**Outlier detection using Hampel**

[[](https://medium.com/@miguel.otero.pedrido.1993?source=post_page-----17db1d265375--------------------------------)](https://medium.com/@miguel.otero.pedrido.1993?source=post_page-----17db1d265375--------------------------------)

[Miguel Otero Pedrido](https://medium.com/@miguel.otero.pedrido.1993?source=post_page-----17db1d265375--------------------------------)

·

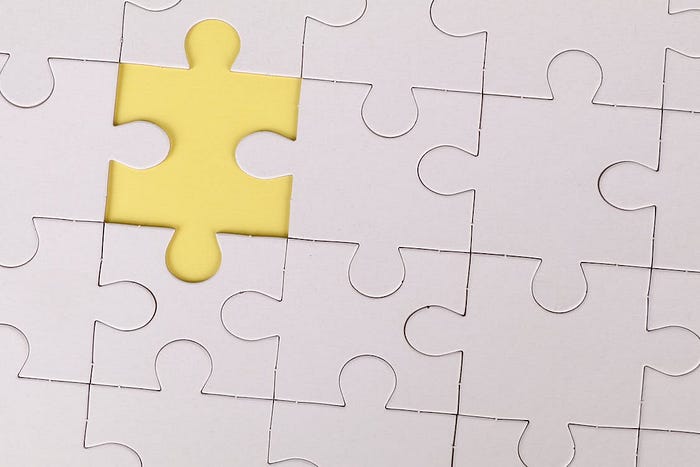
Follow

5 min read

·

Sep 20

42



In the realm of time series data analysis, the identification and handling of anomalies are crucial tasks. ***Anomalies, or outliers, are data points that deviate significantly from the expected patterns, potentially indicating errors, fraud, or valuable insights****.*

**One effective technique for addressing this challenge is the Hampel Filter.**

In this article, we will explore how to apply this outlier detection technique , using my [hampel library](https://github.com/MichaelisTrofficus/hampel_filter).

**Let’s begin!**



1. **The Hampel Filter Demystified**

The Hampel Filter is a **robust method for detecting and handling outliers in time series data.**It relies on the [**Median Absolute Deviation (MAD)**](https://en.wikipedia.org/wiki/Median_absolute_deviation)and employs a rolling window for the identification of outliers. **MAD is a robust measure of data dispersion, calculated as the median of the absolute deviations from the median value.**

Configuring the Hampel filter involves two parameters:

* **Window Size**: This parameter determines the size of the moving window used to evaluate each data point. It essentially defines the scope within which we look for outliers.
* **Threshold**: Careful selection of the threshold is essential to avoid triggering outlier detection for valuable data.

**2. Hampel meets Python**🐍

To use the Hampel filter in your Python project, first install the package via pip:

pip install hampel

And import it in your Python script using:

from hampel import hampel

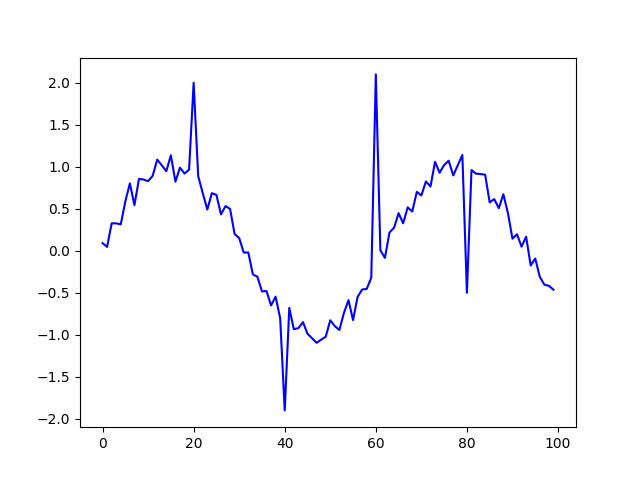
The hampel function has three available parameters:

* data: The input 1-dimensional data to be filtered (pandas.Series or numpy.ndarray).
* window\_size (optional): The size of the moving window for outlier detection (**default is 5**).
* n\_sigma (optional): The number of standard deviations for outlier detection (**default is 3.0**). It is related to the threshold concept discussed in the previous section, i.e. **by tuning this parameter we can have more or less tolerance to possible outliers.**

Now let’s generate synthetic data, in which we will introduce four outliers at positions 20, 40, 60, 80 (**of course in real situations the problem will not be so easy, but it is a good example to understand how hampel works**😅).

import matplotlib.pyplot as plt  
import numpy as np  
from hampel import hampel  
  
original\_data = np.sin(np.linspace(0, 10, 100)) + np.random.normal(0, 0.1, 100)  
  
# Add outliers to the original data  
for index, value in zip([20, 40, 60, 80], [2.0, -1.9, 2.1, -0.5]):  
 original\_data[index] = value

Plotting original\_data you should see something like this:



It is very easy to detect the four outliers we have introduced visually, **but let’s see if Hampel is also capable**🤞.

result = hampel(original\_data, window\_size=10)

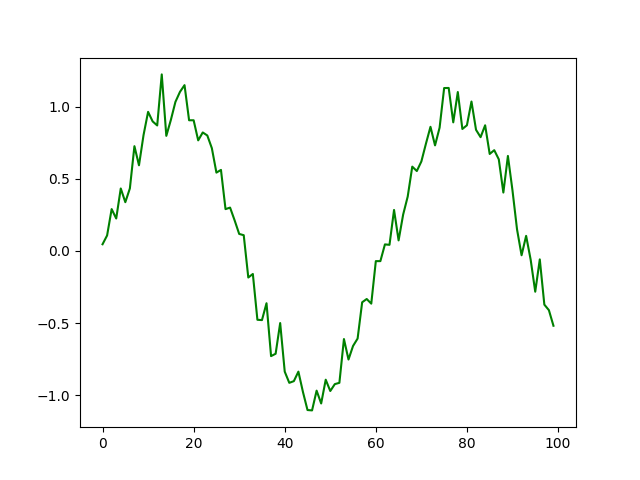
The hampel function returns a Result dataclass, which contains the following attributes:

* filtered\_data: The data with outliers replaced.
* outlier\_indices: Indices of the detected outliers.
* medians: Median values within the sliding window.
* median\_absolute\_deviations: Median Absolute Deviation (MAD) values within the sliding window.
* thresholds: Threshold values for outlier detection.

We can access these attributes as simply as this:

filtered\_data = result.filtered\_data  
outlier\_indices = result.outlier\_indices  
medians = result.medians  
mad\_values = result.median\_absolute\_deviations  
thresholds = result.thresholds

If we now print, for example, the filtered\_data , we’ll have a cleaned version of the original\_data , that is, without the outliers.



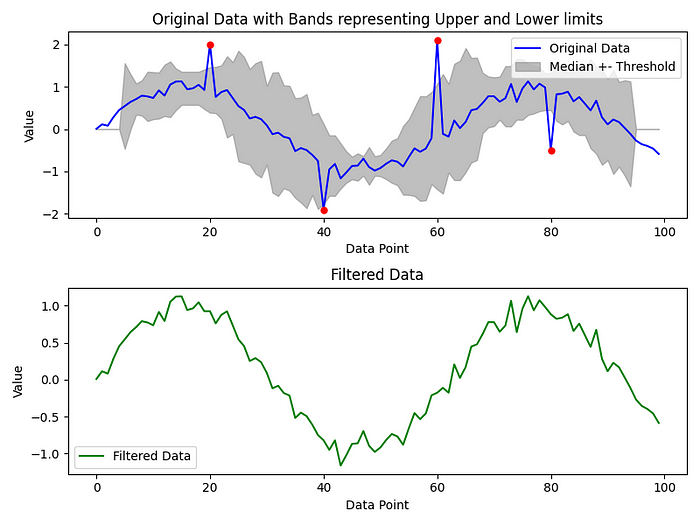
**That’s really cool! Hampel managed to remove the outliers we added previously! 💪**

However, we can take advantage of the information provided by hampel to design **a much more interesting graph**. In my case, I’ll draw the **outliers as red dots** and will also**add a grey band representing the threshold used by the algorithm at each point**. In addition, I’ll **create another plot below the first one showing the filtered data.**

This is very easy to do using matplotlib:

fig, axes = plt.subplots(2, 1, figsize=(8, 6))  
  
  
# Plot the original data with estimated standard deviations in the first subplot  
axes[0].plot(original\_data, label='Original Data', color='b')  
axes[0].fill\_between(range(len(original\_data)), medians + thresholds,  
 medians - thresholds, color='gray', alpha=0.5, label='Median +- Threshold')  
axes[0].set\_xlabel('Data Point')  
axes[0].set\_ylabel('Value')  
axes[0].set\_title('Original Data with Bands representing Upper and Lower limits')  
  
for i in outlier\_indices:  
 axes[0].plot(i, original\_data[i], 'ro', markersize=5) # Mark as red  
  
axes[0].legend()  
  
# Plot the filtered data in the second subplot  
axes[1].plot(filtered\_data, label='Filtered Data', color='g')  
axes[1].set\_xlabel('Data Point')  
axes[1].set\_ylabel('Value')  
axes[1].set\_title('Filtered Data')  
axes[1].legend()  
  
# Adjust spacing between subplots  
plt.tight\_layout()  
  
# Show the plots  
plt.show()

After running the snippet, you should see this beautiful figure 😍.



And just in case you want to copy-paste the full Python script …👇👇 👇

import matplotlib.pyplot as plt  
import numpy as np  
from hampel import hampel  
  
original\_data = np.sin(np.linspace(0, 10, 100)) + np.random.normal(0, 0.1, 100)  
  
# Add outliers to the original data  
for index, value in zip([20, 40, 60, 80], [2.0, -1.9, 2.1, -0.5]):  
 original\_data[index] = value  
  
result = hampel(original\_data, window\_size=10)  
  
filtered\_data = result.filtered\_data  
outlier\_indices = result.outlier\_indices  
medians = result.medians  
thresholds = result.thresholds  
  
fig, axes = plt.subplots(2, 1, figsize=(8, 6))  
  
# Plot the original data with estimated standard deviations in the first subplot  
axes[0].plot(original\_data, label='Original Data', color='b')  
axes[0].fill\_between(range(len(original\_data)), medians + thresholds,  
 medians - thresholds, color='gray', alpha=0.5, label='Median +- Threshold')  
axes[0].set\_xlabel('Data Point')  
axes[0].set\_ylabel('Value')  
axes[0].set\_title('Original Data with Bands representing Upper and Lower limits')  
  
for i in outlier\_indices:  
 axes[0].plot(i, original\_data[i], 'ro', markersize=5) # Mark as red  
  
axes[0].legend()  
  
# Plot the filtered data in the second subplot  
axes[1].plot(filtered\_data, label='Filtered Data', color='g')  
axes[1].set\_xlabel('Data Point')  
axes[1].set\_ylabel('Value')  
axes[1].set\_title('Filtered Data')  
axes[1].legend()  
  
# Adjust spacing between subplots  
plt.tight\_layout()  
  
# Show the plots  
plt.show()

I hope this tutorial has been helpful in explaining how to apply hampel to clean our time series. If you are interested in seeing the details of the algorithm implementation (**in my case it’s implemented using Cython**), you are more than welcome to take a look [at the repository](https://github.com/MichaelisTrofficus/hampel_filter/tree/master)😛.

**See you next time!**👋👋👋

**If this tutorial has been of interest to you, follow me on twitter and Linkedin, where I also publish similar content 😀😀😀**

Linkedin → <https://www.linkedin.com/in/moteropedrido/>

Twitter → <https://twitter.com/MTrofficus>