "signalToNoiseRatio": {
        "value": 12.5,
        "units": "dB",
        "staWindow": 1.0,
        "ltaWindow": 30.0
      },

      "period": {
        "value": 1.5,
        "units": "seconds",
        "measurement": "DOMINANT"
      },

      "classification": {
        "type": "EARTHQUAKE",
        "confidence": 0.87,
        "subtype": "TELESEISMIC"
      },

      "polarities": {
        "firstMotion": "COMPRESSIONAL",
        "uncertainty": 0.05
      },

      "featureMeasures": {
        "frequencyBand": {
          "low": 0.5,
          "high": 5.0,
          "units": "Hz"
        },
```

```
        "emergenceAngle": {
          "value": 45.0,
          "units": "degrees"
        },
        "slowness": {
          "value": 0.08,
          "units": "s/km"
        }
      },

      "processingInfo": {
        "algorithm": "STA/LTA",
        "version": "1.0",
        "parameters": {
          "staWindow": 1.0,
          "ltaWindow": 30.0,
          "triggerThreshold": 3.5,
          "detriggerThreshold": 2.0
        },
        "channelsUsed": ["BHZ", "BHN", "BHE"],
        "filterApplied": {
          "type": "BANDPASS",
          "lowFrequency": 0.5,
          "highFrequency": 5.0
        }
      },

      "associatedEventId": "EVENT-2024031400001",
      "locationContribution": {
        "residualTime": 0.15,
        "weight": 1.0
      }
    }
```

4.2 Waveform Data Structure

```
    {
      "id": "WF-2024031400001",
      "stationId": "STA001",
```

```
      "channel": "BHZ",

      "timeRange": {
        "startTime": "2024-03-14T00:00:00.000Z",
        "endTime": "2024-03-14T00:01:00.000Z",
        "duration": 60.0
      },

      "sampleRate": 40.0,
      "sampleCount": 2400,

      "samples": [0.123, -0.045, 0.089, ...],

      "units": "counts",

      "quality": {
        "completeness": 1.0,
        "minAmplitude": -0.5,
        "maxAmplitude": 0.8,
        "rmsAmplitude": 0.12,
        "meanAmplitude": 0.001,
        "gaps": [],
        "qualityFlags": ["GOOD"]
      },

      "processing": {
        "responseCorrected": true,
        "unitsConverted": true,
        "filters": [
          {
            "type": "BANDPASS",
            "lowFrequency": 0.5,
            "highFrequency": 5.0,
            "order": 4
          }
        ]
      }
    }
```

5. Event Association

5.1 Association Algorithm

Events are formed by associating detections across stations:

```
        func AssociateDetections(detections []Detection, maxTim
float64, minStations int) []Event {
            events := make([]Event, 0)

            // Sort by time
            sort.Slice(detections, func(i, j int) bool {
                return
detections[i].DetectionTime.Before(detections[j].DetectionTime)
            })

            // Group detections by time proximity
            groups := make([][]Detection, 0)
            currentGroup := []Detection{detections[0]}

            for i := 1; i < len(detections); i++ {
                timeDiff :=
detections[i].DetectionTime.Sub(detections[i-1].DetectionTime).S
                if timeDiff < maxTimeGap {
                    currentGroup = append(currentGroup,
detections[i])
                } else {
                    groups = append(groups, currentGroup)
                    currentGroup = []Detection{detections[i]}
                }
            }
            groups = append(groups, currentGroup)

            // Create events from groups
            for _, group := range groups {
                if len(group) >= minStations {
                    event := createEvent(group)
                    events = append(events, event)
                }
            }
```

```
        return events
    }
```

5.2 Event Object

```
{
  "id": "EVENT-2024031400001",
  "originTime": "2024-03-14T00:00:10.000Z",
  "originTimeUncertainty": 0.5,

  "location": {
    "latitude": 35.9275,
    "longitude": -106.4572,
    "depth": 10.5,
    "uncertainty": {
      "horizontal": 2.5,
      "vertical": 5.0,
      "units": "kilometers"
    }
  },

  "magnitude": {
    "value": 5.2,
    "type": "ML",
    "uncertainty": 0.3,
    "numStations": 15
  },

  "type": "EARTHQUAKE",
  "status": "AUTOMATIC",

  "associatedDetections": [
    "SD-2024031400001",
    "SD-2024031400002",
    "SD-2024031400003"
  ],

  "locationMethod": {
```

```
      "algorithm": "LEAST_SQUARES",
      "iterations": 5,
      "residualRMS": 0.15
    }
  }
```

5.3 Location Estimation

Geiger’s Method:

The event location is found by minimizing:

[^2 = _{i=1}^{N} ()^2]

where: - (t_i^{obs}) = observed arrival time at station (i) - (t_i^{calc}) = calculated travel time - (_i) = arrival time uncertainty - ((x, y, z)) = event location - (t_0) = origin time

Travel Time Calculation:

[t^{calc} = + t_{station_correction}]

where: - (d) = source-station distance - (v(x,y,z)) = velocity model - (t_{station_correction}) = station-specific correction

6. Processing Configuration

6.1 Filter Definitions

```
{
  "filterDefinitions": [
    {
      "name": "bandpass-1-5hz",
      "type": "BUTTERWORTH",
      "order": 4,
      "lowFrequencyHz": 1.0,
      "highFrequencyHz": 5.0,
      "passbandRipple": 0.1,
      "stopbandAttenuation": 40.0
    },
    {
      "name": "highpass-0.1hz",
      "type": "BUTTERWORTH",
      "order": 2,
```

```
            "cutoffFrequencyHz": 0.1,
            "passbandRipple": 0.1
          }
        ]
      }
```

6.2 Detection Parameters

```
      {
        "detectionParameters": {
          "staWindow": {
            "value": 1.0,
            "units": "seconds",
            "range": [0.5, 2.0]
          },
          "ltaWindow": {
            "value": 30.0,
            "units": "seconds",
            "range": [10.0, 60.0]
          },
          "triggerThreshold": {
            "value": 3.5,
            "range": [2.5, 5.0]
          },
          "detriggerThreshold": {
            "value": 2.0,
            "range": [1.5, 3.0]
          },
          "minimumSNR": {
            "value": 5.0,
            "units": "dB"
          }
        }
      }
```

6.3 Channel Configuration

```
{
  "channels": [
    {
      "name": "BHZ",
      "type": "BROADBAND",
      "orientation": "VERTICAL",
      "response": {
        "transferFunction": "POLES_ZEROS",
        "poles": [-0.03661+0.03661j, -7.549+0.0j],
        "zeros": [0.0j, 0.0j],
        "normalizationFactor": 1.0,
        "gain": 2946.6
      },
      "sampleRate": 40.0,
      "sensitivity": 1.589e9,
      "bandwidth": {
        "lowFrequency": 0.01,
        "highFrequency": 20.0
      }
    }
  ]
}
```

7. Implementation Specification

7.1 Go Implementation

```
package main

import (
    "math"
    "time"
)

// SignalDetection represents a detected signal
type SignalDetection struct {
    ID                  string    `json:"id"`
```

```
            StationID             string    `json:"stationId"`
            Channel               string    `json:"channel"`
            Phase                 string    `json:"phase"`
            DetectionTime         time.Time `json:"detectionTim
            DetectionTimeUncertainty float64
`json:"detectionTimeUncertainty"`
            Amplitude             float64   `json:"amplitude"`
            AmplitudeUnits        string    `json:"amplitudeUni
            SignalToNoiseRatio    float64
`json:"signalToNoiseRatio"`
            Period                float64   `json:"period"`
            Classification        string    `json:"classificati
            Confidence            float64   `json:"confidence"`
        }

        // WaveformGenerator generates realistic seismic wavefo
        type WaveformGenerator struct {
            sampleRate    float64
            noiseModel    NoiseModel
            stationParams StationParams
        }

        func (g *WaveformGenerator) GeneratePWave(amplitude,
duration float64) []float64 {
            nSamples := int(duration * g.sampleRate)
            samples := make([]float64, nSamples)

            // P-wave parameters
            fP := 2.0  // Hz
            alphaP := 1.0  // attenuation

            for i := 0; i < nSamples; i++ {
                t := float64(i) / g.sampleRate
                samples[i] = amplitude * math.Exp(-alphaP*t) *
                             math.Sin(2*math.Pi*fP*t)
            }

            return samples
        }
```

```
        func (g *WaveformGenerator) GenerateSWave(amplitude,
duration float64) []float64 {
            nSamples := int(duration * g.sampleRate)
            samples := make([]float64, nSamples)

            // S-wave parameters
            fS := 1.0  // Hz
            alphaS := 0.5  // attenuation

            for i := 0; i < nSamples; i++ {
                t := float64(i) / g.sampleRate
                samples[i] = amplitude * math.Exp(-alphaS*t) *
                             math.Sin(2*math.Pi*fS*t)
            }

            return samples
        }

        func (g *WaveformGenerator) GenerateNoise(duration floa
[]float64 {
            nSamples := int(duration * g.sampleRate)
            samples := make([]float64, nSamples)

            // Colored noise model
            for i := 0; i < nSamples; i++ {
                samples[i] = g.noiseModel.Generate()
            }

            return samples
        }

        func (g *WaveformGenerator) GenerateWaveform(event Even
station Station) []float64 {
            // Calculate travel times
            distance := haversine(station.Location, event.Locat
            tP := distance / 5.5  // P-wave velocity km/s
            tS := distance / 3.5  // S-wave velocity km/s

            // Generate components
            pWave := g.GeneratePWave(event.Magnitude * station.
```

```
10.0)
            sWave := g.GenerateSWave(event.Magnitude * station.
* 2.5, 15.0)
            noise := g.GenerateNoise(60.0)

            // Combine with timing
            samples := make([]float64, len(noise))
            for i := 0; i < len(samples); i++ {
                t := float64(i) / g.sampleRate

                // P-wave arrival
                pIndex := int(tP * g.sampleRate)
                if i >= pIndex && i < pIndex + len(pWave) {
                    samples[i] += pWave[i - pIndex]
                }

                // S-wave arrival
                sIndex := int(tS * g.sampleRate)
                if i >= sIndex && i < sIndex + len(sWave) {
                    samples[i] += sWave[i - sIndex]
                }

                // Add noise
                samples[i] += noise[i]
            }

            return samples
        }
```

7.2 Data Stream Specification

```
        // DataStream represents a continuous data stream
        type DataStream struct {
            StationID    string        `json:"stationId"`
            Channel      string        `json:"channel"`
            SampleRate   float64       `json:"sampleRate"`
            StartTime    time.Time     `json:"startTime"`
            EndTime      time.Time     `json:"endTime"`
            Samples      []float64     `json:"samples"`
```

```
    Quality      QualityInfo   `json:"quality"`
}

// QualityInfo contains data quality information
type QualityInfo struct {
    Completeness    float64   `json:"completeness"`
    MinAmplitude    float64   `json:"minAmplitude"`
    MaxAmplitude    float64   `json:"maxAmplitude"`
    RMSAmplitude    float64   `json:"rmsAmplitude"`
    MeanAmplitude   float64   `json:"meanAmplitude"`
    Gaps            []Gap     `json:"gaps"`
    QualityFlags    []string  `json:"qualityFlags"`
}
```

8. Testing Scenarios

8.1 Test Dataset 1: Regional Earthquake

```
{
  "event": {
    "magnitude": 5.2,
    "location": {"latitude": 35.0, "longitude": -106.0,
"depth": 10.0},
    "time": "2024-03-14T00:00:10Z"
  },
  "stations": ["STA001", "STA002", "STA003"],
  "expectedDetections": [
    {
      "station": "STA001",
      "channel": "BHZ",
      "phase": "P",
      "arrivalTime": "2024-03-14T00:00:15.5Z",
      "snr": 15.2
    }
  ]
}
```

8.2 Test Dataset 2: Teleseismic Event

```
        {
          "event": {
            "magnitude": 6.5,
            "distance": 5000,  // km
            "location": {"latitude": 40.0, "longitude": -120.0,
"depth": 50.0},
            "time": "2024-03-14T01:00:00Z"
          },
          "stations": ["STA001", "STA002"],
          "expectedDetections": [
            {
              "station": "STA001",
              "channel": "BHZ",
              "phase": "P",
              "arrivalTime": "2024-03-14T01:08:20Z",
              "snr": 8.5
            }
          ]
        }
```

8.3 Test Dataset 3: Explosion

```
        {
          "event": {
            "magnitude": 4.0,
            "type": "EXPLOSION",
            "location": {"latitude": 37.0, "longitude": -105.0,
"depth": 0.5},
            "time": "2024-03-14T02:00:00Z"
          },
          "stations": ["STA001", "STA002"],
          "expectedDetections": [
            {
              "station": "STA001",
              "channel": "BHZ",
              "phase": "P",
              "arrivalTime": "2024-03-14T02:01:30Z",
```

```
            "snr": 25.0,
            "classification": "EXPLOSION"
        }
    ]
}
```

9. Performance Requirements

9.1 Latency Requirements

Operation	Maximum Latency
Waveform generation (1 minute)	< 100 ms
Signal detection	< 500 ms
Event association	< 1 second
WebSocket broadcast	< 50 ms

9.2 Throughput Requirements

Metric	Minimum
Stations	100+
Channels per station	3
Sample rate	40 Hz
Detection rate	100/second

9.3 Data Volume

Metric	Value
Waveform data	14.4 KB/station/channel/second
Detection data	1 KB/detection
Event data	0.5 KB/event

10. Conclusion

This specification defines a realistic seismic simulation framework that:

1. Generates physically realistic waveforms using seismic wave propagation models
2. Implements standard STA/LTA signal detection
3. Produces signal detection objects matching SNL-GMS data structures
4. Associates detections into events with location estimates
5. Provides continuous data streams via WebSocket

The implementation enables comprehensive testing of GMS signal processing pipelines without requiring access to classified DoD data.

References

[1] Peterson, J. (1993). “Observation and Modeling of Seismic Background Noise.” USGS Open-File Report 93-322.

[2] Havskov, J., & Alguacil, G. (2006). “Tools in Seismology: SO-2.” Instituto Andaluz de Geofísica.

[3] Trnkoczy, A. (2002). “Understanding and Setting STA/LTA Trigger Algorithm Parameters for Broadband Seismometers.” IASPEI New Manual of Seismological Observatory Practice.

[4] Havskov, J., & Ottemöller, L. (2010). “Routine Data Processing in Seismology.” Springer.

[5] Bormann, P. (2002). “New Manual of Seismological Observatory Practice.” GeoForschungsZentrum Potsdam.
