Levene and Brown-Forsynthe: Test for variances

Application to Global Navigation Satellite Systems (GNSS)

Embry-Riddle Aeronautical University

MA412 – Probability and Statistics

Presented by: Jose Nicolas Gachancipa

Presented to: Professor Brenneman

April 30th, 2019

Introduction

Global Navigation Satellite Systems (GNSS) are satellite constellations that provide time and location services for multiple industries. GNSS signals are often collected/processed by antennas and receivers on the ground.

The communication between satellites and receivers is susceptible to interferences. In general, radio-waves tend to bounce with nearby objects before reaching the antenna, disturbing the signal and resulting in position errors. This phenomenon is known as multi-path. GNSS signals sent by satellites at low elevations (i.e. near the horizon line) are more susceptible to multi-path effects. For this reason, elevation thresholds are often used to remove distorted signals from the data. An optimal elevation threshold is needed to minimize multi-path effects while maximizing the amount of valid data collected by the receiver.

The purpose of this project is to develop a Python algorithm capable of determining the optimal elevation threshold of a GNSS receiver, by using a statistical test known as the Brown-Forsynthe test. The receiver is a GPStation-6 model produced by Novatel Inc and is currently located at the Space Physics Laboratory at Embry-Riddle Aeronautical University.

Background

Multipath interference occurs when a radio-signal reaches an antenna from multiple locations at the same time (Figure 1). In most cases, multipath is generated by neighboring objects that reflect the signal as it is trying to reach the antenna. Multipath also becomes prominent when the signal comes from a satellite that is at a low elevation (close to the horizon line) (Sanz, Zornoza, and Hernandez, 2011).

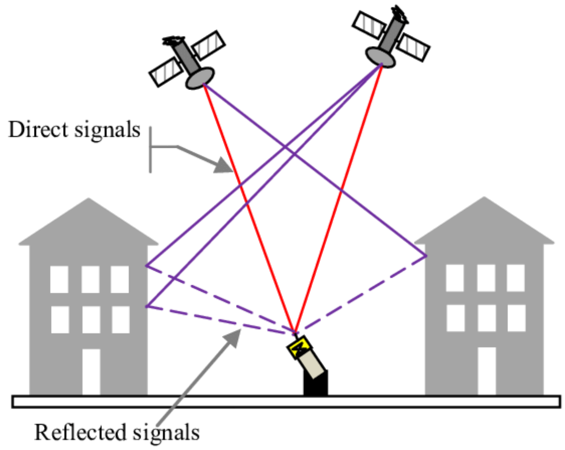

Figure 1. Multipath interference. Reprinted from *Argus Tracking* (2019). Retrieved from: https://argustracking.zendesk.com/

Throughout the past decades, scientists have established different methods to detect and isolate the effects of multipath on GNSS data. Most of these methods use post-processing/ filtering of the data. Additionally, some methods use observable measures, such as Code-minus-carrier (CMC) measurements, to isolate and mitigate multipath error (Blanco-Delgado, and Uijt de Haag, 2011).

The Code-Minus-Carrier (CMC) is defined as the difference between the code-phase and carrier-phase measurements of the signal (Blanco-Delgado, and Uijt de Haag, 2011). Code phase observables are measured by corelating a pseudo-random code generated by a satellite to the one generated by the receiver, and determining the time difference that it takes for the signal to travel between the two points (Blanco-Delgado, and Uijt de Haag, 2011). On the other hand, carrier phase measurements are a way to approximate the distance between the satellite and the receiver in terms of signal frequency cycles, to a more precise value ("Code-Phase GPS", n.d.). The CMC observable is computed by subtracting the carrier phase from the code phase measurements, and is a representative measure of the scintillation, multipath and receiver noise interferences affecting the signal (Ammana, 2018). Therefore, for lower elevations, higher CMC values are expected.

By computing the CMC measurements at different satellite elevations, it is possible to correlate the signal interferences with the position of the satellites in the sky, and determine the optimal elevation threshold of the receiver.

Numerical Solution

Levene's test is a method used to determine if two, or more, samples have equal variances. The equivalence of the variances is examined via a hypothesis test. There are multiple variations of Levene's test, which are used depending on the type of distribution of the data ("Levene Test", n.d.). For instance, the original version of Levene's test is accurate when testing data with normal distributions, while other variations, such as the *Brown-Forsythe* method, provide a more accurate approach to datasets with skewed distributions (SciPy, 2014).

Levene's test assumes that the variances of the groups are approximately equal to each other as shown below (where k is the number of groups being examined):

$$H_0: {\sigma_1}^2 = {\sigma_2}^2 = \ldots = {\sigma_k}^2$$

If any two groups (a_1 and a_2) have different variances, the null hypothesis is automatically rejected:

$$H_a: {\sigma_{a1}}^2 \neq {\sigma_{a2}}^2$$

The first step of the test is to compute a test statistic (W) described by the following equation:

$$\boldsymbol{W} = \frac{(N-k)}{(k-1)} * \frac{\sum_{a=1}^{k} N_a (T_a - T)^2}{\sum_{a=1}^{k} \sum_{b=1}^{N_a} (T_{ab} - T_a)^2} \quad (\mathbf{1})$$

Each of the terms in equation (1) is defined below:

$\boldsymbol{N}$**:** Count of all elements in all groups.
$\boldsymbol{k}$**:** Number of groups.
$\boldsymbol{a}$**:** Group number.
$\boldsymbol{b}$**:** Element of a group.
$\boldsymbol{N_a}$: Count of all elements inside group a.

$$\boldsymbol{T_{ab}}:$$
$$T_{ab} = |Y_{ab} - P|,$$
$$where\ Y_{ab}\ is\ element\ b\ from\ group\ a, and\ P$$
$$is\ the\ parameter\ (dependent\ on\ the\ test\ type):$$
$$P = \bar{Y}_a, \qquad Levene's\ test$$
$$P = \tilde{Y}_a, \qquad Brown-Forsynthe$$
$$\bar{Y}_a\ is\ the\ mean\ of\ group\ a$$
$$\tilde{Y}_a\ is\ the\ median\ of\ group\ a$$

$\boldsymbol{T}$: Mean of all T_{ab} elements in all groups.
$\boldsymbol{T_a}$: Mean of all T_{ab} elements in group a.

From equation (1) and the definition of the term $\boldsymbol{T_{ab}}$, it is worth mentioning that the only difference between Levene's (original) test and Brown-Forsynthe test is the parameter. Brown-Forsynthe uses the median, rather than the mean, when computing $\boldsymbol{T_{ab}}$.

Once the test statistic (W) is computed, it is compared to the upper critical value of the F distribution. The F distribution is often used in Analysis of Variance (Dinov, 2019). In this project, the F critical value (or F statistic) is computed using a python package (*scipy.stats.f.ppf*). The F upper critical value is dependent on the degrees of freedom of the W test statistic, and α (significance level). The degrees of freedom in Levene's test are $N-k$ and $k-1$ for the numerator and denominator respectively.

If the W test statistic results to be larger than the F test statistic, the variances of the groups are not equal, and therefore, the null hypothesis is rejected.

Results

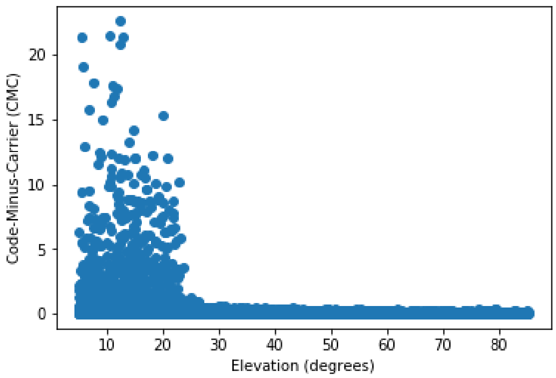

Figure 2. Code-Minus-Carrier data for a GNSS receiver ($N = 20{,}500$)

Figure 2 shows the relationship between the CMC coefficient and satellite elevation. The data was collected on February 11th, 2019 by a GNSS receiver located at Embry-Riddle Aeronautical University. The figure comprises 20,500 data points from multiple GPS satellites.

As seen in the figure, the data follows a skewed (not normal) distribution. For this reason, a Brown-Forsynthe test was performed (rather than a regular Levene test). Initially, the data was subdivided into multiple groups. A total of 90 windows ($k = 90$) were defined, each including data for a range of one degree of elevation.

In order to perform the test, the python algorithm (Appendix A) iterated, adding one window at a time and computing the test statistic W repeatedly. If at any iteration, W resulted to be bigger than the F critical value, the null hypothesis was rejected. The algorithm would then stop iterating, defining the elevation threshold at this point. The F statistic was computed at a level of significance of 0.001. The results are shown in Figure 3, where all red values are datapoints that lie under the elevation threshold set by the algorithm.

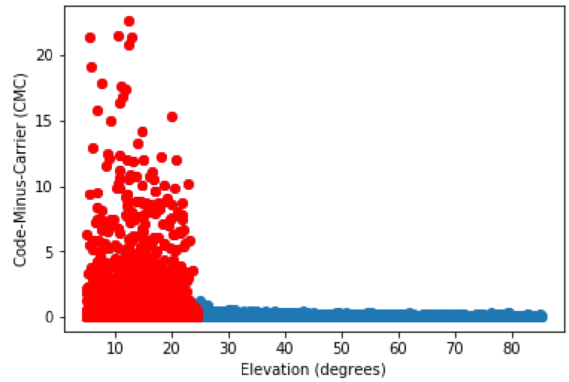

Figure 3. Code-Minus-Carrier data for a GNSS receiver ($N = 20{,}500$) after setting an elevation threshold.

In order to validate the accuracy and precision of the algorithm, the elevation threshold was computed using different number of windows (k). In general, the algorithm showed consistent results, defining the elevation threshold at ~24°:

```
k: 90 | Cut-off elevation threshold:  22.99 degrees.
k: 190 | Cut-off elevation threshold:  22.73 degrees.
k: 290 | Cut-off elevation threshold:  23.27 degrees.
k: 390 | Cut-off elevation threshold:  23.52 degrees.
k: 490 | Cut-off elevation threshold:  23.69 degrees.
k: 590 | Cut-off elevation threshold:  24.1 degrees.
k: 690 | Cut-off elevation threshold:  23.85 degrees.
k: 790 | Cut-off elevation threshold:  23.92 degrees.
k: 890 | Cut-off elevation threshold:  24.06 degrees.
k: 990 | Cut-off elevation threshold:  24.45 degrees.
```

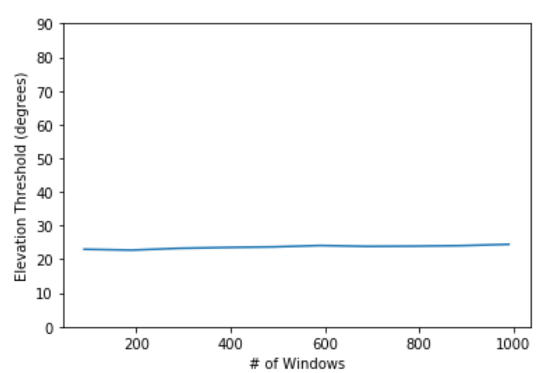

Figure 3. Elevation threshold as a function of number of data groups, computed using Brown-Forsynthe test.

Conclusions

The relationship between Code-Minus-Carrier (CMC) data and satellite elevation can be used to determine an optimal elevation threshold for a GNSS receiver. The algorithm developed in Python provides an accurate approximation of this threshold by using the Brown-Forsynthe test. This model may be used in the future for any receiver capable of collecting CMC values, and will allow the minimization of multipath and noise effects on GNSS data.

References

Ammana, S. (2018) *An Analysis of GPS Code Minus Carrier Measurements for GBAS Applications.* Retrieved from: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3220953

Blanco-Delgado, and Uijt de Haag (2011) *Multipath Analysis using Code-minus-Carrier for Dynamic Testing of GNSS Receivers.* Retrieved from: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5955254

Code-Phase GPS vs. Carrier-Phase GPS (n.d.) Retrieved from: https://www.trimble.com/gps_tutorial/sub_phases.aspx

Dinov, I. (March 25, 2019). *F Distribution Tables.* Retrieved from http://www.socr.ucla.edu/Applets.dir/F_Table.html

Levene Test for Equality of Variances. (n.d.). Retrieved from https://www.itl.nist.gov/div898/handbook/eda/section3/eda35a.htm

Sanz, Zornoza, and Hernandez (2011). *Multipath.* Retrieved from https://gssc.esa.int/navipedia/index.php/Multipath

Sanz, Zornoza, and Hernandez (2011). *GNSS basic observables.* Retrieved from: https://gssc.esa.int/navipedia/index.php/GNSS_Basic_Observables

SciPy. (May 11, 2014). *scipy.stats.levene.* Retrieved from https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.levene.html

```python
import csv
import numpy as np
import math
import matplotlib.pyplot as plt
import statistics
import scipy.stats
import scipy

#
# EMBRY-RIDDLE AERONAUTICAL UNIVERSITY
# MA412 - PROBABILITY AND STATISTICS
# FINAL PROJECT
# Jose Nicolas Gachancipa
#
print('\n#####################\n')
print('Embry-Riddle Aeronautical University')
print('Department of Mathematics')
print('MA412 - Probability and Statistics')
print('Code developed by: Jose Nicolas Gachancipa')
print('Purpose: Establish the elevation threshold for a GPS receiver using CMC.')
print('\n#####################\n')

# Inputs.
min_value = 0
max_value = 90
groups_range = range(90,1000,100)        # [Start, End, Increment]
significance_level = 0.001
reverse = 0
levene = 0
brown_forsynthe = 1

# Open the Excel file and extract the info.
x_axis_column = 0
y_axis_column = 1
header_cutoff_row = 0
with open('MA412.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    count = 1
    x_axis = []
    y_axis = []
    for row in csv_reader:
        if count>header_cutoff_row:
            x_axis.append(float(row[x_axis_column]))
            y_axis.append(float(row[y_axis_column]))
        count = count + 1

# Sort the values using the x_axis as a baseline.
original_y_axis = [t for _,t in sorted(zip(x_axis,y_axis))]
original_x_axis = sorted(x_axis)
plt.plot(x_axis,y_axis,'o')

#  Subdivide the array.
def divide_1(x,y,number_of_groups):
    divisions = np.linspace(min_value,max_value,number_of_groups+1)
    output_x = []
    output_y = []
    for i in range(len(divisions)-1):
```

```
        lower_limit = divisions[i]
        upper_limit = divisions[i+1]
        x_vector = []
        y_vector = []
        count = 0
        for value in x:
            if value>=lower_limit and value<upper_limit:
                x_vector.append(value)
                y_vector.append(y[count])
            count = count + 1
        if len(x_vector) > 0:
            output_x.append(x_vector)
            output_y.append(y_vector)
    return [output_x,output_y]

# Define functions.
# Function:
def divide_2(vector, number_of_divisions):
    a = np.linspace(0,len(vector),number_of_divisions+1)
    out = []
    for i in range(len(a)-1):
        out.append(vector[math.floor(a[i]):math.floor(a[i+1])])
    return out

# Run iteratevely for multiple divisions.
all_cut_off = []
for groups in groups_range:

    # Divide the data into the number of groups specified by the user.
    [input_x_axis, input_y_axis] = divide_1(original_x_axis,original_y_axis,groups)
    #input_x_axis = divide_2(original_x_axis, groups)
    #input_y_axis = divide_2(original_y_axis, groups)
    if reverse == 1:
        input_x_axis = input_x_axis[::-1]
        input_y_axis = input_y_axis[::-1]

    # Run the test progressively, until the variances are not equal.
    for i in range(2,len(input_x_axis)+1):

        # Set the arrays correspondingly first.
        divided_x_axis = input_x_axis[:i]
        divided_y_axis = input_y_axis[:i]
        x_axis = [j for i in divided_x_axis for j in i]
        y_axis = [j for i in divided_y_axis for j in i]

        # Compute the means and lengths of the lists as needed.
        mean = sum(y_axis)/len(y_axis)
        Nis = [len(x) for x in divided_x_axis]
        medians = [statistics.median(j) for j in divided_y_axis]
        means = [sum(j)/len(j) for j in divided_y_axis]
        N = len(x_axis)   # Total number of cases in all groups.
        k = len(divided_x_axis)        # Number of groups.

        # Compute the degrees of freedom.
        v1 = N-k
        v2 = k-1
```

```
# Find (Zi.) and (Zi..).
Zi_dots = []
zi_doubledot = 0
count = 0
for group in divided_y_axis:
    if levene == 1:
        local_parameter = means[count] # Use the mean for Levene's test.
    elif brown_forsynthe == 1:
        local_parameter = medians[count] # Use the median for brown-forsynthe.
    element_differences = []
    for element in group:
        element_differences.append(abs(element - local_parameter))
    zi_doubledot = zi_doubledot + sum(element_differences)
    average_differences = sum(element_differences)/len(element_differences)
    Zi_dots.append(average_differences)
    count = count + 1
zi_doubledot = zi_doubledot/len(x_axis)

# Compute the numerator of the function using (Zi.) and (Zi..).
count = 0
numerator = 0
for zi_dot in Zi_dots:
    local_length = Nis[count]
    numerator = numerator + (local_length*((zi_dot-zi_doubledot)**2))
    count = count + 1
numerator = numerator*(v1)

# Compute the denominator.
count = 0
denominator = 0
for group in divided_y_axis:
    if levene == 1:
        local_parameter = means[count] # Use the mean for Levene's test.
    elif brown_forsynthe == 1:
        local_parameter = medians[count] # Use the median for brown-forsynthe.
    element_differences = 0
    local_zdot = Zi_dots[count]
    for element in group:
        element_differences = element_differences + ((abs(element - \
                                        local_parameter)-local_zdot)**2)
    denominator = denominator + element_differences
    count = count + 1
denominator = denominator*(v2)

# Determine the f critical value.
F = scipy.stats.f.ppf(q=1-significance_level, dfn=v1, dfd=v2)

# Compute W.
W = numerator/denominator

# Run until we reject the null hypothesis.
# The null hypothesis establishes that the variances of all selected groups are equal.
# The null hypothesis is rejected when W is bigger than F.
# W = Test value
# F = Critical falue of the F distrbution
if W>F:
    cutoff_threshold =  divided_x_axis[-1][-1]
```

```
            print('k:',groups,'| Cut-off elevation threshold: ',cutoff_threshold,'degrees.')
            all_cut_off.append(cutoff_threshold)
            break

# Plot.
r = [j for i in divided_x_axis[:-1] for j in i]
q = [j for i in divided_y_axis[:-1] for j in i]
plt.plot(r,q,'ro')
plt.xlabel('Elevation (degrees)')
plt.ylabel('Code-Minus-Carrier (CMC)')
plt.show()
plt.plot(groups_range,all_cut_off)
plt.ylim((0,90))
plt.xlabel('# of Windows')
plt.ylabel('Elevation Threshold (degrees)')
plt.show()
```