TDA on US Stock Indices

Philip N Kaddaj

Higher School of Economics

December 15, 2023

Overview

Topological Data Analysis joins parts of statistics, computing and topology in order to find 'shape'-like structures in data.

We apply the techniques of Katz and Gidea to perform TDA on the returns of the 4 major US stock indices. Our analysis helps us clearly visualize the market crash resulting to the COVID 19 pandemic as spikes on certain TDA metrics.

Background

The general set up for TDA is as follows. We start with a point cloud $P \subset \mathbb{R}^n$.

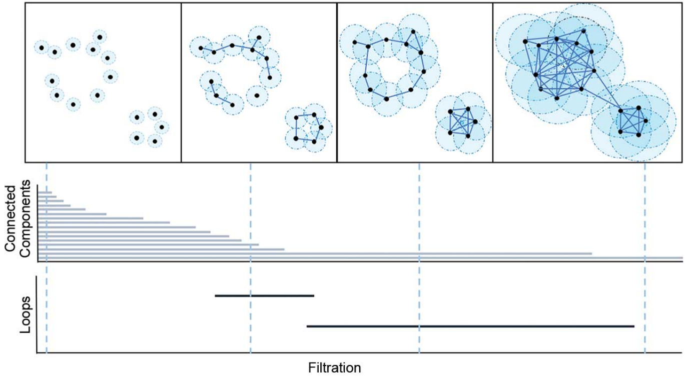
From this point cloud we create a filtered simplicial complex:

$$K_0 \subset K_1 \subset K_2 \subset \ldots \subset K_n$$

There are multiple methods to do this, one convenient for computation is the Vietoris Rips complex which is determined by its 1-skeleton.

One then takes homology of filtered complexes and keeps track of the persistent cycles in each dimension. This can be summarised as a barcode or persistence diagram.

From Point Cloud to Simplicial Complex to Barcode



Persistence Diagrams and Persistene Landscapes

A persistence diagram is a multiset of points in $\mathbb{R}^2$ where points are of the form (b, d) coresponding to the birth and death time of a particular cycle.

The space of persistence diagrams is a metric space (endowed with the Wasserstein metric) however lacks any more structure thus it is hard to perform statistical analysis.

One solution is to embed the set of persistence diagrams into a Banach space. In other words from a persistence diagram we obtain a PL function we call a persistence landscape.

Persistence Landscape

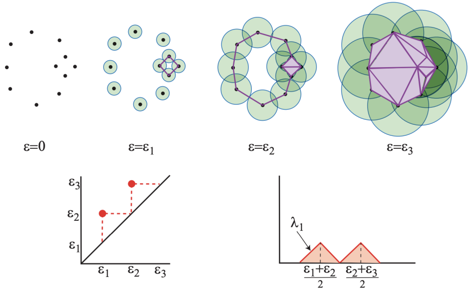


Figure 1: Rips filtration of simplicial complexes illustrating the birth and death of loops; the 1-dimensional persistence diagram and the corresponding persistence landscape are shown below.

Katz and Gidea

In a 2017 paper Katz and Gidea analyse the log returns of the 4 major US Stock Indices using tools from persistence homology.

First they create a point cloud $P \subset \mathbb{R}^4$. A point $p \in P$ has coordinates $(r_1^t, r_2^t, r_3^t, r_4^t)$. Where r_i^t is the log return of the $i^t h$ index at time t (measured in days).

They then consider a sliding window of size $w = 50$. That is each sliding window is a subcloud $P_t \subset P$. For each P_t first the Vietoris Rips complex is computed, then the persistence diagram which is converted to a persistence landscape (for statistical analysis).

$$P_t \to H^*(P_t) \to f_{P_t}$$

Katz and Gidea

Taking the L^2 and L^1 norms of the persistence landscapes and plotting them as a graph over time they noticed that the norms rose sharply around periods of market crashes.

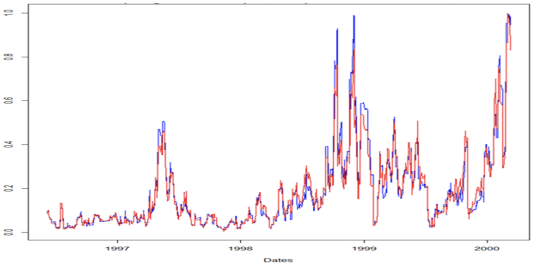
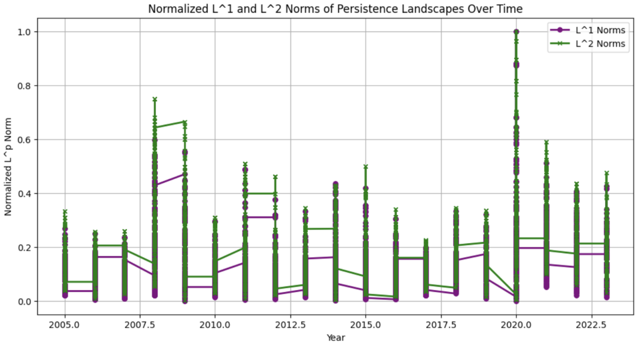


Figure 9: The time series of normalized L^1 (blue line) and L^2 (red line) norms of persistence landscapes calculated with the sliding window of 50 days. Color online.

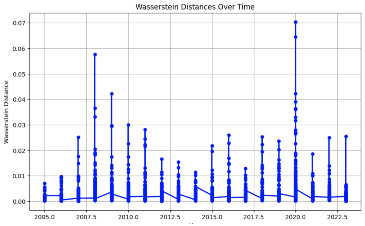
Results

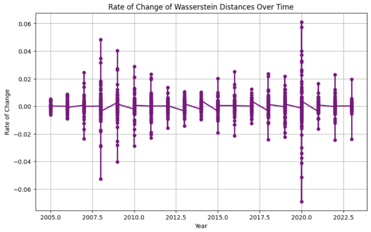
We follow their methods, downloading data on the DJIA, Russel 2000, NASDAQ and SP500 from Yahoo Finance and reproduce them over a larger time frame. The result is a higher spike around the time of the COVID19 pandemic (the first spike is the 2008 crash).



Results

We also independently compute the Wasserstein distances between persistence diagrams as well as its rate of change. The crash spikes are also apparent using this analysis.





The Code

All index data is imported from yahoo finance.

To perform some TDA calculations we use a library called gudhi which written in C++ with a Python interface.

The main tools we use from there are:

1. Rips Complex
2. Simplex Tree
3. Wasserstein Distance

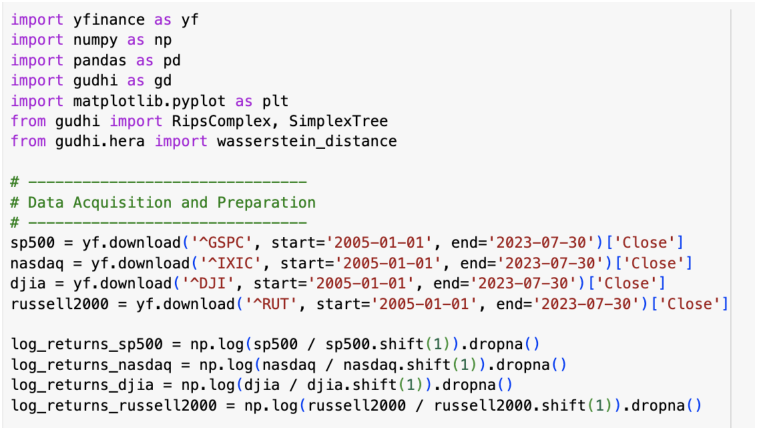
We code independently the persistence landscape with the help of a lambda function and plot all diagrams using matplotlib.

Numpy is used for convenince of calculations with arrays.

Data Acquisition and Preparation

- Downloading Stock Data: The code uses yfinance to download closing prices of four major indices: S&P 500, NASDAQ, Dow Jones Industrial Average (DJIA), and Russell 2000. The data is collected from January 1, 2005, to July 30, 2023.
- Computing Log Returns: Log returns are calculated for each index. The log return for a given day is the natural logarithm of the closing price divided by the previous day's closing price. This normalizes the data and stabilizes variance.

Code: Data Acquisition and Preparation

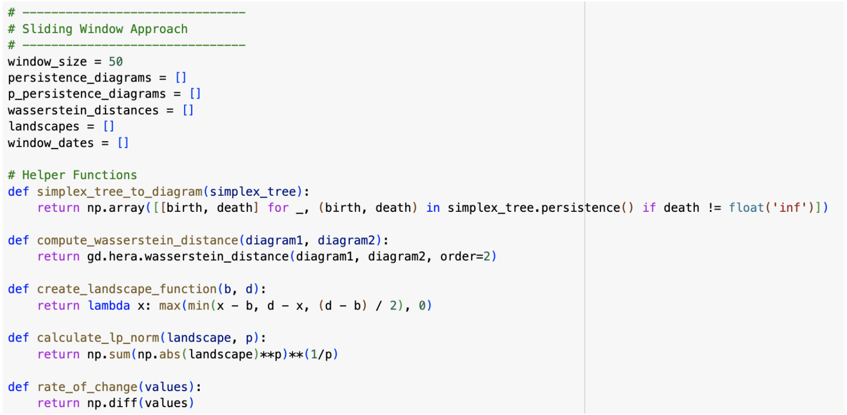


```python
import yfinance as yf
import numpy as np
import pandas as pd
import gudhi as gd
import matplotlib.pyplot as plt
from gudhi import RipsComplex, SimplexTree
from gudhi.hera import wasserstein_distance

# ------------------------------
# Data Acquisition and Preparation
# ------------------------------
sp500 = yf.download('^GSPC', start='2005-01-01', end='2023-07-30')['Close']
nasdaq = yf.download('^IXIC', start='2005-01-01', end='2023-07-30')['Close']
djia = yf.download('^DJI', start='2005-01-01', end='2023-07-30')['Close']
russell2000 = yf.download('^RUT', start='2005-01-01', end='2023-07-30')['Close']

log_returns_sp500 = np.log(sp500 / sp500.shift(1)).dropna()
log_returns_nasdaq = np.log(nasdaq / nasdaq.shift(1)).dropna()
log_returns_djia = np.log(djia / djia.shift(1)).dropna()
log_returns_russell2000 = np.log(russell2000 / russell2000.shift(1)).dropna()
```

Sliding Window Approach

- Initializing Variables: Several lists are initialized to store data for each sliding window: persistence_diagrams, p_persistence_diagrams, wasserstein_distances, landscapes, and window_dates.
- Helper Functions:
- simplex_tree_to_diagram: Converts a GUDHI SimplexTree object into a persistence diagram.
- compute_wasserstein_distance: Computes the Wasserstein distance between two persistence diagrams.
- create_landscape_function: Creates a landscape function for a given birth and death pair in a persistence diagram.
- calculate_lp_norm: Calculates the L^p norm of a landscape.
- rate_of_change: Computes the rate of change of a given list of values.

Code: Sliding Window Approach

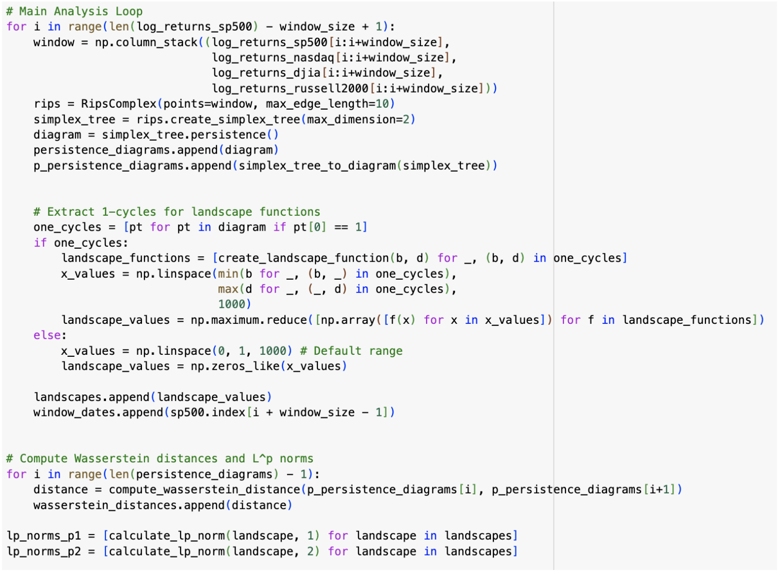


```python
# --------------------------------
# Sliding Window Approach
# --------------------------------
window_size = 50
persistence_diagrams = []
p_persistence_diagrams = []
wasserstein_distances = []
landscapes = []
window_dates = []

# Helper Functions
def simplex_tree_to_diagram(simplex_tree):
    return np.array([[birth, death] for _, (birth, death) in simplex_tree.persistence() if death != float('inf')])

def compute_wasserstein_distance(diagram1, diagram2):
    return gd.hera.wasserstein_distance(diagram1, diagram2, order=2)

def create_landscape_function(b, d):
    return lambda x: max(min(x - b, d - x, (d - b) / 2), 0)

def calculate_lp_norm(landscape, p):
    return np.sum(np.abs(landscape)**p)**(1/p)

def rate_of_change(values):
    return np.diff(values)
```

Main Analysis Loop

- The code iterates over each sliding window of size 50 days. For each window:
- A point cloud is created using log returns of the indices.
- TDA is performed using RipsComplex to create a persistence diagram.
- Persistence diagrams and their SimplexTree representations are stored.
- Landscape values are computed for 1-cycles in the persistence diagrams.
- Wasserstein Distances and L^p Norms:
- Wasserstein distances are calculated between consecutive persistence diagrams. L^1 and L^2 norms are calculated for the landscapes.

Code: Main Analysis Loop

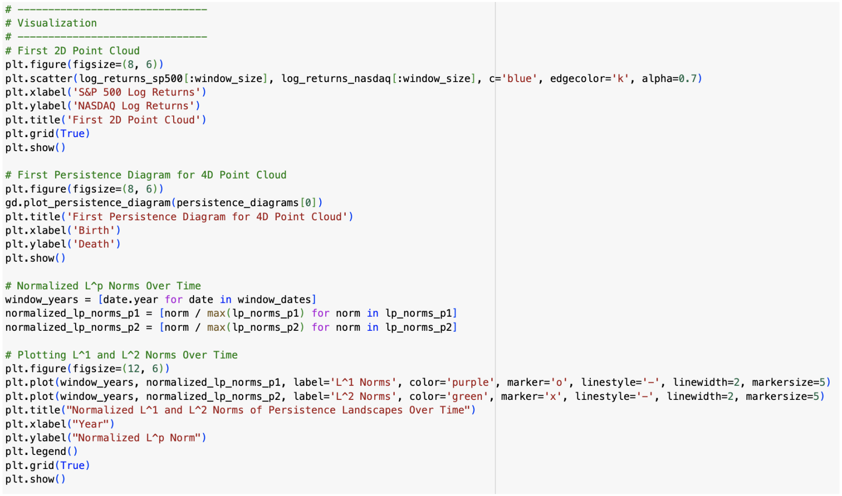


```python
# Main Analysis Loop
for i in range(len(log_returns_sp500) - window_size + 1):
    window = np.column_stack((log_returns_sp500[i:i+window_size],
                              log_returns_nasdaq[i:i+window_size],
                              log_returns_djia[i:i+window_size],
                              log_returns_russell2000[i:i+window_size]))
    rips = RipsComplex(points=window, max_edge_length=10)
    simplex_tree = rips.create_simplex_tree(max_dimension=2)
    diagram = simplex_tree.persistence()
    persistence_diagrams.append(diagram)
    p_persistence_diagrams.append(simplex_tree_to_diagram(simplex_tree))


    # Extract 1-cycles for landscape functions
    one_cycles = [pt for pt in diagram if pt[0] == 1]
    if one_cycles:
        landscape_functions = [create_landscape_function(b, d) for _, (b, d) in one_cycles]
        x_values = np.linspace(min(b for _, (b, _) in one_cycles),
                               max(d for _, (_, d) in one_cycles),
                               1000)
        landscape_values = np.maximum.reduce([np.array([f(x) for x in x_values]) for f in landscape_functions])
    else:
        x_values = np.linspace(0, 1, 1000) # Default range
        landscape_values = np.zeros_like(x_values)

    landscapes.append(landscape_values)
    window_dates.append(sp500.index[i + window_size - 1])


# Compute Wasserstein distances and L^p norms
for i in range(len(persistence_diagrams) - 1):
    distance = compute_wasserstein_distance(p_persistence_diagrams[i], p_persistence_diagrams[i+1])
    wasserstein_distances.append(distance)

lp_norms_p1 = [calculate_lp_norm(landscape, 1) for landscape in landscapes]
lp_norms_p2 = [calculate_lp_norm(landscape, 2) for landscape in landscapes]
```

Visualization

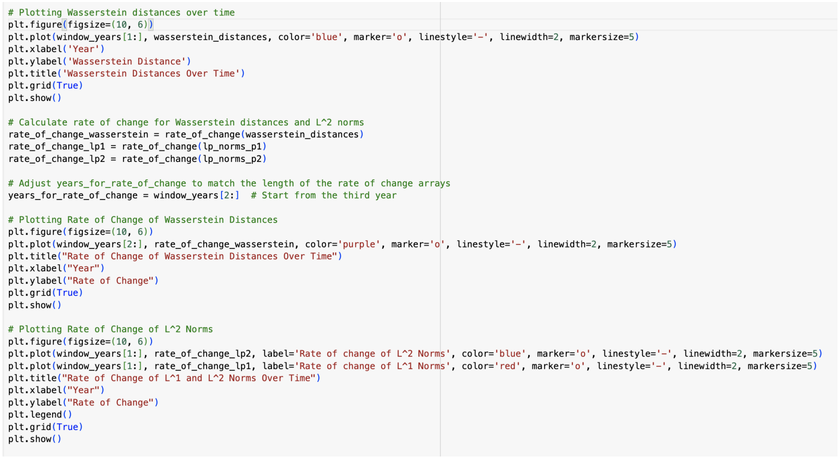
- First 2D Point Cloud: The first 2D point cloud is plotted using log returns of the S/P 500 and NASDAQ.
- First Persistence Diagram: The first persistence diagram for the 4D point cloud is plotted.
- Normalized L^p Norms Over Time: L^1 and L^2 norms of persistence landscapes over time are plotted. The norms are normalized to their respective maximum values for better comparison.
- Wasserstein Distances Over Time: A plot showing how the Wasserstein distances vary over the selected years.
- Rate of Change of Wasserstein Distances and L^p Norms:
- Rate of change of Wasserstein distances and L^p norms are calculated. Separate plots are made to show the rate of change over time for both Wasserstein distances and L^p norms.

Code: Visualization



```python
# -------------------------------
# Visualization
# -------------------------------
# First 2D Point Cloud
plt.figure(figsize=(8, 6))
plt.scatter(log_returns_sp500[:window_size], log_returns_nasdaq[:window_size], c='blue', edgecolor='k', alpha=0.7)
plt.xlabel('S&P 500 Log Returns')
plt.ylabel('NASDAQ Log Returns')
plt.title('First 2D Point Cloud')
plt.grid(True)
plt.show()

# First Persistence Diagram for 4D Point Cloud
plt.figure(figsize=(8, 6))
gd.plot_persistence_diagram(persistence_diagrams[0])
plt.title('First Persistence Diagram for 4D Point Cloud')
plt.xlabel('Birth')
plt.ylabel('Death')
plt.show()

# Normalized L^p Norms Over Time
window_years = [date.year for date in window_dates]
normalized_lp_norms_p1 = [norm / max(lp_norms_p1) for norm in lp_norms_p1]
normalized_lp_norms_p2 = [norm / max(lp_norms_p2) for norm in lp_norms_p2]

# Plotting L^1 and L^2 Norms Over Time
plt.figure(figsize=(12, 6))
plt.plot(window_years, normalized_lp_norms_p1, label='L^1 Norms', color='purple', marker='o', linestyle='-', linewidth=2, markersize=5)
plt.plot(window_years, normalized_lp_norms_p2, label='L^2 Norms', color='green', marker='x', linestyle='-', linewidth=2, markersize=5)
plt.title("Normalized L^1 and L^2 Norms of Persistence Landscapes Over Time")
plt.xlabel("Year")
plt.ylabel("Normalized L^p Norm")
plt.legend()
plt.grid(True)
plt.show()
```

Code: Visualization



```python
# Plotting Wasserstein distances over time
plt.figure(figsize=(10, 6))
plt.plot(window_years[1:], wasserstein_distances, color='blue', marker='o', linestyle='-', linewidth=2, markersize=5)
plt.xlabel('Year')
plt.ylabel('Wasserstein Distance')
plt.title('Wasserstein Distances Over Time')
plt.grid(True)
plt.show()

# Calculate rate of change for Wasserstein distances and L^2 norms
rate_of_change_wasserstein = rate_of_change(wasserstein_distances)
rate_of_change_lp1 = rate_of_change(lp_norms_p1)
rate_of_change_lp2 = rate_of_change(lp_norms_p2)

# Adjust years_for_rate_of_change to match the length of the rate of change arrays
years_for_rate_of_change = window_years[2:]  # Start from the third year

# Plotting Rate of Change of Wasserstein Distances
plt.figure(figsize=(10, 6))
plt.plot(window_years[2:], rate_of_change_wasserstein, color='purple', marker='o', linestyle='-', linewidth=2, markersize=5)
plt.title("Rate of Change of Wasserstein Distances Over Time")
plt.xlabel("Year")
plt.ylabel("Rate of Change")
plt.grid(True)
plt.show()

# Plotting Rate of Change of L^2 Norms
plt.figure(figsize=(10, 6))
plt.plot(window_years[1:], rate_of_change_lp2, label='Rate of change of L^2 Norms', color='blue', marker='o', linestyle='-', linewidth=2, markersize=5)
plt.plot(window_years[1:], rate_of_change_lp1, label='Rate of change of L^1 Norms', color='red', marker='o', linestyle='-', linewidth=2, markersize=5)
plt.title("Rate of Change of L^1 and L^2 Norms Over Time")
plt.xlabel("Year")
plt.ylabel("Rate of Change")
plt.legend()
plt.grid(True)
plt.show()
```