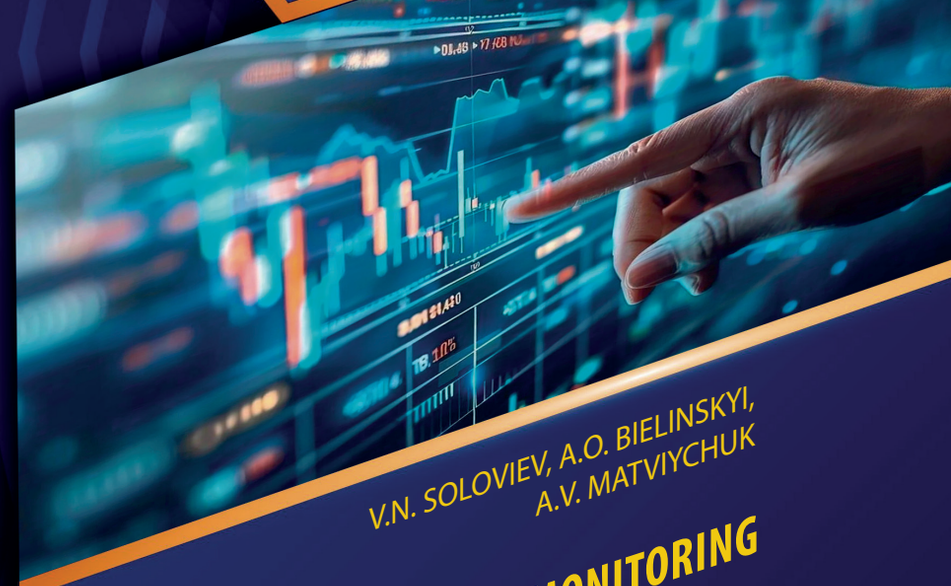
V.N. SOLOVIEV, A.O. BIELINSKYI,
A.V. MATVIYCHUK

CRISIS PHENOMENA MONITORING AND PREVENTION IN COMPLEX SOCIO-ECONOMIC SYSTEMS

Ministry of Education and Science of Ukraine
Kryvyi Rih State Pedagogical University
Kyiv National Economic University
named after Vadym Hetman



V.N. SOLOVIEV, A.O. BIELINSKYI,
A.V. MATVIYCHUK

CRISIS PHENOMENA MONITORING AND PREVENTION IN COMPLEX SOCIO-ECONOMIC SYSTEMS

Monograph

Cherkasy – 2024

Ministry of Education and Science of Ukraine

Kryvyi Rih State Pedagogical University

Kyiv National Economic University named after Vadym Hetman

V.N. SOLOVIEV, A.O. BIELINSKYI,

A.V. MATVIYCHUK

Crisis Phenomena Monitoring and Prevention in Complex Socio-Economic Systems

Monograph

Cherkasy – 2024

УДК 330.4(075.8)
ББК 65в631я-1
С 60

Recommended for publication by the decision of the Academic Council of Kryvyi Rih State Pedagogical University (Protocol No. 5 of November 09, 2023).

Reviewers:

O.H. Osaulenko, Academician of NAS of Ukraine in «Statistics», DSc in Public Administration, Professor, Rector of National Academy of Statistics, Accounting and Audit
M.I. Skripnichenko, Corresponding Member of NAS of Ukraine in «Econometrics", DSc in Economics, Professor, Leading Researcher of Department of Modeling and Forecasting of Economic Development of State Institution «Institute of Economics and Forecasting of NAS of Ukraine»
A.I. Shevchenko, Corresponding Member of NAS of Ukraine in «Computing Systems», DSc in Engineering, Professor, Director of Institute of Artificial Intelligence Problems

Soloviev V.N., Bielinskyi A.O., Matviychuk A.V.

С 60

Crisis Phenomena Monitoring and Prevention in Complex Socio-Economic Systems. Monograph. – Cherkasy: Publisher Tretiakov O.M., 2024. – 345.

ISBN 978-617-7827-89-3

This work is a part of the applied research "Monitoring, Forecasting, and Prevention of Crisis Phenomena in Complex Socio-Economic Systems", which is funded by the Ministry of Education and Science of Ukraine (project No. 0122U001694).

This monograph presents research findings on the dynamic and structural characteristics of financial and economic systems, grounded in the principles of complex systems theory. For this purpose, the study employs methodologies such as recurrence analysis, entropy-based techniques, network science, etc. Significant attention is devoted to modeling critical phenomena within financial and economic systems. The investigation delves into the peculiarities of the collective dynamics of complex systems during periods of crisis and recovery. Special emphasis is placed on the identification and construction of indicators of pre-crisis states through advanced processing of time series data.

This monograph will be valuable to a broad audience interested in the further development and practical application of interdisciplinary fields such as synergetics and econophysics, including specialists in economic and mathematical modeling, as well as graduate and undergraduate students.

The authors would also like to thank the Armed Forces of Ukraine for providing security to perform this work. This study has become possible only because of the resilience and courage of the Ukrainian Army.

УДК 330.4(075.8)
ББК 65в631я-1

ISBN 978-617-7827-89-3

Context

Introduction

Despite the expected predictions that the 21st century will be the century of biology (by analogy with the 20th century of physics), it can definitely be considered the century of complexity, thanks to the genius British astrophysicist Stephen William Hawking (1942-2018). Indeed, it turned out that complex systems of different nature exhibit similar patterns of complexity and can be characterized by universal interdisciplinary quantitative methods and algorithms. This is evidenced by the 2021 Nobel Prize in Physics "for pioneering contributions to our understanding of complex physical systems" was awarded to Italian physicist Giorgio Parisi 'for discovering the relationship between disorder and fluctuations in physical systems from the atomic to the planetary scale' and to Japanese-American climatologist Shukuro Manabe and German scientist Klaus Hasselmann "for physical modeling of the Earth's climate, quantifying variability, and reliably predicting global warming".

Although the definition of complex systems has historically been widely discussed, there is hardly any broad agreement on a single specific definition: a complex system is formed by many interacting elements that give rise to emergent phenomena [67]. However, some terms remain undefined. For example, how many elements are enough to reflect complex emergent phenomena? Interestingly, this number can be as large as the number of neurons and synapses in the human brain, but it can also be relatively small: a minimal cell is also formed by only a few hundred genes and is already alive. From these examples, it is clear that true complexity and new phenomena require a number of elements that can be very far from the large numbers typically considered in physics (the number of molecules in a mole of gas is defined by the Avogadro number, i.e., approximately 6×10^{23}. It seems to be a very commonly accepted idea that complexity arises from the competition between randomness and order, and that the topology of a complex system is inherently related to a certain amount of randomness in the network of

interactions between its elements (the so-called network approach) or by the sign of their interaction (the spin glass approach).

A point of departure often cited in the context of collective effects of complex systems is the article “More is different” written by Philip Anderson (1923-2020) [129], winner of the 1977 Nobel Prize in Physics. Complex systems science looks at the ways in which the constituent parts give rise to the collective behavior of the whole system. However, this interpretation has proven to be of limited utility because it covers too broad a set of circumstances.

Historically, some time ago, another more useful step in defining complex systems appeared in physics: a system is complex if its behavior depends significantly on its details [176]. In this context, we mean such phenomena as deterministic chaos, quantum entanglement, protein folding, spin glasses, etc. Collective complex behavior can occur under the influence of frustration and structural disorder. As a result, it is difficult to reach a state of equilibrium, reactions to external perturbations are slow and very often random. Such different phenomena are studied in different areas of physics: dynamical systems, quantum mechanics, and statistical physics. What they have in common is that infinitesimal changes in initial conditions (albeit of a very different nature) lead to radically different scenarios of the time evolution of these systems.

There is a second component that is important for defining complex systems. On the one hand, the interactions between the constituent parts lead to collective behavior and determine the macro state, but on the other hand, the interactions change during the evolution of the system and are influenced by the macro state. In other words, the macrostate and microstates dynamically update each other. The analysis of such effects has led to the creation of methods and the development of concepts that have been successfully applied to describe formally similar phenomena occurring in chemical, biological, social, and other systems formed by agents of non-physical origin.

Despite the impressive progress of complexity science over the past 50 years, we are still far from fully understanding complexity because we have not yet identified the necessary conditions for a system to reflect complex emergent phenomena. For example, we are far from fully understanding how the brain works. As a result, the field may appear fragmented compared to other more traditional scientific fields, such as physics. To solve the problem of complexity, we really need to investigate different aspects of complex systems, and we need to adopt an open viewpoint that is able to describe and predict data from complex systems, avoiding top-down predefined dogmas.

Complexity science is also key to understanding and predicting the evolution of major pandemics and to informing policy makers and the general public about the risks of epidemic spread. Indeed, the networked scientific community was already aware of the dangers of global pandemics that take advantage of scale-free global transportation systems long before COVID. Unfortunately, the pandemic took most countries by surprise, as contingency plans were not really prepared for an epidemic of the scale of COVID-19. To monitor the evolution of this pandemic and any future pandemic, scientists will likely combine large amounts of social mobility data with predictive models, which are key to monitoring the pandemic and informing policy makers, despite many uncertainties about the biological evolution of viruses.

In the future, progress at the intersection between complexity and biology will be key to achieving much-needed advances in precision medicine. This large, complex problem will require a truly interdisciplinary approach that combines network science, machine learning, and artificial intelligence with molecular biology and neuroscience. Indeed, while biology in recent decades has widely adopted a single-molecule approach or relied heavily on the central dogma of molecular biology, it is now well recognized that most diseases are complex, and to understand these diseases it is important to embrace the complexity and heterogeneity of the cell’s interacting networks. Finally, in the near future,

complexity will be key to laying the foundation for the quantum Internet, which will require combining advances in quantum information with our understanding of classical complex communication systems such as the current Internet.

Another important challenge regarding network robustness is brain research, as the brain is undoubtedly a robust complex system, but it is very important to understand how its function is affected by diseases. To answer this question, I believe we need to accept the stochastic nature of brain activity and gain further insight into the interplay between brain functions and brain network topology. Thanks to fundamental advances in network science, we already know that the resilience of networks to random damage is highly dependent on the statistical properties of the network. Indeed, the scale-free degree distribution of networks dramatically alters the phase diagram of percolation, exhibiting critical behavior that is dramatically different from percolation on regular lattices or on random graphs. These results were key to understanding the interaction between the underlying network structure of complex systems and their dynamics.

Predicting complex systems is a challenging task and, of course, is limited by the pervasive nonlinearity of their dynamics. However, important progress has been made in forecasting complex systems over the past twenty years (e.g., unprecedented progress in predicting the spread of an epidemic). Improvements in the power of predicting complex systems are largely due to the abundance of data available to modelers and the important advances that can be made by complexity science combined with data science and artificial intelligence (AI). This is evidenced by revolutionary developments in materials science [11] or finance [177].

Improving our ability to predict complex systems, which will eventually combine network science, data science, and artificial intelligence algorithms, is indeed key for a variety of applications, including providing possible climate change scenarios. However, the power of simple models to understand complex systems is crucial for interpreting results: simple models may not capture all the

details of complex systems, but they allow us to understand and tame complexity, which can be crucial in developing better AI algorithms. Spin glass theory teaches us that a model that is actually quite simple (just adding a random mixture of positive and negative interactions to an Ising model in a fully connected network) can already be very complex.

Methods of modeling complex systems are the subject of our previous monographs and textbooks focused on the Matlab computer mathematics system [1, 55, 169, 170]. Taking into account the dominance of Python in applied research of complex systems, the appearance of this monograph is justified in the authors' opinion. This monograph will be based on a Ukrainian-language guide to modeling complex systems in the Python programming language [168].

The selection of specific stock indices for constructing indicators-precursors of crash phenomena in stock markets is critical due to their role as benchmarks representing the overall health and dynamics of financial markets in different regions. The chosen indices – **S&P 500** for the United States, **Hang Seng Index** for China (Hong Kong), **DAX (Deutscher Aktienindex)** for Europe, and **BSE SENSEX** for India – are particularly suitable for this purpose, each offering unique advantages and relevance. Each of these indices is a leading benchmark in its respective region, representing the core economic and financial activities:

- The **S&P 500** reflects the performance of 500 major companies across diverse sectors in the United States, making it a globally recognized indicator of economic health.
- The **Hang Seng Index (HSI)** represents the largest and most influential companies traded in Hong Kong, serving as a bridge between the Chinese economy and global markets.
- The **DAX (Deutscher Aktienindex)** tracks the performance of 40 major German companies, acting as a barometer for the largest economy in the Eurozone.

- The **BSE SENSEX** captures the performance of 30 large and established companies in India, one of the world's fastest-growing economies.

Fig. 1 illustrates the historical trends of the aforementioned major global stock market indices.

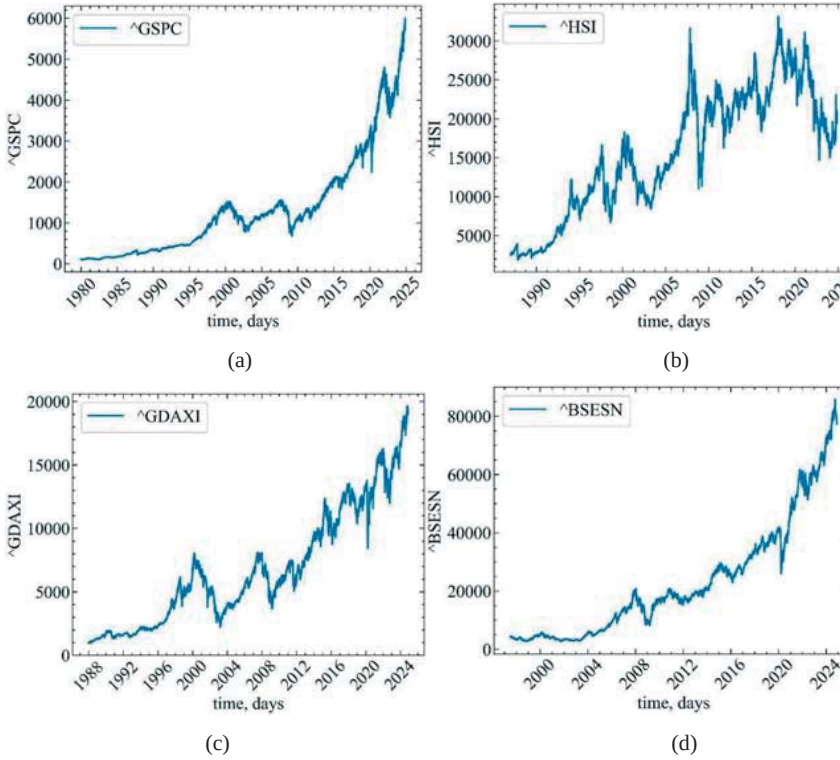


Fig. 1: Historical trends of major global stock market indices: S&P 500 (^GSPC) (a), Hang Seng Index (^HSI) (b), DAX (^GDAXI) (c), and BSE SENSEX (^BSESN) (d)

These indices are supported by extensive historical and real-time data, which are essential for constructing and validating indicators of crash phenomena. The availability of robust datasets enables detailed analysis of patterns, anomalies, and early warning signs that often precede market crashes.

Each index reflects distinct economic, political, and regulatory environments: (1) the **S&P 500** is influenced by U.S. monetary policy, global trade

dynamics, and technological innovation, making it a key driver of global markets; (2) the **Hang Seng Index** represents the intersection of China's economic influence and Hong Kong's role as a global financial hub, influenced by both domestic and international factors; (3) the **DAX** is shaped by Germany's export-driven economy, European Union policies, and its role as a leader in industrial innovation; (4) the **BSE SENSEX** reflects India's rapid economic growth, industrialization, and market reforms, while being sensitive to global commodity prices and domestic policy changes.

These indices are highly responsive to economic shocks, systemic risks, and speculative bubbles. Their historical performance includes well-documented instances of market downturns, making them ideal for analyzing crash precursors such as: rising market volatility; abnormal trading volumes; divergences between market prices and fundamental indicators; changes in cross-market correlations and capital flows.

By focusing on these indices, researchers can uncover early warning signals that indicate heightened risk of market instability across different regions.

Therefore, the entire monograph will be devoted to the analysis of complexity indicators derived from the above indices.

It should be noted that the selection of stock indices from developed countries with disparate stock market models is a consequence of the devastation inflicted on the national economy by Russia's military aggression against Ukraine. In light of the above, it is evident that the stock market is not a reliable indicator of the state of the national economy. However, it is possible to anticipate the recovery of economic development by constructing and subsequently adapting effective stock market indicators, utilising sophisticated economic signals such as the stock indices of the United States, Germany, China and India.

1 Complexity. Quantitative measures of complexity. Information methods of complexity assessment

This century is called the century of complexity. Today, the question “what is complexity?” is studied by physicists, biologists, mathematicians, and computer scientists, although with current advances in understanding the world around us, there is no unambiguous answer to this question.

For this reason, in accordance with the idea of I. Prigogine, we will study the manifestations of system complexity, using modern methods of quantitative complexity analysis [112].

Among these methods, the following deserve attention:

- information and entropy;
- based on chaos theory;
- multifractal.

Of course, based on the different nature of the methods underlying the formation of the complexity measure, they place certain requirements on the time series that serve as input data. For example, the first two groups of methods require stationarity of the input data. At the same time, they have different sensitivities to such characteristics as determinism, stochasticity, causality, and correlation. Therefore, in the future, when comparing the effectiveness of various complexity indicators, we will pay attention to these circumstances, emphasizing the specific applicability of a particular indicator for characterizing different aspects of the complexity of the systems under study.

We will begin our consideration of the first group of methods with the well-known measure of complexity proposed by A. Kolmogorov [12].

1.1 Kolmogorov complexity

The notion of Kolmogorov complexity (or, as it is also called, algorithmic entropy) appeared in the 1960s at the intersection of algorithm theory, information theory, and probability theory.

Kolmogorov's idea was to measure the amount of information contained in individual finite objects (and not in random variables, as in Shannon information theory). It turned out that this was possible (although only up to a limited limit). Kolmogorov proposed to measure the amount of information in finite objects using the theory of algorithms, defining the complexity of an object as the minimum length of the program that generates that object. This definition became the basis of algorithmic information theory and algorithmic probability theory: an object is considered random if its complexity is close to the maximum.

So what is Kolmogorov complexity and how can it be measured? In practice, we often come across programs that compress files (to save space in the archive). The most common are zip, gzip, compress, rar, arj, and others. By applying such a program to a file (with text, data, or a program), we get its compressed version (which is usually shorter than the original file). This version can be used to restore the original file using a paired program – a "decompressor". So, to a first approximation, the Kolmogorov complexity of a file can be described as the length of its compressed version. Thus, a file that has a regular structure and is well compressed has a small Kolmogorov complexity (compared to its length). On the contrary, a poorly compressible file has a complexity close to its length.

Suppose we have a fixed way of describing (decompressor) D. For a given word x, let us consider all its descriptions, i.e., all words y for which $D(y)$ is defined and equal to x. The length of the shortest of them, $l(y)$, is called the Kolmogorov complexity of word x for a given description method D:

$$KS_D(x) = \min\{l(y) \mid D(y) = x\},$$

where $l(y)$ denotes the length of the word y. The subscript D emphasizes that the definition depends on the choice of the way D is represented.

It can be shown that there are optimal ways to describe. A way of describing is better the shorter it is. Therefore, it is natural to give the following definition: method D_1 is not worse than method D_2 if $KS_{D_1}(x) \leq KS_{D_2}(x) + c$ for some c and all x.

Therefore, according to Kolmogorov, the complexity of an object (for example, a text – a sequence of characters) is the length of the minimal program that outputs this text, and the entropy is the complexity divided by the length of the text. You can also think of algorithmic complexity as the minimum time (or other computational resources) required to perform this task on a computer. And we can also talk about the communication complexity of tasks that involve more than one processor: this is the number of bits that need to be transmitted when solving this task [50, 100]. Unfortunately, this definition is purely speculative. There is no reliable way to unambiguously define this program. However, there are algorithms that actually try to calculate the Kolmogorov complexity of a text [110] and entropy [35].

1.2 Lempel-Ziv complexity

A universal (in the sense of applicability to different language systems) measure of the complexity of a finite symbolic sequence was proposed by Lempel and Ziv (LZ) [10]. The **Lempel-Ziv complexity** (LZC) is a classical measure that links the concepts of complexity (in the Kolmogorov-Chaitin sense) and entropy rate for ergodic sources [91, 153]. For an ergodic dynamic process, the amount of new information received per unit time (entropy rate) can be estimated by measuring the ability of this source to generate new patterns. Due to the simplicity of the LZC method, the entropy rate can be estimated from a single discrete measurement sequence with low computational cost [58]. In their approach, the complexity of the sequence is estimated by the number of steps of the process that generates it. Acceptable (editorial) operations in this case are:

1. Character generation (required at least for the synthesis of alphabet elements).

2. Copying a “ready-made” fragment from the background (i.e., from an already synthesized part of the text).

Let Σ be a finite alphabet, S be a text (a sequence of characters) composed of elements of Σ; $S[i]$ be the i-th character of the text; $S[i:j]$ be a fragment of the text from the i-th to the j-th character inclusive ($i < j$); $N = |S|$ be the length of the text S. Then the sequence synthesis scheme can be represented as a concatenation

$$H(S) = S[1:i_1]S[i_1+1:i_2]\dots S[i_{k-1}+1:i_k]\dots S[i_{m-1}+1:N], \qquad (1.1)$$

where $S[i_{k-1}+1:i_k]$ is the fragment of S generated at the k-th step, and $m = m_H(S)$ is the number of steps of the process. Of all the possible schemes for generating S, the one with the minimum number of steps is chosen. Thus, the complexity of the sequence S in terms of LZ

$$c_{LZ}(S) = \min_H\{m_H(S)\}.$$

The minimum number of steps is ensured by choosing the longest prototype from the prehistory to copy at each step. If we denote by $j(k)$ the number of the position from which copying begins at the k-th step, then the length of the copy fragment

$$l_{j(k)} = i_k - i_{k-1} - 1 = \max_{j\le i_{k-1}}\{l_j : S[i_{k-1}+1:i_{k-1}+l_j]\} = S[j:j+l_j-1], \qquad (1.2)$$

and the k-th component of the complex decomposition (1.1) can be written in the form

$$S[i_{k-1}+1:i_k] = \begin{cases} S[j(k):j(k)+l_{j(k)}-1], & \text{if } j(k)\neq 0, \\ S[i_{k-1}+1], & \text{if } j(k)=0. \end{cases} \qquad (1.3)$$

The case $j(k) = 0$ corresponds to the situation when the position $i_{k-1}+1$ contains a character that has not been encountered before. In this case, we use the symbol generation operation.

We will find the LZ complexity for a time series that represents, for example, daily values of a financial index. To study the dynamics of LZ and

compare it with other complex systems, we will find this complexity measure for a fixed-length subset (window). To do this, we will calculate the logarithmic returns and convert them into a sequence of bits. In doing so, you can specify the number of states to be differentiated (the number system). For example, for two different states, we have 0, 1, for three states, 0, 1, 2, etc. For a binary encoding system, the threshold will be set by the average value and the states, for example, of returns (ret) will be encoded as follows [41, 137, 138]:

$$ret = \begin{cases} 0, & ret_t < \langle ret \rangle, \\ 1, & ret_t > \langle ret \rangle. \end{cases} \tag{1.4}$$

It is also possible to define the so-called permutation LZC (PLZC) [103, 175]. In this case, we will rely on the phase space reconstruction procedure that will be mention in the next section. According to the permutation procedure, we will take a fragment of the series of length m, which serves as the dimension of the reconstructed attractor, and replace each value of the series with its ordinal index. Fig. 1.1 shows the time series and its possible ordinal patterns:

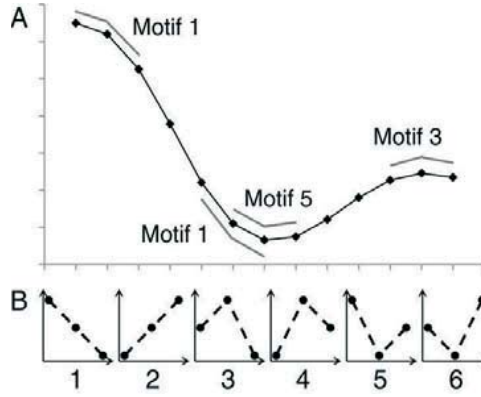


Fig. 1.1: A fragment of the time series (a) and 6 possible ordinal patterns that can occur in this signal (b) [29]

The LZ algorithm performs two operations: (1) adds a new bit to an existing sequence; (2) copies an already formed sequence. The algorithmic complexity is the number of such operations required to generate a given sequence.

For a random sequence of length n, the algorithmic complexity is calculated by the expression $LZC_r = n/\log(n)$. Then the relative algorithmic complexity is the ratio of the resulting complexity to the complexity of the random sequence: $LZC = LZC/LZC_r$.

Let us consider the possibility of using the LZC index as an indicator of catastrophic events.

For further work on modeling complex systems, we will use the `yfinance` library as a basis, which allows working with financial market data using the Python programming language.

> ⓘ Note
>
> **Yahoo!, Y!Finance, and Yahoo! finance are registered trademarks of Yahoo, Inc.**
>
> `yfinance` is not affiliated with, endorsed by, or verified by Yahoo, Inc. It is an open source tool that uses publicly available Yahoo APIs and is intended for research and educational purposes.
>
> You should refer to Yahoo!'s terms of use for detailed information about your rights to use the actual data you download. Remember – the Yahoo! Financial API is for your personal use only.

To install the `yfinance` library, you can use the following command:

```
!pip install yfinance --upgrade --no-cache-dir
```

The GitHub repository (https://github.com/ranaroussi/yfinance) contains more information about the library itself, errors that may occur, and potential solutions.

First, we import the necessary modules for further work:

```
import matplotlib.pyplot as plt
import numpy as np
import neurokit2 as nk
import yfinance as yf
import pandas as pd
import scienceplots
from tqdm import tqdm
```

```
%matplotlib inline
```

And then we'll customize the figures for the output:

```
plt.style.use(['science', 'notebook', 'grid'])


size = 22
params = {
'figure.figsize': (8, 6),            # set the default width and height of th
e figures
'font.size': size,                   # the size of fonts
'lines.linewidth': 2,                # line width
'axes.titlesize': 'small',           # size of titles above figures
'axes.labelsize': size,              # size of labels on the axes
'legend.fontsize': size,             # font size of legend
'xtick.labelsize': size,             # the size of the labeling on the Ox axi
s
'ytick.labelsize': size,             # the size of the labeling on the Oy axi
s
"font.family": "Serif",              # font family
"font.serif": ["Times New Roman"],   # font style
'savefig.dpi': 300,                  # dots per inch
'axes.grid': False                   # creating a grid on the figure itself
}

plt.rcParams.update(params)          # update the style according to the sett
ings
```

As it was already mentioned in the "Introduction" section, the chosen indices – **S&P 500** for the USA, **Hang Seng Index** for China (Hong Kong), **DAX (Deutscher Aktienindex)** for Europe, and **BSE SENSEX** for India – are particularly suitable for the purpose of the indicators-precursors construction.

Using the functionality of the `yfinance` library, let's try to download the historical values of these indices for the period from January 1, 1980 to November 20, 2024. Obviously, not all indices will have the values of the specified starting period. Some of them will start to exist a little later. Nevertheless, the `yfinance` library will take this into account and automatically pull up the values for the available period:

```
symbol = '^GSPC'                          # index symbol
start = "1980-01-01"                      # data reading start date
end = "2024-11-20"                        # end date of data reading

data = yf.download(symbol, start, end)  # download data
time_ser = data['Adj Close'].copy()     # saving only adjusted closing prices
```

```
xlabel ='time, days'                    # caption on the x-axis
ylabel = symbol                         # caption along the y-axis
```

To bring the series to a standardized initial series or standardized returns, we define the `transformation()` function:

```
def transformation(signal, ret_type):

    for_rec = signal.copy()

    if ret_type == 1:       # given the type of series, we perform
# necessary transformations
        pass
    elif ret_type == 2:
        for_rec = for_rec.diff()
    elif ret_type == 3:
        for_rec = for_rec.pct_change()
    elif ret_type == 4:
        for_rec = for_rec.pct_change()
        for_rec -= for_rec.mean()
        for_rec /= for_rec.std()
    elif ret_type == 5:
        for_rec = for_rec.pct_change()
        for_rec -= for_rec.mean()
        for_rec /= for_rec.std()
        for_rec = for_rec.abs()
    elif ret_type == 6:
        for_rec -= for_rec.mean()
        for_rec /= for_rec.std()

    for_rec = for_rec.dropna().values

    return for_rec
```

To plot a pair of time series, we define the `plot_pair()` function:

```
def plot_pair(x_values,
              y1_values,
              y2_values,
              y1_label,
              y2_label,
              x_label,
              file_name, clr="magenta"):

    fig, ax = plt.subplots()

    ax2 = ax.twinx()
    ax2.spines.right.set_position(("axes", 1.03))

    p1, = ax.plot(x_values,
                  y1_values,
"b-", label=fr"{y1_label}")
    p2, = ax2.plot(x_values,
                   y2_values,
                   color=clr,
```

```
                    label=y2_label)

    ax.set_xlabel(x_label)
    ax.set_ylabel(f"{y1_label}")
    ax.yaxis.label.set_color(p1.get_color())
    ax2.yaxis.label.set_color(p2.get_color())

    tkw = dict(size=2, width=1.5)

    ax.tick_params(axis='x', rotation=35, **tkw)
    ax.tick_params(axis='y', colors=p1.get_color(), **tkw)
    ax2.tick_params(axis='y', colors=p2.get_color(), **tkw)
    ax2.legend(handles=[p1, p2])

    plt.savefig(file_name +".jpg")

    plt.show();
```

Now, let's set the parameters for LZC procedure and perform necessary calculations:

```
ret_type = 1                            # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

window = 500                            # sliding window length
tstep = 1                               # time step
length = len(time_ser_1.values)         # lenth of a series
m = 4                                   # embedding dimension
tau = 1                                 # time delay

LZC = []                                # classical Lempel-Ziv complexity
PLZC = []                               # permutation Lempel-Ziv complexity

for i in tqdm(range(0, length-window, tstep)): # fragments of window length w
ith a step "tstep"

# choose a fragment
    fragm = time_ser_1.iloc[i:i+window].copy()

# perform a series transformation procedure
    fragm = transformation(fragm, ret_type)

# calculate the classical Lempel-Ziv complexity
    lzc, _ = nk.complexity_lempelziv(fragm)

# and the permutation Lempel-Ziv complexity
    plzc, _ = nk.complexity_lempelziv(fragm,
                                      delay=tau,
                                      dimension=m,
                                      permutation=True)
```

```
# and add the results to the array of values
    LZC.append(lzc)
    PLZC.append(plzc)
```

Saving the results to the text files:

```
np.savetxt(f"lzc_name={symbol_1}_window={window}_step={tstep}_rettype={ret_ty
pe}.txt" , LZC)
np.savetxt(f"plzc_name={symbol_1}_window={window}_step={tstep}_ \
    rettype={ret_type}_m={m}_tau={tau}.txt" , PLZC)
```

And visualizing them:

```
fig, ax = plt.subplots(1, 1)

ax2 = ax.twinx()
ax3 = ax.twinx()
ax2.spines.right.set_position(("axes", 1.03))
ax3.spines.right.set_position(("axes", 1.12))

p1, = ax.plot(time_ser_1.index[window:length:tstep],
                time_ser_1.values[window:length:tstep],
"b-",
                label=fr"{symbol_1}")
p2, = ax2.plot(time_ser_1.index[window:length:tstep],
                LZC,
'gold',
                label=fr"$LZC$")
p3, = ax3.plot(time_ser_1.index[window:length:tstep],
                PLZC,
'red',
                label=fr"$PLZC$")


ax.set_xlabel(xlabel)
ax.set_ylabel(f"{symbol_1}")
ax.yaxis.label.set_color(p1.get_color())
ax2.yaxis.label.set_color(p2.get_color())
ax3.yaxis.label.set_color(p3.get_color())

tkw = dict(size=3, width=1.5)

ax.tick_params(axis='x', rotation=45, **tkw)
ax.tick_params(axis='y', colors=p1.get_color(), **tkw)
ax2.tick_params(axis='y', colors=p2.get_color(), **tkw)
ax3.tick_params(axis='y', colors=p3.get_color(), **tkw)
ax3.legend(handles=[p1, p2, p3])

plt.savefig(f"plzc_lzc_name={symbol_1}_ \
    window={window}_step={tstep}_ \
    rettype={ret_type}_m={m}_tau={tau}.jpg")

plt.show();
```

Fig. 1.2 shows the comparative dynamics of the S&P 500 (a), the Hang Seng Index (b), the DAX (c), the BSE Sensex (d) and their classical monoscale LZ complexity and its permutation version.

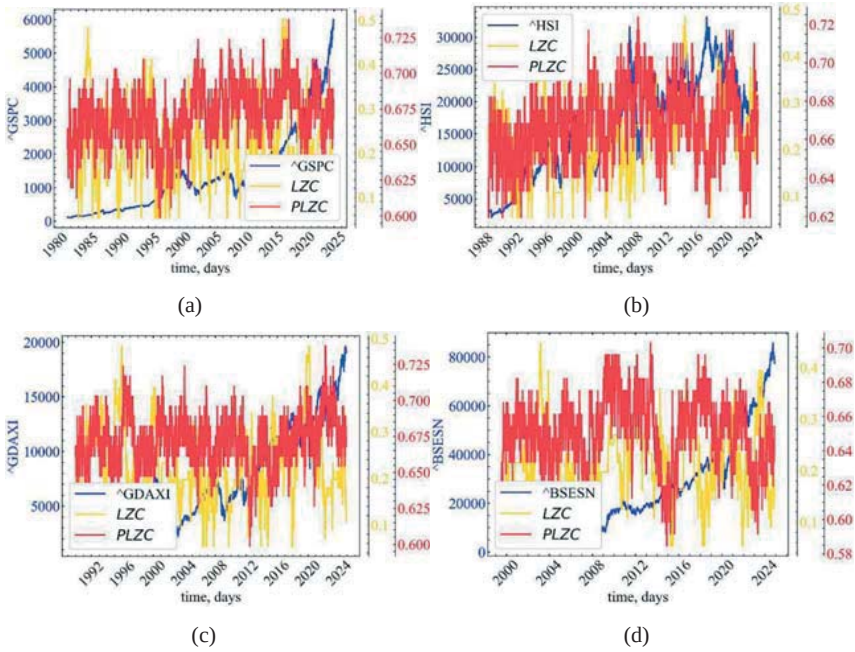


Fig. 1.2: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), their classical monoscale LZ complexity, and its permutation version

This figure shows that the 2 measures behave chaotically. In general, both of them decline before crisis events in stock market indices. It is worth investigating the multiscale dynamics of LZ measure for more meaningful conclusions.

However, even this approach may not be enough. The fact is that complex signals exhibit their inherent complexity on different spatial and temporal scales, i.e., they have scale invariant properties. In particular, they are manifested through power laws of distribution. Therefore, mono-scale calculations of algorithmic complexity may be unacceptable and lead to erroneous conclusions.

To overcome such difficulties, multiscale methods are used, and we will now consider them.

1.3 A granularity procedure for multiscale time series analysis. Multiscale measures of complexity

The idea of this group of methods includes two sequential procedures:

- the process of "coarse graining" of the initial time series – averaging data on non-overlapping segments, the size of which (the averaging window) will increase by one when moving to the next largest scale;
- calculation of a certain (still mono-scale) complexity indicator at each of the scales.

The process of "coarse-graining" ("granulation") consists in averaging successive counts of a series within non-overlapping windows, the size of which τ increases when moving from scale to scale. Each element of the "granular" time series y_j^τ is determined according to the expression [104]:

$$y_j^\tau = \frac{1}{\tau} \sum_{i=(j-1)\tau+1}^{j\tau} x_i\,,\ 1 \le j \le N/\tau,$$

where τ characterizes the scaling factor. The length of each "granular" row depends on the window size and is equal to N/τ. For a scale equal to 1, the "granular" series is simply identical to the original one.

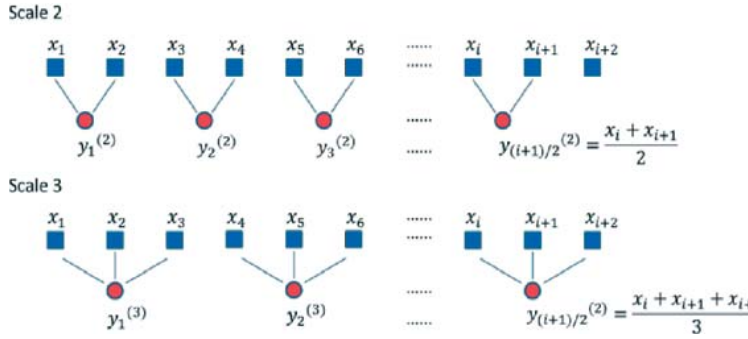


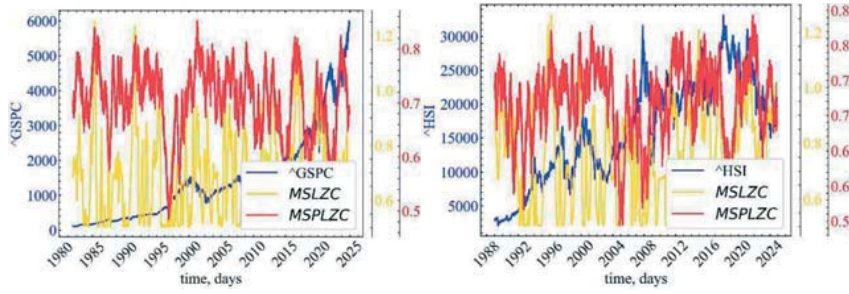
Fig. 1.3: Schematic illustration of the process of coarse-graining ("granulation") of the original time series for scales 2 and 3

Let us calculate the window dynamics of the multiscale LZ indicators. We return the total complexity of the LZ over all scales:

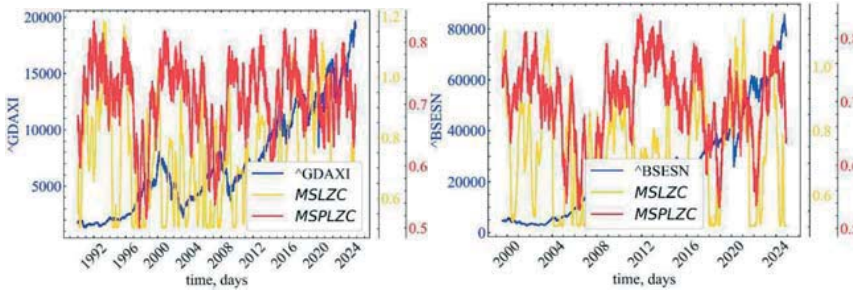
```
ret_type = 4                        # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

window = 500                        # window length
tstep = 1                           # time step of the sliding window
length = len(time_ser_1.values)     # length of a series
m = 3                               # embedding dimension
tau = 1                             # time delay

MSLZC = []                          # multiscale Lempel-Ziv complexity
MSPLZC = []                         # multiscale permutation Lempel-Ziv complex
ity

for i in tqdm(range(0, length-window, tstep)): # fragments of window length w
ith a step “tstep”

# select a fragment
    fragm = time_ser_1.iloc[i:i+window].copy()

# perform the series transformation procedure
    fragm = transformation(fragm, ret_type)

# calculate the multiscale Lempel-Ziv complexity
    mslzc, _ = nk.entropy_multiscale(fragm)

# and the multiscale permutation Lempel-Ziv complexity
    msplzc, _ = nk.entropy_multiscale(fragm,
                                      delay=tau,
                                      dimension=m,
                                      permutation=True)


# and add the results to the array of values
    MSLZC.append(mslzc)
    MSPLZC.append(msplzc)

np.savetxt(f"mslzc_name={symbol_1}_window={window}_step={tstep}_ \
    rettype={ret_type}.txt" , MSLZC)
np.savetxt(f"msplzc_name={symbol_1}_window={window}_step={tstep}_ \
    rettype={ret_type}_m={m}_tau={tau}.txt" , MSPLZC)

fig, ax = plt.subplots(1, 1)

ax2 = ax.twinx()
ax3 = ax.twinx()
ax2.spines.right.set_position(("axes", 1.03))
ax3.spines.right.set_position(("axes", 1.12))
```

```
p1, = ax.plot(time_ser_1.index[window:length:tstep],
                time_ser_1.values[window:length:tstep],
"b-",
                label=fr"{symbol_1}")
p2, = ax2.plot(time_ser_1.index[window:length:tstep],
                MSLZC,
'gold',
                label=fr"$MSLZC$")
p3, = ax3.plot(time_ser_1.index[window:length:tstep],
                MSPLZC,
'red',
                label=fr"$MSPLZC$")

ax.set_xlabel(xlabel)
ax.set_ylabel(f"{symbol_1}")
ax.yaxis.label.set_color(p1.get_color())
ax2.yaxis.label.set_color(p2.get_color())
ax3.yaxis.label.set_color(p3.get_color())

tkw = dict(size=3, width=1.5)

ax.tick_params(axis='x', rotation=45, **tkw)
ax.tick_params(axis='y', colors=p1.get_color(), **tkw)
ax2.tick_params(axis='y', colors=p2.get_color(), **tkw)
ax3.tick_params(axis='y', colors=p3.get_color(), **tkw)
ax3.legend(handles=[p1, p2, p3])

plt.savefig(f"msplzc_mslzc_name={symbol_1}_ \
    window={window}_step={tstep}_ \
    rettype={ret_type}_m={m}_tau={tau}.jpg")

plt.show();
```

In Fig. 1.4, the comparative dynamics of the S&P 500 (a), the Hang Seng Index (b), the DAX (c), the BSE Sensex (d) and their classical multiscale Lempel-Ziv complexity and its permutation analog can be observed.



(a) (b)



(c) (d)

Fig. 1.4: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), their classical multiscale LZ complexity and its permutation analog

Now the picture is more clear: both measures behave synchronously and decline in crisis and pre-crisis periods, indicating an increase in the degree of determinism and self-organization of the market.

1.4 Informational measures of complexity

1.4.1 Fisher's information

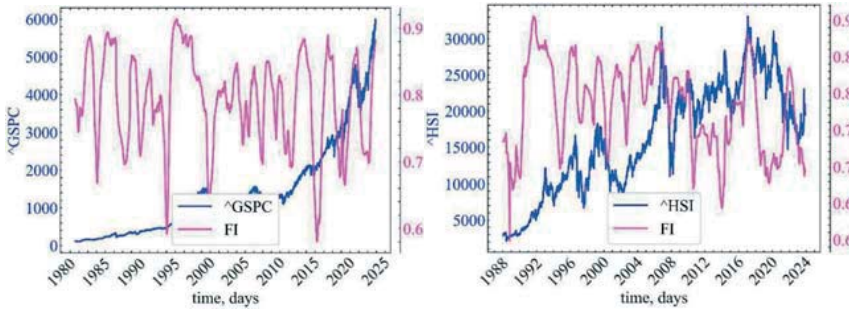
Fisher information (FI) was introduced by R. A. Fisher in 1922 as a measure of "internal precision" in the theory of statistical estimation [132]. It is central to many statistical applications that go far beyond complexity theory. It measures the amount of information that an observed random variable carries about an unknown parameter. Complexity analysis measures the amount of information a system has "about itself". It is based on the singular value decomposition of the reconstructed phase space. The FI value is usually uncorrelated with other indicators of complexity (the more information a system hides about itself, the more predictable and, accordingly, the less complex it is).

First of all, we set the parameters for calculations:

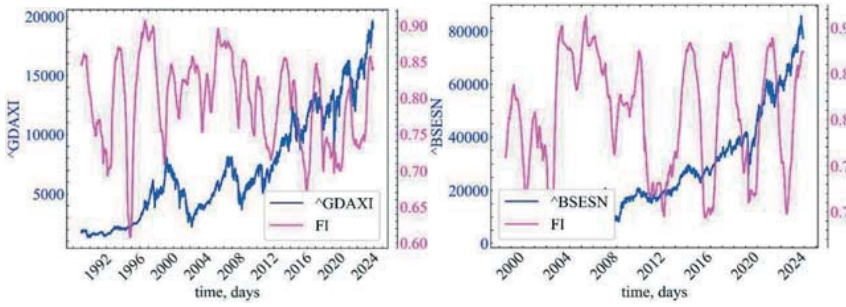
```
ret_type = 6                        # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
```

```
# 6 – standardized series

window = 500                        # window length
tstep = 1                           # time step of the sliding window
length = len(time_ser_1.values)     # length of a series
m = 3                               # embedding dimension
tau = 1                             # time delay

fisher = []                         # Fisher information

for i in tqdm(range(0, length-window, tstep)):


# select a fragment
    fragm = time_ser_1.iloc[i:i+window].copy()

# perform the series transformation procedure
    fragm = transformation(fragm, ret_type)

    fish_inf, _ = nk.fisher_information(signal=fragm, dimension=m, delay=tau)

# and add the result to the array of values
    fisher.append(fish_inf)

np.savetxt(f"fisher_inf_name={symbol_1}_window={window}_step={tstep}_rettype=
{ret_type}_dimension={m}_delay={tau}.txt", fisher)

values_plot = time_ser_1.values[window:length:tstep], fisher
ylabels = ylabel_1, "FI"
file_name = f"fisher_name={symbol_1}_window={window}_step={tstep}_rettype={re
t_type}_dimension={m}_delay={tau}"

plot_pair(time_ser_1.index[window:length:tstep], values_plot, xlabel, ylabels
, file_name)
```

In Fig. 1.5, the comparative dynamics of the S&P 500 (a), the Hang Seng Index (b), the DAX (c), the BSE Sensex (d) and their Fisher's information indicator can be observed.



(a) (b)



(c) (d)

Fig. 1.5: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their Fisher's information indicator

Fig. 1.5 shows that the Fisher index decreases in crisis and pre-crisis periods, which indicates a decline in the amount of information needed to describe the self-organized dynamics of financial crises and an increase in correlation between traders' actions in the market.

1.4.2 Hjorth's complexity and its parameters

Hjorth's parameters are statistical property measures that were originally introduced by **Hjorth** [28] to describe the general characteristics of electroencephalogram signals. The parameters are activity, mobility, and complexity:

1. The **Activity** parameter is simply the variance of the signal, which corresponds to the average power of the signal (if its average value is 0):

$$Activity = \sigma^2_{signal}.$$

2. The **Mobility** parameter is the average frequency or proportion of the standard deviation of the power spectrum. It is defined as the square root of the variance of the first derivative of the signal divided by the variance of the signal:

$$Mobility = \frac{\sigma_{dd}/\sigma_d}{Complexity}.$$

3. The **Complexity** parameter gives an estimate of the signal bandwidth, indicating the similarity of the waveform to a pure sine wave (for which the value converges to 1). In other words, it is a characteristic of "excessive detail" in relation to the "softest" possible waveform. The "Complexity" parameter is defined as the ratio of the mobility of the first derivative of the signal to the mobility of the signal itself:

$$Complexity = \sigma_d / \sigma_{signal},$$

where d and dd represent the first and second derivatives of the signal, respectively.

```
ret_type = 1                        # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

window = 500                        # window length
tstep = 1                           # time step of the sliding window
length = len(time_ser_1.values)     # length of a series
activity = []
mobility = []
complexity = []

for i in tqdm(range(0, length-window, tstep)): # fragment with the length win
dow and delay tstep

# select a fragment
    fragm = time_ser_1.iloc[i:i+window].copy()
# perform the series transformation procedure
    fragm = transformation(fragm, ret_type)
# calculate the Hjorth's complexity indicators
    cmpl, info = nk.complexity_hjorth(fragm)

# and add the result to the array of values
    activity.append(info['Activity'])
    mobility.append(info['Mobility'])
    complexity.append(cmpl)

np.savetxt(f"activity_name={symbol_1}_window={window}_ \
    step={tstep}_rettype={ret_type}.txt", activity)
np.savetxt(f"mobility_name={symbol_1}_window={window}_ \
    step={tstep}_rettype={ret_type}.txt", mobility)
np.savetxt(f"complexity_name={symbol_1}_window={window}_ \
    step={tstep}_rettype={ret_type}.txt", complexity)

fig, ax = plt.subplots(1, 1)
```

```
ax2 = ax.twinx()
ax3 = ax.twinx()
ax4 = ax.twinx()

ax2.spines.right.set_position(("axes", 1.03))
ax3.spines.right.set_position(("axes", 1.16))
ax4.spines.right.set_position(("axes", 1.24))

p1, = ax.plot(time_ser_1.index[window:length:tstep],
              time_ser_1.values[window:length:tstep],
"b-", label=fr"{ylabel_1}")
p2, = ax2.plot(time_ser_1.index[window:length:tstep],
               activity, "r--", label=r"$Act$")
p3, = ax3.plot(time_ser_1.index[window:length:tstep],
               mobility, "g-", label=r"$Mob$")
p4, = ax4.plot(time_ser_1.index[window:length:tstep],
               complexity, "m-", label=r"$Comp$")

ax.set_xlabel(xlabel)
ax.set_ylabel(f"{ylabel_1}")
ax.yaxis.label.set_color(p1.get_color())
ax2.yaxis.label.set_color(p2.get_color())
ax3.yaxis.label.set_color(p3.get_color())
ax4.yaxis.label.set_color(p4.get_color())

tkw = dict(size=4, width=1.5)

ax.tick_params(axis='y', colors=p1.get_color(), **tkw)
ax.tick_params(axis='x', rotation=45, **tkw)
ax2.tick_params(axis='y', colors=p2.get_color(), **tkw)
ax3.tick_params(axis='y', colors=p3.get_color(), **tkw)
ax4.tick_params(axis='y', colors=p4.get_color(), **tkw)
ax4.legend(handles=[p1, p2, p3, p4])

plt.savefig(f"hjorth_name={symbol_1}_ret={ret_type}_wind={window}_step={tstep
}.jpg")
plt.show();
```

Fig. 1.6 shows the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their Hjorth's indicators of activity, mobility, and complexity.

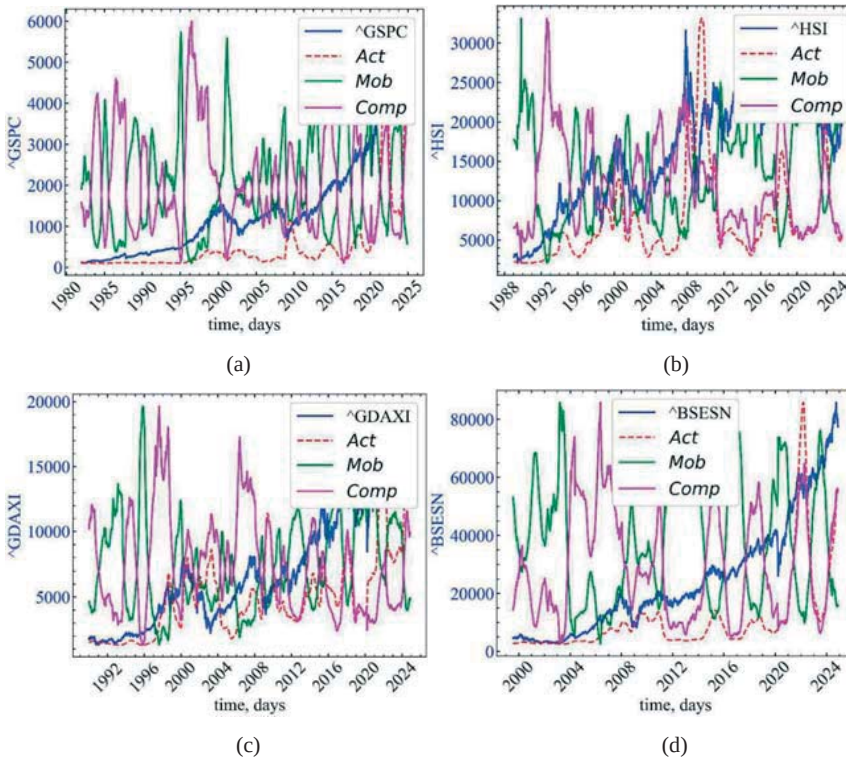


Fig. 1.6: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their Hjorth's indicators of activity, mobility, and complexity

Obviously, the activity parameter (Act) seems to be the least informative, as it only indicates an increase in the total variance of the signal. The issue of premature identification of the growth of a crisis phenomenon is best solved by the Mobility indicator (Mob). We can see that this indicator increased before the crashes of 1997, 2001, during 2008-2009, and COVID-19 crisis. The Hjorth's complexity measure ($Comp$) reacts in an asymmetric way: while mobility increases, the complexity measure decreases, indicating that the system tends to be more periodic or correlated.

1.4.3 Decorrelation time

The **decorrelation time** (DT) is defined as the time (in samples) of the first zero crossing of the autocorrelation function. A shorter DT corresponds to a less correlated signal. For example, a decrease in DT in electroencephalogram signals is observed before seizures, which is associated with a decrease in low frequency power [64].

```
ret_type = 1                        # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series
window = 500                        # window length
tstep = 1                           # time step of the sliding window
length = len(time_ser_1.values)     # length of a series

decorrelation_time = []             # array for decorrelation time

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser_1.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)
    dec_time, _ = nk.complexity_decorrelation(fragm)
    decorrelation_time.append(dec_time)

np.savetxt(f"dec_time_name={symbol_1}_window={window}_ \
    step={tstep}_rettype={ret_type}.txt", decorrelation_time)

values_plot = time_ser_1.values[window:length:tstep], decorrelation_time
ylabels = ylabel_1, "DT"
file_name = f"dec_time_name={symbol_1}_window={window}_ \
    step={tstep}_rettype={ret_type}"

plot_pair(time_ser_1.index[window:length:tstep], values_plot,
            xlabel, ylabels, file_name)
```

Fig. 1.7 demonstrates the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their decorrelation time.

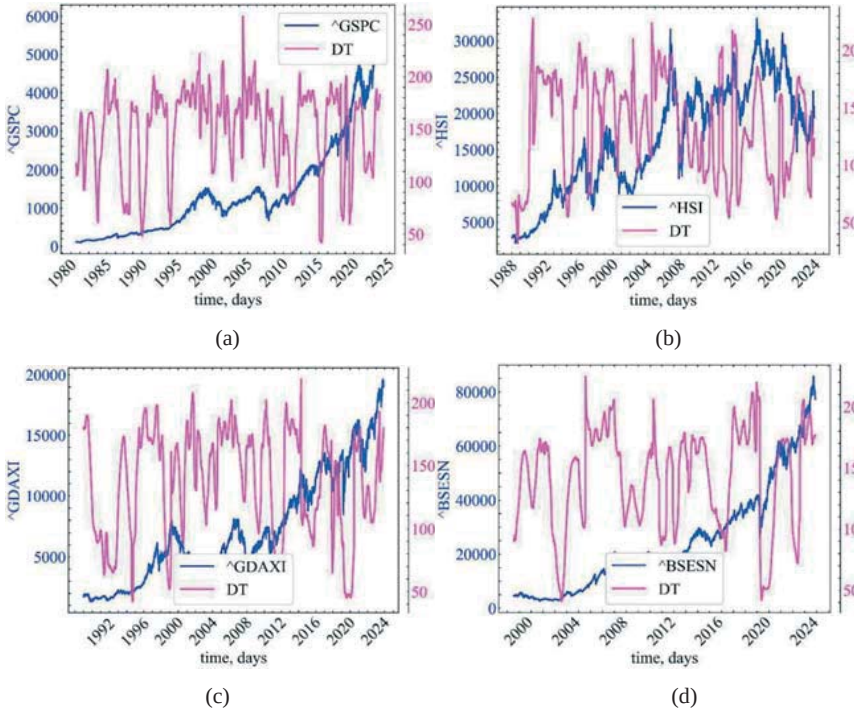


Fig. 1.7: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their DT indicator

The decorrelation time increases in the pre-crash period, indicating that the system is more correlated during this period.

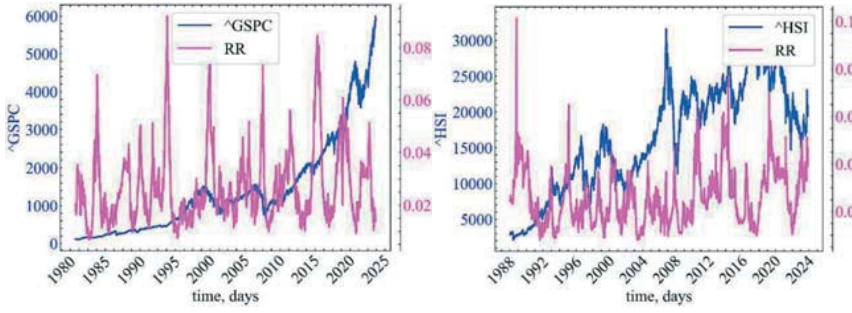
1.4.4 Relative roughness (irregularity, sharpness)

Relative roughness (RR) is the ratio of local variance (autocovariance with lag 1) to global variance (autocovariance with lag 0), which can be used to classify various "noises". This indicator can also be used as an index of the applicability of fractal analysis [162].

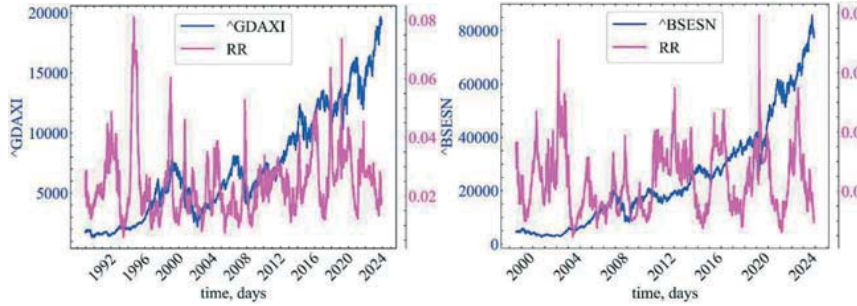
```
ret_type = 1                        # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
```

```
# 6 – standardized series
window = 500                        # window length
tstep = 1                           # time step of the sliding window
length = len(time_ser_1.values)     # length of a series

relative_roughness = []             # relative roughness

for i in tqdm(range(0, length-window, tstep)):
    fragm = time_ser_1.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)
    rr, _ = nk.complexity_relativeroughness(fragm)

# and add the result to the array of values
    relative_roughness.append(rr)

np.savetxt(f"rel_rough_name={symbol_1}_window={window}_ \
    step={tstep}_rettype={ret_type}.txt", relative_roughness)

values_plot = time_ser_1.values[window:length:tstep], relative_roughness
ylabels = ylabel_1, "RR"
file_name = f"rel_rough={symbol_1}_window={window}_ \
    step={tstep}_rettype={ret_type}"

plot_pair(time_ser_1.index[window:length:tstep], values_plot,
          xlabel, ylabels, file_name)
```

Fig. 1.8 shows the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their relative roughness indicator.



(a) (b)



(c) (d)

Fig. 1.8: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their relative roughness indicator

The relative roughness indicator shows that crash events are characterized by an increase in roughness. This kind of behavior is an indicator of increasing market noise activity: correlation characteristics and overall market variation.

1.5 Entropy analysis of complex systems

The issue of the dynamics of development and functioning of complex systems can be considered in two ways:

- as a study of noise activity;
- as a deterministic case with a certain degree of order.

In recent years, several approaches have been used to identify the mechanisms underlying the evolution of complex systems. Particularly useful results have been obtained by studying them using the methods of random matrix theory, mono- and multifractal analysis, chaos theory with reconstruction of the system trajectory in phase space, recurrence analysis, etc. We have reviewed these methods in previous papers. However, the use of some of these methods imposes requirements for the stationarity of the data under study, requires long time series, and complex calculation of several parameters.

Another well-known approach to modeling the characteristics of complex systems is to calculate the characteristics of different types of entropy.

The concept of thermodynamic entropy as a measure of system chaos is well known in physics, but in recent years the concept of entropy has been applied to complex systems of other objects (biological, economic, social, etc.). For example, one of the most commonly used methods for determining entropy is based on the calculation of the Fourier power spectrum and is used to study time series of various nature. However, using the discrete Fourier transform to analyze time series has its drawbacks, in particular, the results are affected by the non-stationarity of the series, the variation of their length from hundreds to hundreds of thousands, and the limitations of the method itself (the invariance of the frequency-time characteristics throughout the entire time of the system's operation). This raises the question of calculating entropy values using other methods.

The **thermodynamic entropy** S, often simply referred to as **entropy**, in chemistry and thermodynamics is a measure of the amount of energy in a physical system that cannot be used to do work. It is also a measure of the disorder present in the system.

The concept of entropy was first introduced in 1865 by Rudolf Clausius [133]. He defined the change in entropy of a thermodynamic system during a reversible process as the ratio of the change in the total amount of heat ΔQ to the absolute temperature T:

$$\Delta S = \Delta Q / T.$$

Rudolf Clausius gave the value S the name "entropy", which comes from the Greek word τροπή, "change" (alteration, transformation).

In 1877, Ludwig Boltzmann [97] realized that the entropy of a system can refer to the number of possible "microstates" (microscopic states) consistent with their thermodynamic properties. Consider, for example, an ideal gas in a vessel. A microstate is defined as the positions and momenta of each atom that makes up the system. Connectivity requires us to consider only those microstates for which: (i) the location of all parts is limited by the boundaries of the vessel, (ii) the kinetic

energies of the atoms are summed to obtain the total energy of the gas. Boltzmann postulated that

$$S = k_B \ln \Omega,$$

where the constant $k_B = 1{,}38 \cdot 10^{-23}\text{J/K}$ is now known as the Boltzmann constant, and Ω is the number of microstates that are possible in the existing macroscopic state. This postulate, known as Boltzmann's principle, can be evaluated as the beginning of statistical mechanics, which describes thermodynamic systems using the statistical behavior of components. Boltzmann's principle relates the microscopic properties of a system (Ω) to one of its thermodynamic properties (S).

According to Boltzmann's definition, entropy is simply a function of state. Moreover, since (Ω) can only be a positive integer, the entropy must be positive, based on the properties of the logarithm.

In the case of discrete states of quantum mechanics, the number of states is counted in the usual way. In classical mechanics, the microscopic state of a system is described by the coordinates q_i and momenta p_i of individual particles, which take continuous values. In this case

$$S = k_B \ln \frac{1}{(2\pi\hbar)^s} \int \prod_{i=1}^{s} dq_i dp_i,$$

where s is the number of independent coordinates, $\hbar$ is the reduced Planck constant, and integration is performed over a region of phase space corresponding to a certain macroscopic state.

Claude Shannon [35] proposed a formula for estimating the uncertainty of coded information in communication channels, called Shannon entropy:

$$S = -k \sum_{i=1}^{n} p_i \ln p_i,$$

where p_i is the probability that character i occurs in a code containing N characters, and k is a dimensional factor.

We will calculate it using a sliding window procedure:

```
ret_type = 1
window = 500
tstep = 1
length = len(time_ser_1.values)
log_base = np.exp(1)

shannon = []                              # array for Shannon entropy values

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser_1.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

# calculate Shannon entropy
    p, be = np.histogram(fragm,           # calculate the probability density f
unction
                         bins='auto',
                         density=True)
    r = be[1:] - be[:-1]                  # find dx
    P = p * r                             # represent probability as f(x)*dx
    P = P[P!=0]                           # filter by all non-zero probabilitie
s

    sh_ent, _ = nk.entropy_shannon(freq=P, base=log_base) # calculate entropy

    sh_ent /= np.log(len(P))                              # and normalize

# and add the result to the array of values
    shannon.append(sh_ent)

np.savetxt(f"shannon_ent_name={symbol_1}_window={window}_ \
    step={tstep}_rettype={ret_type}.txt" , shannon)

values_plot = time_ser_1.values[window:length:tstep], shannon
ylabels = ylabel_1, "ShEn"
file_name = f"shannon_ent_name={symbol_1}_window={window}_ \
    step={tstep}_rettype={ret_type}"

plot_pair(time_ser_1.index[window:length:tstep],
            values_plot, xlabel, ylabels, file_name)
```

In Fig. 1.9 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their Shannon entropy.

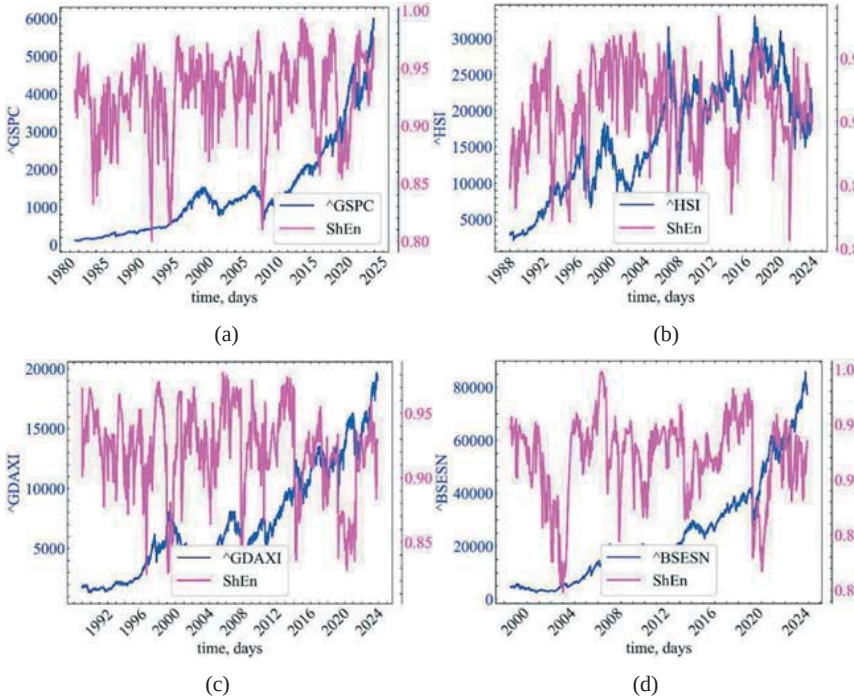


Fig. 1.9: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their ShEn

As we can see from the figure above, ShEn responds with a decline to crisis periods of the stock indices, which indicates an increase in the degree of correlation of the system, its determinism.

The connection between entropy and information can be seen in the following example. Let's consider a body at absolute zero temperature, and assume that we have complete information about the coordinates and momentum of each particle. For simplicity, let's assume that the momenta of all particles are zero. In this case, the thermodynamic probability is one and the entropy is zero. At finite temperatures, the entropy in equilibrium reaches a maximum. We can measure all the macro parameters that characterize this macro state. However, we know virtually nothing about the microstate of the system. To be more precise, we know

that a given macro state can be realized with the help of a very large number of micro states. Thus, zero entropy corresponds to complete information (the degree of ignorance is zero), and maximum entropy corresponds to complete ignorance of microstates (the degree of ignorance is maximum).

In information theory, entropy is defined as the amount of information. Let P be the a priori probability of an event (the probability before the experiment), and P_1 be the probability of this event after the experiment. For simplicity, we assume that $P_1 = 1$. According to Shannon, the amount of information I that gives an accurate answer (after the experiment) is

$$I = k \log P.$$

By definition, this amount of information is equal to one bit.

The physical meaning of I represents a measure of our ignorance. In other words, I is the information we can get by solving a problem. In the example (a body at absolute zero temperature) discussed above, the measure of our ignorance is zero, since $P = 1$. After the experiment, we get zero information $I = 0$, since everything was known before the experiment. If we consider a body at finite temperatures, the number of microstates, and hence P, is very large before the experiment. After the experiment, we get a lot of information, since we know the coordinates and momenta of all the particles.

The analogy between the amount of information and the entropy S, determined from Boltzmann's principle, is obvious. It is enough to set the multiplier K equal to the Boltzmann constant k_B and use the natural logarithm. It is for this reason that the value of I is called information entropy. Information entropy (the amount of information) was defined by analogy with ordinary entropy, and it has properties characteristic of ordinary entropy: additivity, extreme properties, etc. However, it is impossible to equate ordinary entropy with information entropy, since it is unclear what the second law of thermodynamics has to do with information. Recall that an extensive quantity is a characteristic of a system that increases with the size of the system, i.e., if our system consists of two

independent subsystems A and B, then the entropy of the whole system can be obtained by adding the entropies of the subsystems:

$$S(A+B) = S(A) + S(B).$$

This is the property that means the extensivity, or additivity, of entropy.

Let's look at how we can use entropy indicators as indicators or precursors of crisis events. First of all, let's import and install the necessary modules for further work:

```
!pip install EntropyHub

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import yfinance as yf
import neurokit2 as nk
import EntropyHub as eh
import warnings
import scienceplots
from tqdm import tqdm

warnings.filterwarnings('ignore')
```

1.5.1 Approximation entropy

Approximate Entropy (ApEn) is a "regularity measure" [148, 149] that determines the ability to predict fluctuations in time series [167]. Intuitively, it means that the presence of repeating patterns (sequences of a certain length constructed from consecutive numbers in a series) of fluctuations in a time series leads to greater predictability of the time series compared to series without repeating patterns. A relatively large value of ApEn shows the probability that similar patterns of observations will not follow each other. In other words, a time series containing a large number of repeating patterns has a relatively small ApEn value, while the ApEn value for a less predictable (more complex) process is larger.

When calculating ApEn for a given time series S_N, consisting of N values $t(1), t(2), t(3), \ldots, t(N)$, two parameters, d_E and r, are selected. The first of these parameters, m, indicates the length of the pattern, and the second, r, defines the similarity criterion. Subsequences of elements of the time series S_N, consisting of

d_E numbers taken starting from number i, are studied and are called vectors $p_{d_E}(i)$. Two vectors (templates), $p_{d_E}(i)$ and $p_{d_E}(j)$, are similar if all differences of pairs of their respective coordinates are less than the value of r, i.e. if

$$|t(i+k) - t(j+k)| < r \quad \text{for} \quad 0 \le k < d_E.$$

For the considered set P_{d_E} of all vectors of length d_E of the time series S_N, we can calculate the values

$$C_{id_E}(r) = n_{id_E}(r)/(N - d_E + 1),$$

where $n_{id_E}(r)$ is the number of vectors in P_{d_E} that are similar to the vector $p_{d_E}(i)$ (given the chosen similarity criterion r). The value of $C_{id_E}(r)$ is the proportion of vectors of length d_E that have similarity to a vector of the same length whose elements start with the number i. For a given time series, the values of $C_{id_E}(r)$ are calculated for each vector in P_{d_E}, and then the average value of $C_{d_E}(r)$ is found, which reflects the prevalence of similar vectors of length d_E in the series S_N. Directly, the approximation entropy for the time series S_N using the vectors of length d_E and the similarity criterion r is determined by the formula:

$$ApEn(S_N, d_E, r) = \ln C_{d_E}(r) - \ln C_{d_E+1}(r).$$

That is, as the natural logarithm of the ratio of the repeatability of vectors of length d_E to the repeatability of vectors of length $d_E + 1$.

Thus, if similar vectors are found in the time series, ApEn will estimate the logarithmic probability that the following intervals after each vector will be different. Smaller ApEn values correspond to a higher probability that vectors are followed by similar ones. If the time series is very irregular, the presence of similar vectors cannot be predicted and the ApEn value is relatively large.

Note that ApEn is an unstable characteristic to the input data, as it depends quite strongly on the parameters d_E and r.

```
window = 500
tstep = 1

m = 3             # embedding dimension
tau = 1           # time delay
r = 0.45          # similarity parameter
```

```
ret_type = 6     # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

length = len(time_ser.values)  # length of a series

ApEn = []                      # an array for storing entropy values

for i in tqdm(range(0, length-window, tstep)): #

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

# calculate the approximation entropy
    Ap, _ = nk.entropy_approximate(signal=fragm,
                                   dimension=m,
                                   delay=tau,
                                   tolerance=r,
                                   corrected=False)
    ApEn.append(Ap)
```

Save the value of ApEn to a text file:

```
np.savetxt(f"ApEn_name={symbol}_window={window}_step={tstep}_\
           dim={m}_tau={tau}_radius={r}_sertype={ret_type}.txt", ApEn)
```

Defining the labels for the figures and the names of the saved figures:

```
label_apen = fr'$ApEn$'

file_name_apen = f"ApEn_name={symbol}_window={window}_step={tstep}_\
           dim={m}_tau={tau}_radius={r}_sertype={ret_type}"
```

Plotting the results:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          ApEn,
          ylabel,
          label_apen,
          xlabel,
          file_name_apen)
```

In Fig. 1.10 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their approximate entropy.

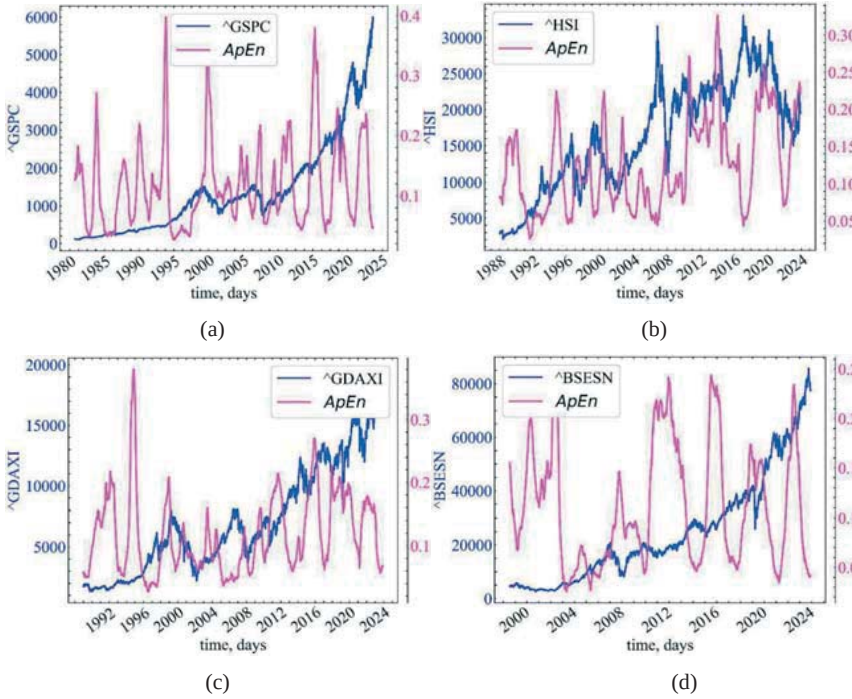


Fig. 1.10: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their ApEn

As can be seen from Fig. 1.10, the approximation entropy falls at crisis and pre-crisis moments. This indicates that the average value of the correlation integral obtained for the phase space of dimension d_{E+1} does not differ much from that obtained for the phase space of dimension d_E. That is, when the space is reconstructed in different dimensions, all points are simply close enough to each other, despite the geometric transformations of the stock market attractor. This indicates a fairly high degree of correlation between the price fluctuations of the stock indices and the focus of market traders on a single trend.

1.5.2 Fuzzy entropy

One of the modifications of Shannon entropy is **Fuzzy entropy** (FuzzEn) [16, 79, 171]. This approach excludes self-similarity between the studied vectors,

and instead of the Heaviside function, which gives either 0 or 1 for similar vectors, a fuzzy membership function is used, which in the case of FuzzEn will associate the similarity between two vectors with a real value in the range $[0, 1]$. The difference can be seen at the stage of constructing the contribution vector, where we perform detrending for the reconstructed vectors:

$$\vec{X}(i) = \left\{x(i), x(i+1), \ldots, x(i+d_E-1) - x_0(i)\right\},\ i = 1, \ldots, N - d_E + 1,$$

where $x_0(i) = (d_E)^{-1} \sum_{j=0}^{d_E-1} x\,(i+j)$. Next, for consecutive embedded vectors, we find the distance

$$d\left[\vec{X}(i), \vec{X}(j)\right] = \max\left|\vec{X}(i) - \vec{X}(j)\right|, i \geq 1, j \leq N - d_E + 1.$$

In the classical ApEn, distance values are passed through the Heaviside function. The fuzzy modification uses membership functions to measure the membership of one trajectory to another:

$$D_{i,j} = \mu\left(d\left[\vec{X}(i), \vec{X}(j)\right]\right),$$

where $\mu = \exp(-x^{r_2}/r_1)$, and r_1 and r_2 are the width and gradient of the exponential function.

Next, the following function is calculated, which is similar to the correlation integral in classical ApEn:

$$\phi^{d_E} = \frac{1}{(N - d_E + 1)} \sum_{i=1}^{N-d_E+1} \left(\frac{1}{N - d_E} \sum_{j=1, j\neq i}^{N-d_E} D_{i,j} \right).$$

Finally,

$$FuzzEn\left(\vec{X}, d_E\right) = -[\ln \phi^{d_E+1} - \ln \phi^{d_E}].$$

```
window = 500        # window length
tstep = 1           # time step

m = 3               # embedding dimension
tau = 1             # time delay

characteristic_func = "default" # type of membership function:
                                # default,
                                # sigmoid,
                                # gudermannian,
                                # linear
```

```
r = (0.4, 2.0)                  # parameters that are passed to the membershi
p function:
# for ‘default’ and ‘sigmoid’ – 2 values of r,
# for ‘gudermannian’ and ‘linear’ - 1 value of r,

ret_type = 6 # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

length = len(time_ser.values)  # length of a series

FuzzEn = [] # an array for storing entropy values

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

# calculation of fuzzy entropy
    Fuzz, _, _ = eh.FuzzEn(Sig=fragm, m=m, tau=tau, Fx=characteristic_func, r
=r)
    FuzzEn.append(Fuzz[-1]) # add the calculated value to the array of values
```

Save FuzzEn value to a text file:

```
np.savetxt(f"FuzzEn_name={symbol}_window={window}_step={tstep}_\
        dim={m}_tau={tau}_radius={r}_sertype={ret_type}_\
        memberfunc={characteristic_func}.txt", FuzzEn)
```

Defining the labels for the figures and the names of the saved figures:

```
label_fuzzen = fr'$FuzzEn$'

file_name_fuzzen = f"FuzzEn_name={symbol}_window={window}_step={tstep}_\
        dim={m}_tau={tau}_radius={r}_sertype={ret_type}_\
        memberfunc={characteristic_func}"
```

Plotting the results:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          FuzzEn,
          ylabel,
          label_fuzzen,
          xlabel,
          file_name_fuzzen,
          clr='red')
```

Fig. 1.11 shows the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their sample entropy.

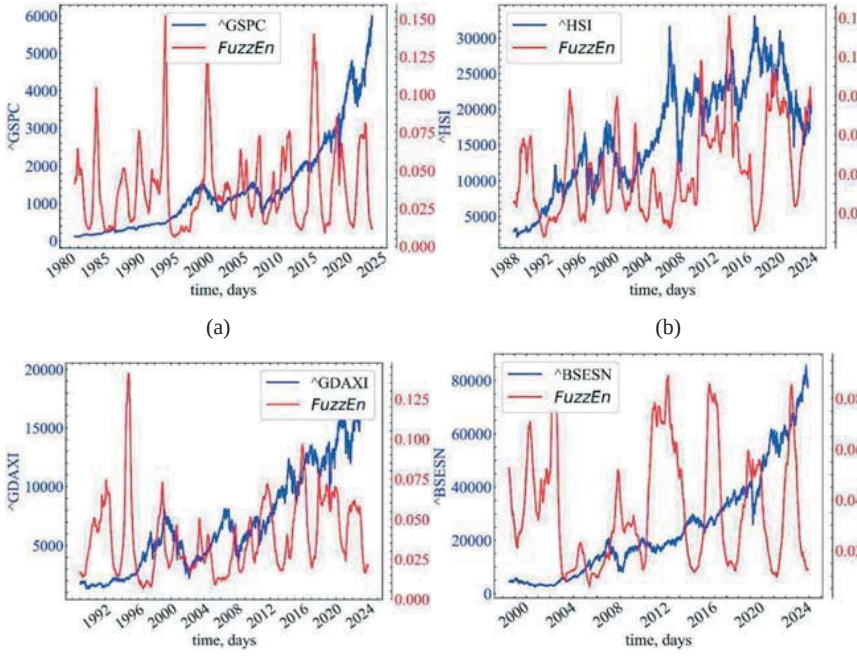


Fig. 1.11: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their FuzzEn

Fig. 1.11 demonstrates the downward dynamics of fuzzy entropy in crisis and pre-crisis periods, which works similarly to approximation entropy. It follows that fuzzy entropy also indicates an increase in the correlation of the system and its trend resistance due to the unidirectionality of traders' opinions on the future dynamics of the stock market.

1.5.3 Sample entropy

The calculation of ApEn takes into account the similarities of a particular vector $p_n(i)$ to itself, which is used to avoid a possible value of ln0 in the absence of similar vectors. However, this feature leads to the leveling of two important characteristics in the similarity entropy:

- ApEn is highly dependent on the length of the pattern (vector) under consideration and is lower than expected for vectors of small dimensionality;
- ApEn does not take into account the relative density of the data.

This means that when the ApEn value for one series is higher than for another, it should remain so (but is not) for any possible initial conditions. This conclusion is all the more important since ApEn is recommended as a measure of comparison between two data sets by different authors.

Taking into account these limitations, another characteristic, **Sample Entropy** (SampEn), was developed for calculation [86].

When calculating SampEn, unlike the ApEn algorithm, two conditions are added:

- the similarity of the vector to itself is not taken into account;
- when calculating the values of conditional probabilities, SampEn does not use the length of the vectors.

Based on the analysis of the above, we can conclude that SampEn:

- more than ApEn, corresponds to the theory of random numbers for a series with a known distribution density function;
- preserves the relative density, while ApEn loses this characteristic;
- adds a much smaller error to the calculated value when using vectors of small dimensionality.

```
window = 500        # window length
tstep = 1           # time step

m = 3               # embedding dimension
tau = 1             # time delay
r = 0.4             # similarity parameter

ret_type = 6        # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series
```

```
length = len(time_ser.values)  # length of a series

SampEn = []                    # an array for storing sample entropy values

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

# calculations of sample entropy
    Samp, _ = nk.entropy_sample(signal=fragm,
                                dimension=m,
                                delay=tau,
                                tolerance=r)
    SampEn.append(Samp)
```

Save the SampEn values to a text file:

```
np.savetxt(f"SampEn_name={symbol}_window={window}_step={tstep}_\
        dim={m}_tau={tau}_radius={r}_sertype={ret_type}.txt", SampEn)
```

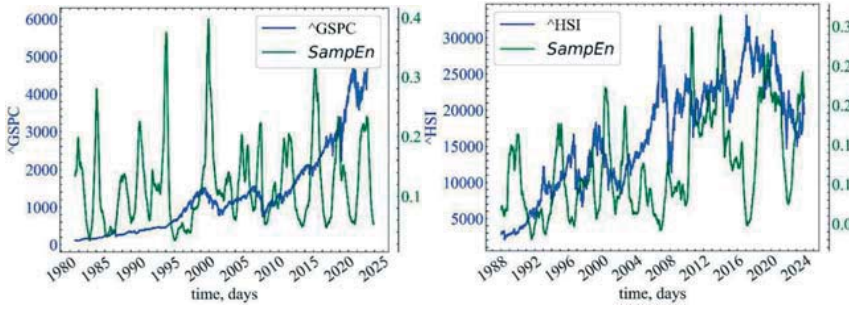
Defining the labels for the figures and the titles of the saved figures:

```
label_sampen = fr'$SampEn$'

file_name_sampen = f"SampEn_name={symbol}_window={window}_step={tstep}_\
        dim={m}_tau={tau}_radius={r}_sertype={ret_type}"
```

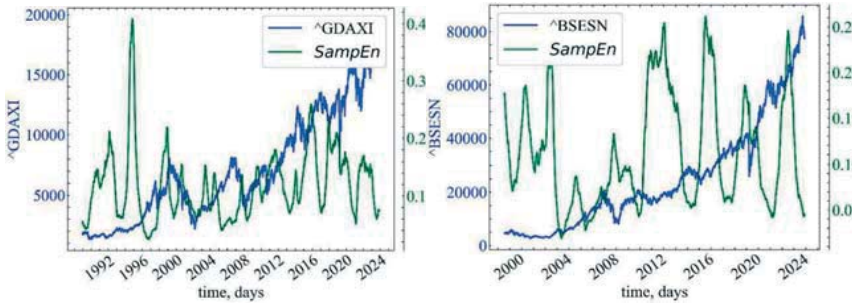
Plotting the results:

```
plot_pair(time_ser.index[window:length:tstep], time_ser.values[window:length:
tstep], SampEn, ylabel, label_sampen, xlabel, file_name_sampen, clr='darkgree
n')
```

Fig. 1.12 demonstrates the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their sample entropy.



(a) (b)



(c) (d)

Fig. 1.12: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their SampEn

Fig. 1.12 shows that SampEn decreases in the pre-crisis periods of the stock market, which indicates an increase in the correlation of the trajectories of the reconstructed phase space of the N225 index. This suggests that the market in pre-crisis periods becomes more orderly and trend-resistant.

1.5.4 Permutation entropy

Permutation entropy (PEn) is a measure from chaos theory proposed by Bandt and Pompe [33] and characterized by conceptual simplicity and computational speed. The idea of PEn is based on the usual Shannon entropy, but uses permutation patterns – ordinal relations between the values of the system. Compared to other measures of complexity, it has certain advantages, such as noise resistance and invariance to nonlinear monotonic transformations [77].

As in the previous types of entropy, we reconstruct a time series of N values with a fixed embedding dimension d_E and a time delay τ, and use the embedding matrix to form time vector sequences

$$\vec{X}(i) = \{x(i), x(i+1), \ldots, x(i+[d_E-1]\tau)\},$$

and as a result we get $N - [d_E - 1]\tau$ vectors.

Each element of $\vec{X}(i)$ is converted into numerical ranks according to their order. For example, for $d_E = 2$ and $\tau = 1$ and the time series $\vec{X}(i) = (-0.1, 0.4, 3.2, 12.0, 6.5)$, the embedded matrix will have the following pairs: $\vec{X}(1) = -0.1{,}0.4$, $\vec{X}(2) = 0.4{,}3.2$, $\vec{X}(3) = 3.2{,}12.0$, $\vec{X}(4) = 12.0{,}6.5$.

Next, we form ordinal sequences according to their numerical order. Such vectors as $\vec{X}(1), \vec{X}(2), \vec{X}(3)$ satisfy the condition $x(i) < x(i+1)$ and one vector $\vec{X}(4)$ satisfies the condition $(i) > x(i+1)$. We can consider $d_E!$ possible permutations of order d_E. In our example, there are only 2! patterns: $\pi_1 = 0{,}1, \pi_2 = 1{,}0$.

For each pattern, we determine its relative frequency:

$$p(\pi) = \#\{\vec{X}(i) \text{ has pattern } \pi\}/(N - [d_E - 1]\tau).$$

The probability of finding a vector with a pattern π_1 is $3/4$ and with a pattern π_2 is $1/4$, i.e., we form the probability distribution $P = \{p(\pi_1), \ldots, p(\pi_{d_E})\}$. Finally, this type of entropy can be calculated in the same way as the ShEn:

$$PEn(\vec{X}, d_E) = -\sum_{i=1}^{d_E} p\,(\pi_i) \ln p(\pi_i).$$

For convenience, PEn is normalized according to the following equation [165]:

$$\overline{PEn(\vec{X}, d_E)} = PEn(\vec{X}, d_E)/PEn_{max},$$

where $PEn_{max} = \ln D!$, and the normalized entropy of permutations is in the range $0 \le \overline{PEn(\vec{X}, d_E)} \le 1$.

```
window = 500          # window length
tstep = 1             # time step

m = 3                 # embedding dimension
tau = 15              # time delay

Type = 'none'         # none - classical
                      # finegrain – Finegrained PEn
                      # modified – Modified PEn
                      # weighted – Weighted PEn
```

```
                    # ampaware - Amplitude-Aware PEn
                    # edge - Edge PEn
                    # uniquant - Uniquant PEn

tpx = -1    # finegrain tpx - parameter α, positive scalar (by default: 1)
            # ampaware tpx - parameter A, value in the range [0, 1] (by defau
lt: 0.5)
            # edge tpx - sensitivity parameter r, scalar > 0 (by default: 1)
            # uniquant tpx - parameter L, integer > 1 (by default: 4)

log = np.exp(1)
norm = True       # normed entropy

ret_type = 1      # type of a series:
# 1 - initial
# 2 - detrending (difference between present and previous values)
# 3 - initial returns
# 4 - standardized returns
# 5 - absolute values (volatility)
# 6 - standardized series

length = len(time_ser.values)  # length of a series

PEn = []        # array for storing values of normalized PEn

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()

    fragm = transformation(fragm, ret_type)

# calculate PEn
    _, Pnorm, cPE = eh.PermEn(fragm,
                              m=m,
                              tau=tau,
                              Typex=Type,
                              tpx=tpx,
                              Logx=log,
                              Norm=norm)

    PEn.append(Pnorm[-1])
```

Save the permutation entropy value to a text file:

```
np.savetxt(f"PEn_name={symbol}_window={window}_step={tstep}_\
        dim={m}_tau={tau}_sertype={ret_type}_type={Type}_param={tpx}.txt", PE
n)
```

Defining the labels for the figures and the titles of the saved figures:

```
label_permen = fr'$PEn$'

file_name_perm = f"PEn_name={symbol}_window={window}_step={tstep}_\
        dim={m}_tau={tau}_sertype={ret_type}_type={Type}_param={tpx}"
```

Let’s visualize the result for PEn:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          PEn,
          ylabel,
          label_permen,
          xlabel,
          file_name_perm,
          clr='indigo')
```

Fig. 1.13 demonstrates the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their permutation entropy.

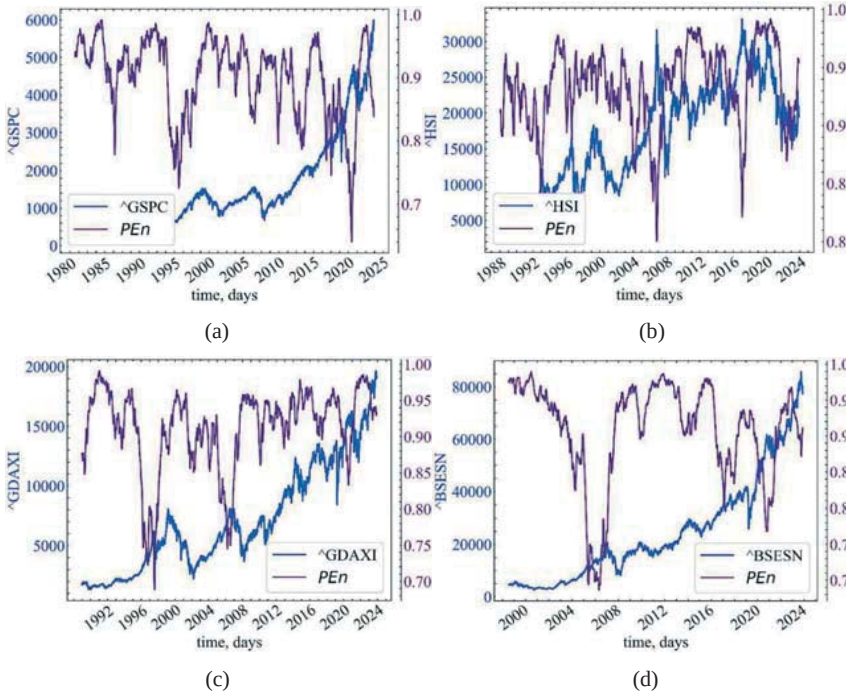


Fig. 1.13: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their PEn

Fig. 1.13 shows that PEn decreases during crisis and pre-crisis periods in the stock market. This indicates an increase in the likelihood of one particular pattern emerging for further market dynamics, and thus in the amount of information expected when analyzing fluctuations in the stock market indices.

1.5.5 Singular value decomposition entropy

The **singular value decomposition entropy** (SVDEn) [147] can be intuitively viewed as an indicator of how many eigenvectors are needed to adequately explain a data set. In other words, it measures the richness of features: the higher the SVDEn, the more orthogonal vectors are needed to adequately explain the state of the space. Similar to Fisher's information index, SVDEn is based on the decomposition of the singular value of the signal reconstructed by the time-delay method.

```
window = 500        # window length
tstep = 1           # time step

m = 3               # embedding dimension
tau = 1             # time delay

ret_type = 6        # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

length = len(time_ser.values)  # length of a series

SVDEn = [] # an array for storing values of SVD entropy

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

# calculation of the svd entropy
    svden, _ = nk.entropy_svd(signal=fragm,
                              dimension=m,
                              delay=tau)

    SVDEn.append(svden)
```

Saving SVDEn to a text file:

```
np.savetxt(f"SVDEn_name={symbol}_window={window}_step={tstep}_\
        dim={m}_tau={tau}_sertype={ret_type}.txt", SVDEn)
```

Defining the labels for the figures and the titles of the saved figures:

```
label_svden = fr'$SVDEn$'
```

```
file_name_svden = f"SVDEn_name={symbol}_window={window}_step={tstep}_\
        dim={m}_tau={tau}_sertype={ret_type}"
```

Plotting the results:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          SVDEn,
          ylabel,
          label_svden,
          xlabel,
          file_name_svden,
          clr='darkorange')
```

Fig. 1.14 shows the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their singular value decomposition entropy.

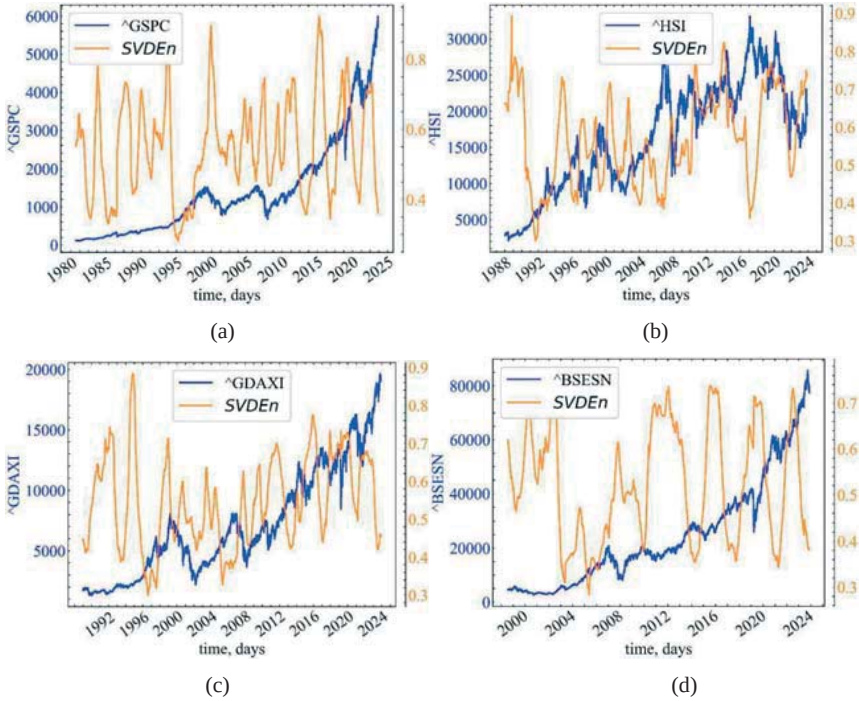


Fig. 1.14: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their SVDEn

Fig. 1.14 shows that the entropy of the singular value decomposition decreases in (pre)crisis periods, which indicates an increase in correlations in the

stock market. Since SVDEn is based on the distribution of eigenvectors, we can assume that in pre-crisis moments, market dynamics are driven by one or more eigenvectors, which are the driving component of the index under study.

1.5.6 Dispersion entropy

For a given one-dimensional signal of length N: $x = \{x_1, x_2, \dots, x_N\}$, the **Dispersion entropy** algorithm (DispEn) includes 4 main steps [111]:

1) First, the x_j $(j = 1,2, \dots N)$ are mapped to c classes labeled from 1 to c. There are a number of linear and nonlinear approaches for this. Although the linear mapping algorithm is the fastest when the maximum and/or minimum values of the time series are much larger or smaller than the mean/median value of the signal, most values of x_i are assigned to only a few classes. Thus, we first use a normal cumulative distribution function (NCDF) to map x to $y = \{y_1, y_2, \dots, y_N\}$ from 0 to 1. Next, a linear algorithm is performed to assign each y_j an integer from 1 to c. To do this, for each term of the displayed signal, we use $z_j^c = \text{round}(y_j + 0.5)$, where z_j^c represents the j-th term of the classified time series, and rounding implies either increasing or decreasing the number to the next digit. It is worth noting that this step can be performed using other linear and nonlinear mapping methods.
2) Each vector $\mathbf{z}_i^{d_E,c}$ with dimension d_E and time delay τ is of the form $\mathbf{z}_i^{d_E,c} = \{z_i^c, z_{i+\tau}^c, \dots, z_{i+(d_E-1)\tau}^c\}$, $i = 1, \dots, N - (d_E - 1)\tau$ and is projected onto the variance pattern $\pi_{v_0,v_1,\dots,v_{d_E-1}}$, where $z_i^c = v_0, z_{i+\tau}^c = v_1, \dots, z_{i+(d_E-1)\tau}^c = v_{d_E-1}$. The number of possible dispersion patterns that can be assigned to each vector $\mathbf{z}_i^{d_E,c}$ is c^m, since the signal has d_E elements and each element can be assigned an integer value from 1 to c.
3) For all potential c^m dispersion patterns, the relative frequency is calculated:

$$p\left(\pi_{v_0,v_1,\ldots,v_{d_E-1}}\right)=\frac{\#\left\{i|i\le N-\left(d_E-1\right)\tau,\boldsymbol{z}_i^{d_E,c}\text{ has pattern }\pi_{v_0,v_1,\ldots,v_{d_E-1}}\right\}}{N-\left(d_E-1\right)\tau}.$$

4) Finally, based on the Shannon entropy formula, DispEn is calculated as

$$DispEn(x,d_E,c,\tau)=-\sum_{\pi=1}^{c^{d_E}}p\left(\pi_{v_0,v_1,\ldots,v_{d_E-1}}\right)\ln p\left(\pi_{v_0,v_1,\ldots,v_{d_E-1}}\right).$$

When all possible dispersion patterns have the same probability, we get the largest value of DispEn, which is $\ln c^{d_E}$. Conversely, if only one $p\left(\pi_{v_0,v_1,\ldots,v_{d_E-1}}\right)$ differs from zero (a perfectly regular/predictable signal), we get the smallest value of DispEn.

```
window = 500       # sliding window width
tstep = 1          # sliding window time step
m = 3              # embedding dimension
tau = 1            # time delay

fluct = False # fluctuation-dispersion entropy
rho = 1       # parameter for Type="finesort", positive scalar value (by defa
ult: 1)
classes = 6   # number of symbols, which are used during transofrmation

type = 'ncdf' # type of symbolic conversion of a series:
# "ncdf" - Normalized cumulative distribution function
# "kmeans" - K-means clustering algorithm
# "linear" - Linear segmentation of the signal range
# "finesort" - Entropy of fine scattering

ret_type = 6      # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

length = len(time_ser.values)  # length of a series

DispEn = [] # an array of values for storing dispersion entropy

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

# calculations of the dispersion entropy
    Disp, _ = nk.entropy_dispersion(signal=fragm,
                                    dimension=m,
```

```
                                    delay=tau,
                                    c=classes,
                                    symbolize=type,
                                    fluctuation=fluct,
                                    rho=rho)
    DispEn.append(Disp)
```

Saving DispEn value to a text file:

```
np.savetxt(f"DispEn_symbol={symbol}_window={window}_step={tstep}_d_e={m}_tau=
{tau}_\
           series_type={ret_type}_fluct={fluct}_rho={rho}_\
           classes={classes}_type={Type}.txt", DispEn)
```

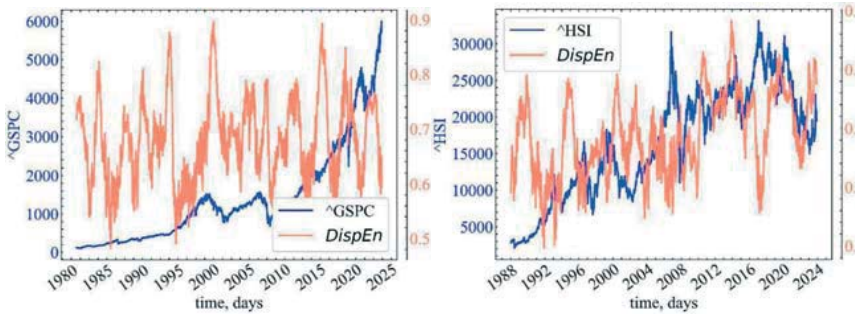
Defining the labels for the figures and the names of the saved figures:

```
label_dispen = fr'$DispEn$'

file_name_dispen = f"DispEn_symbol={symbol}_window={window}_step={tstep}_d_e=
{m}_tau={tau}_\
           series_type={ret_type}_fluct={fluct}_rho={rho}_\
           classes={classes}_type={Type}"
```

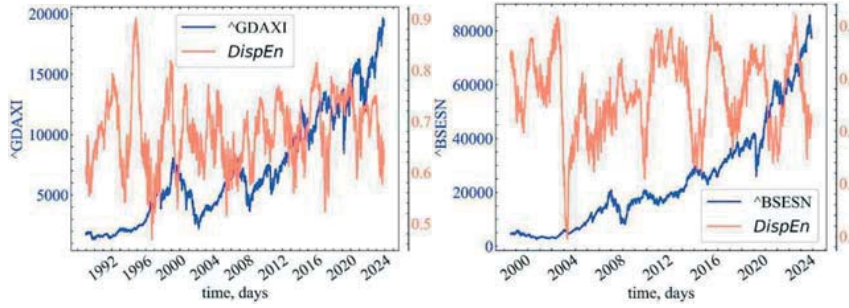
Plotting the results:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          DispEn,
          ylabel,
          label_dispen,
          xlabel,
          file_name_dispen,
          clr='coral')
```

Fig. 1.15 shows the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their dispersion entropy.



(a) (b)



(c) (d)

Fig. 1.15: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their DispEn

Fig. 1.15 shows DispEn declines in the run-up to the crashes. This is particularly noticeable for the 1970, 1990, 2010, and 2020 crashes. This suggests that the distribution of dispersion patterns becomes skewed, which is reflected in the drop in entropy. This also indicates a periodization of the market. For periods when the variance entropy is at its highest, traders' expectations also remain diverse, making the market more unpredictable.

1.5.7 Spectral entropy

Spectral entropy (SE or SpecEn) [84] considers the normalized power spectral density (PSD) of a signal in the frequency domain as a probability distribution and calculates its Shannon entropy:

$$SpEn = -\sum P(f)\log P(f).$$

A signal with a single frequency component (for example, a pure sine wave) has the lowest entropy. On the other hand, a signal with all frequency components of equal power (white noise) has the highest entropy.

```
window = 500 # window length
tstep = 1    # time step

num_bins = 30 # if an integer is passed, divides the PSD into several frequen
cy bands

method = 'fft' # method for calculating the PSD:
```

```
                # welch
                # fft
                # multitapers
                # lombscargle
                # burg

ret_type = 1      # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

length = len(time_ser.values)  # length of a series

SpEn = [] # an array of values for storing spectral entropy

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

# calculate the spectral entropy
    spec, _ = nk.entropy_spectral(signal=fragm,
                                  bins=num_bins,
                                  method=method)

    SpEn.append(spec)
```

Saving SpecEn values to a text file:

```
np.savetxt(f"SpEn_symbol={symbol}_window={window}_step={tstep}_\
           series_type={ret_type}_bins={num_bins}_psd={method}.txt", SpEn)
```

Defining the labels for the figures and the names of the saved figures:

```
label_spen = fr'$SpEn$'

file_name_spen = f"SpEn_symbol={symbol}_window={window}_step={tstep}_\
                   series_type={ret_type}_bins={num_bins}_psd={method}"
```

Plotting the results:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          SpEn,
          ylabel,
          label_spen,
          xlabel,
          file_name_spen,
          clr='deeppink')
```

In Fig. 1.16 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their spectral entropy.

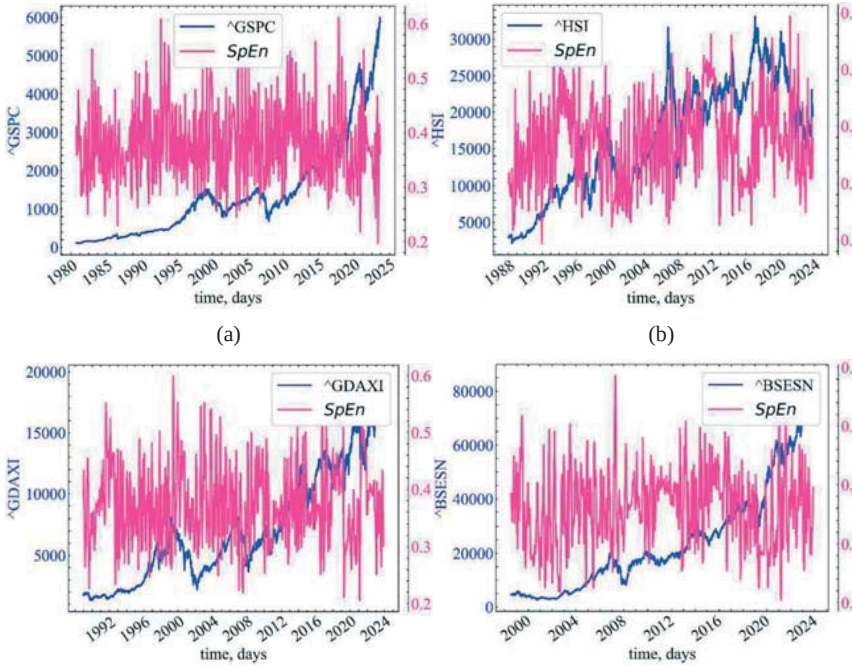


Fig. 1.16: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their SpecEn

Fig. 1.16 demonstrates that SpecEn has too chaotic a dynamics, which makes it inapplicable for monitoring and prevention of the stock market crashes.

1.6 Conclusions on informational measures of complexity

Thus, the considered information measures of complexity allow us to study certain aspects of the complexity of systems of any nature. The multiscale version of the introduced measures is especially productive. A thorough analysis of time series for systems of different nature, different levels of complexity, comparing them with test signals, studying the behavior of systems in different (not necessarily equilibrium, stationary) conditions will allow us to understand the

nature of complexity and predict the possible behavior of systems in critical conditions.

Using the example of entropy measures of complexity, this section tests the hypothesis about the relationship between complexity measures and crisis phenomena, which was put forward on the basis of complex systems theory. Using a sliding window algorithm based on a set of entropy indicators, it is shown that financial collapses are characterized by changes in complexity: in the pre-crisis period, as a rule, we can observe the ordering of the system, and in the crisis and post-crisis periods, the growth of chaos. Comparing entropy characteristics opens up the possibility of early identification and prevention of crisis phenomena in systems of different nature and complexity.

Thus, the presented indicators-precursors of crisis phenomena, theoretically, allow to circumvent the need for significant computing resources and rather controversial methods of forecasting price fluctuations and their trends.

2 Application of recurrence analysis and recurrence diagrams to the study of dynamics and topology of complex systems

2.1 Topological and structural analysis of recurrence diagrams

Studies of complex systems, both natural and artificial, have shown that they are based on nonlinear processes, a thorough study of which is necessary for understanding and modeling complex systems. In recent decades, the set of traditional (linear) research methods has been significantly expanded by nonlinear methods derived from the theory of nonlinear dynamics and chaos; many studies have been devoted to the assessment of nonlinear characteristics and properties of processes occurring in nature (scaling, fractal dimensionality). However, most methods of nonlinear analysis require either sufficiently long or stationary data series, which are quite difficult to obtain naturally. Moreover, it has been shown that these methods give satisfactory results for models of real systems that are idealized. These factors required the development of new methods of nonlinear data analysis.

The state of natural or artificial systems usually changes over time. The study of these often complex processes is an important task in many disciplines, allowing to understand and describe their essence, for example, to predict the state for some time into the future. The goal of such research is to find mathematical models that sufficiently correspond to real processes and can be used to solve the tasks at hand.

Let’s consider the idea and briefly describe the theory of recurrent analysis, give some examples, and consider its possible applications in the analysis and forecasting of complex financial and economic systems.

2.2 Phase space and its reconstruction

The state of the system is described by its state variables $x^1(t), x^2(t), \ldots, x^d(t)$, where the upper index is the variable number. The set of d state variables at time t constitutes the state vector $\vec{x}(t)$ in the d-dimensional phase space. This vector moves in time and in the direction determined by its velocity vector $\dot{\vec{x}}(t) = \partial_t \vec{x}(t) = \vec{F}(t)$.

The sequence of vectors $\vec{x}(t)$ forms a trajectory in phase space, and the velocity field $\vec{F}$ is tangent to this trajectory. The evolution of the trajectory describes the dynamics of the system and its attractor. Knowing $\vec{F}$, we can obtain information about the state of the system at time t by integrating the expression. Since the shape of the trajectory allows us to judge the nature of the process (periodic or chaotic processes have characteristic phase portraits), it is not necessary to perform integration to determine the state of the system, it is enough to build a graphical representation of the trajectory.

When studying complex systems, there is often no information on all state variables, or not all of them can be measured. As a rule, we have a single observation made at a discrete time interval Δt. Thus, measurements are written in the form of a series $u_i(t)$, where $t = i \cdot \Delta t$. The interval Δt can be constant, but this is not always possible and creates problems for the application of standard data analysis methods that require a uniform scale of observations.

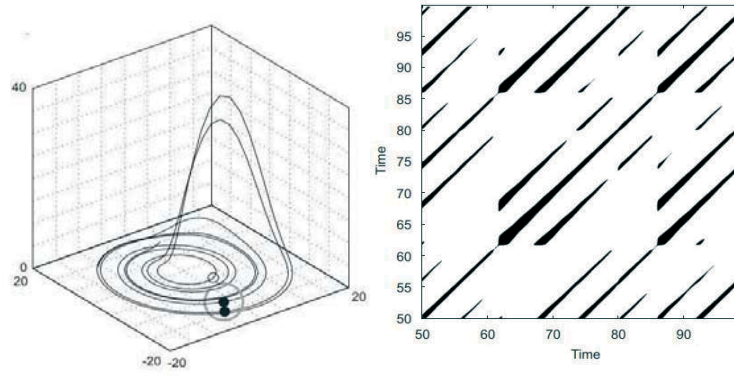
The interactions and their number in complex systems are such that even one state variable can be used to judge the dynamics of the entire system as a whole. Thus, an equivalent phase trajectory that preserves the structures of the original phase trajectory can be recovered from a single observation or time series [118] by the **Taken's theorem** using the **time-delay method** [66]:

$$\hat{\vec{x}}(t) = \left(u_i, u_{i+\tau}, \ldots, u_{i+(d_E-1)\tau}\right).$$

Here d_E is the embedding dimension, τ is the time delay (the real time delay is defined as $\tau \cdot \Delta t$). The topological structures of the recovered trajectory are

preserved if $d_E \geq 2 \cdot d + 1$, where d is the dimension of the attractor [66]. In practice, in most cases, the attractor can be recovered when $d_E \leq 2d$. The delay is usually chosen a priori.

There are several approaches to choosing the minimum sufficient dimension d_E, except for the analytical one. Methods based on the concept of false nearest neighbors (FNN) have shown high efficiency. Its essence lies in the fact that when the dimensionality of the embedding is reduced, the number of false points falling in the neighborhood of any point of the phase space increases. This leads to a simple method – determining the number of FNNs as a function of the dimensionality. There are other methods based on this concept, for example, determining the distance relations between the same neighboring points at different d_E. The dimensionality of an attractor can also be determined using cross-correlation sums.



(a) (b)

Fig. 2.1: A trajectory segment in the phase space of the Rösler system (a) and the corresponding recurrence plot (b). The phase space vector at point j, which falls in the neighborhood (gray circle in (a)) of the given phase space vector at point i, is a considered recurrence point. It is indicated by a black dot on the recurrence diagram at position (i, j). The phase space vector outside the neighborhood (the empty circle in (a)) is denoted by a white dot on the recurrence diagram

2.3 Recurrence analysis

Processes in nature are characterized by pronounced recurrent behavior, such as periodicity or irregular cyclicity. Moreover, the recurrence (repeatability) of states in the sense of following a subsequent trajectory close enough to the previous one is a fundamental property of dissipative dynamical systems. This property was noted back in the 80s of the XIX century by the French mathematician Poincaré and subsequently formulated in the form of the "recurrence theorem" published in 1890:

If the system reduces its dynamics to a limited subset of the phase space, then it almost certainly, i.e. with a probability practically equal to 1, returns to any initially state as close as possible.

The essence of this fundamental property is that even a small perturbation in a complex dynamic system can lead the system to an exponential deviation from its state, and after a while the system tends to return to a state close to the previous one, and goes through similar stages of evolution.

This can be verified by graphically depicting the system's trajectory in phase space. However, the possibilities of such an analysis are severely limited. As a rule, the dimensionality of the phase space of a complex dynamical system is greater than three, which makes it practically inconvenient to consider it directly; the only possibility is projection into two- and three-dimensional spaces, which often does not give a correct picture of the phase portrait.

In 1987, Eckmann and co-authors proposed a way to map an m-dimensional phase trajectory of states of a system $\vec{x}(t)$ of length N onto a two-dimensional square binary matrix of size $N \times N$ [93], where 1 (black dot) corresponds to the repetition of a state at some time i at some other time j, and both coordinate axes are time axes. Such a representation was called a **recurrence plot** (RP) or recurrence diagram because it captures information about the recurrent behavior of the system.

Mathematically, the above is described as

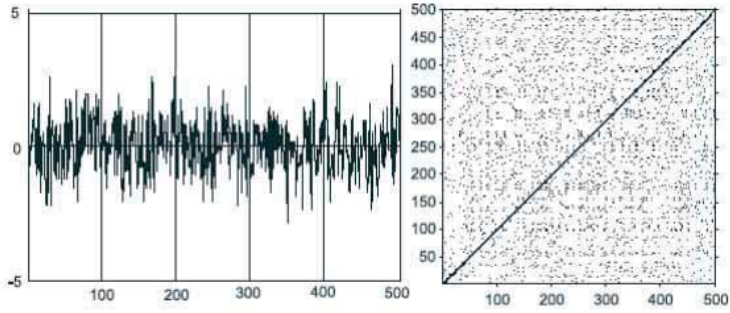
$$R_{i,j}^{d_E,\varepsilon_i} = \Theta\left(\varepsilon_i - \left\|\vec{x}_i - \vec{x}_j\right\|\right), \quad \vec{x} \in \Re^{d_E}, \quad i,j = 1,\ldots,N,$$

where N is the number of states $\vec{x}_i$, ε_i is the size of the neighborhood of point $\vec{x}$ at time i, $\|\cdot\|$ is the norm, and $\Theta(\cdot)$ is the Heaviside function.

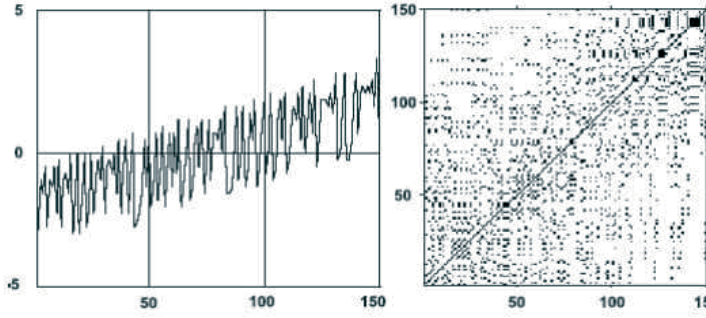
It is impractical and, as a rule, impossible to find full recurrence in the value of $\vec{x}_i \equiv \vec{x}_j$ (the state of a dynamic, and especially a chaotic system, does not repeat itself completely equivalent to the initial state, but approaches it as close as possible). Thus, recurrence is defined as the sufficient closeness of state $\vec{x}_j$ to state $\vec{x}_i$. In other words, recurrent states are those $\vec{x}_j$ states that fall into an d_E-dimensional neighborhood with radius ε_i and centered at $\vec{x}_i$. These points $\vec{x}_j$ are called **recurrence points** [163, 164].

Since $R_{i,i} = 1$, $i = 1,\ldots,N$ by definition, a recurrence diagram always contains a black diagonal line – the line of identity (LOI) at an angle $\pi/4$ to the coordinate axes. An arbitrarily taken recurrent point does not carry any useful information about the states at times i and j. Only the entire set of recurrent points allows you to restore the properties of the system.

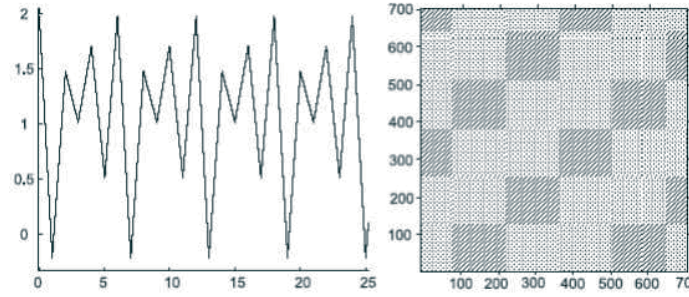
The appearance of the recurrence diagram allows us to judge the nature of the processes taking place in the system, the presence and influence of noise, states of repetition and freezing (laminarity), and abrupt changes in the system's state (extreme events).



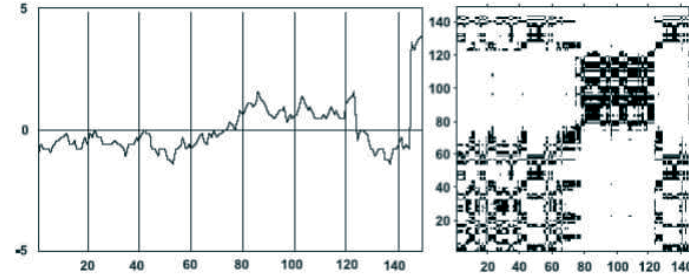
(a)


5
0
-5
50
100
150
150
100
50
50
100
150

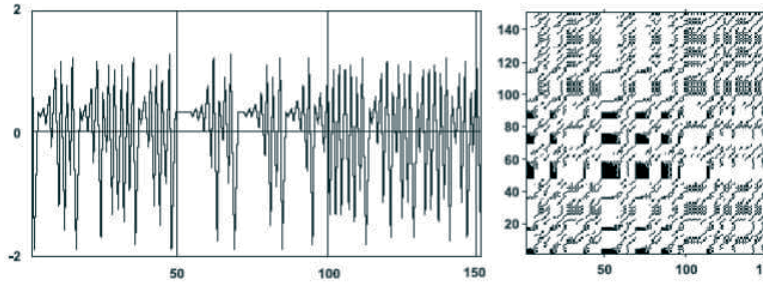
(b)


2
1.5
1
0.5
0
0
5
10
15
20
25
700
600
500
400
300
200
100
100 200 300 400 500 600 700

(c)


5
0
-5
20
40
60
80
100
120
140
140
120
100
80
60
40
20
20 40 60 80 100 120 140

(d)



(e)

Fig. 2.2: Dynamic time series characterizing homogeneity (a), drift (b), oscillation (c), contrasting topology (d), laminarity (e) and their recurrence diagrams

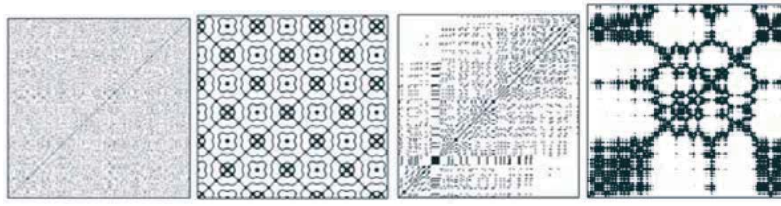
2.4 Analysis of the diagrams

Obviously, processes of different behavior will produce recurrence diagrams with different patterns. Thus, visual evaluation of the diagrams can give an idea of the evolution of the trajectory under study. There are two main classes of image structure: **topology**, represented by large-scale structures, and **texture**, formed by small-scale structures.

Topology gives a general idea of the nature of the process. There are four main classes:

- **homogeneous** recurrence diagrams are typical for stationary and autonomous systems in which the relaxation time is small compared to the length of the series;
- **periodic** structures that repeat (diagonal lines, staggered patterns) correspond to various oscillating systems with periodicity in dynamics;
- **drift** corresponds to systems with slowly changing parameters, and this makes the upper left and lower right corners of the recurrence diagram white;
- **adrupt changes** in the system dynamics, as well as extreme situations, cause the appearance of white areas or bands.

RPs make it easier to identify extreme and rare events.



(a) (b) (c) (d)

Fig. 2.3: Characteristic topologies of recurrent diagrams: (a) – homogeneous (normally distributed noise); (b) – periodic (Van der Pol generator); (c) – drift (Ikeda mapping with a superimposed linearly growing sequence); (d) – contrasting regions or bands (generalized Brownian motion) [96]

A detailed examination of recurrence diagrams reveals small-scale structures – a texture made up of simple points, diagonal, horizontal, and vertical lines. Combinations of vertical and horizontal lines form rectangular clusters of points:

- **single**, separately located recurrent points appear when the corresponding states are rare, or unstable in time, or caused by strong fluctuations. In this case, they are not signs of randomness or noise;
- **diagonal lines** $R_{i+k,j+k} = 1$ (for $k = 1 \ldots l$ where l is the length of the diagonal line) appear when a segment of the trajectory in phase space runs parallel to another segment, i.e., the trajectory repeats itself, returning to the same region of phase space at different times. The length of such lines is determined by the time during which the trajectory segments remain parallel; the direction (angle of inclination) of the lines characterizes the internal time of the subprocesses corresponding to these trajectory segments. The passage of lines parallel to the identity line (at an angle of $\pi/4$ to the coordinate axes) indicates the same direction of the trajectory segments, perpendicularly – the opposite ("reflected" segments), which may also be a sign of phase space reconstruction with an inappropriate embedding dimension. Irregular appearance of diagonal lines is a sign of a chaotic process;

- **vertical (horizontal) lines** $R_{i,j+k} = 1$ (with $k = 1 \ldots v$, where v is the length of a vertical or horizontal line) highlight the time intervals in which the state of the system does not change or changes insignificantly (the system is "frozen" at this time), which is a sign of "laminar" states.

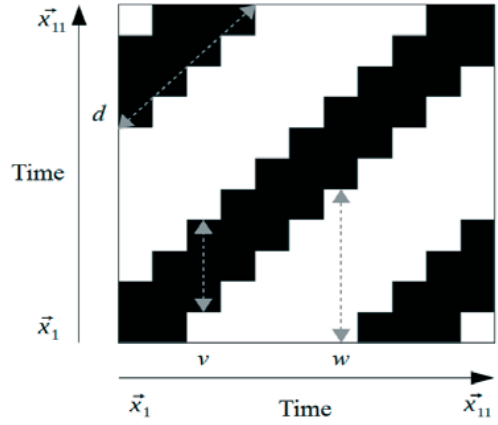


Fig. 2.4: Basic concepts of recurrence analysis. The displayed recurrence diagram is based on a time series that has been reconstructed to 11 reconstructed vectors, from $\vec{X}(0)$ to $\vec{X}(10)$. A diagonal line of length $d = 4$, a vertical line of length $v = 3$, and a white vertical line of length $w = 5$ were identified [158]

2.5 Quantitative analysis of recurrence diagrams

For a qualitative description of a system, a graphical representation of the system is the best. However, the main disadvantage of graphical representation is that it forces users to subjectively intuitively interpret the patterns and structures presented in the recurrence diagram.

In addition, as the size of the data increases, it becomes problematic to analyze all N^2 values. As a result, you have to work with separate sections of the original data. Analyzing in this way can create new defects that distort the objectivity of the observed patterns and lead to incorrect interpretations. To overcome this limitation and to disseminate an objective assessment among researchers, definitions and procedures for quantifying the complexity of recurrent

charts were introduced by Webber and Zbilut [39, 85] in the early 1990s and later extended by Marwan et al [119].

Small-scale clusters can be a combination of isolated points (random recurrences). Such an evolution at different time periods or in reverse time order will represent diagonal lines (deterministic structures), as well as vertical/horizontal lines to indicate laminar states (discontinuities) or states representing singularities. For quantitative description of the system of systems, such small-scale clusters serve as the basis for **recurrence quantification analysis** (RQA) [18].

2.5.1 RQA within the sliding window procedure

For further work, we create a window procedure in which we again define the type of series and a few more parameters. Then we initialize the arrays for each recurrence measure:

```
ret_type = 6            # type of a series
window = 500            # sliding window length
tstep = 1               # sliding window step
length = len(time_ser)  # length of a series

m = 1                   # embedding dimension
tau = 1                 # time delay
eps = 0.3               # radius

# Initialize arrays to store windowed values of recurrence measures

RR = []                 # recurrence rate
DET = []                # determinism
DIV = []                # divergence
AVG_DIAG_LINE = []      # average diagonal line length
ENT_DIAG = []           # entropy of diagonal lines
LAM = []                # laminarity
TT = []                 # trapping time
ENT_VERT = []           # entropy of vertical lines
ENT_WHITE_VERT = []     # entropy of white vertical lines
AVG_WVERT_LINE = []     # average white vertical line length
VERT_DIV = []           # divergence of vertical lines
RATIO_DET_REC = []      # ratio of determinism to recurrence rate
RATIO_LAM_DET = []      # ratio of laminarity to determinism
WHITE_VERT_DIV = []     # divergence of white vertical lines
DIAG_RR = []            # diagonal recurrence rate
```

For further calculations, we will use the `complexity_rqa()` method of the `neuralkit2` library. Its syntax is the following:

```
complexity_rqa(signal, dimension=3, delay=1, tolerance='sd', min_linelength=2, method='python', show=False)
```

Parameters:

- **signal** (*Union[list, np.array, pd.Series]*) – the signal (i.e., a time series) in the form of a vector of values;
- **delay** (*int*) – time delay (often denoted τ, sometimes referred to as lag) in samples;
- **dimension** (*int*) – embedding dimension (d_E, sometimes referred to as d or $order$ or m);
- **tolerance** (*float*) – tolerance (often denoted as r), distance to consider two data points as similar. If "sd" (default), will be set to $0.2 * \text{SDsignal}$;
- **min_linelength** (*int*) – minimum length of diagonal and vertical lines. Default to 2;
- **method** (*str*) – can be "pyrqa" to use the *PyRQA* package (requires to install it first);
- **show** (*bool*) – visualize recurrence matrix.

Returns:

- **rqa** (*DataFrame*) – the RQA results;
- **info** (*dict*) – a dictionary containing additional information regarding the parameters used to compute RQA.

Now we can start the sliding window procedure:

```
for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

    resultRQA, _ = nk.complexity_rqa(fragm,
                                     delay=tau,
                                     dimension=m,
                                     tolerance=eps)
```

```
# Calculating the ratio of laminarity to determinism
    resultRQA['LamiDet'] = resultRQA['Laminarity']/resultRQA['Determinism']

# Calculating the divergence of black vertical lines
    resultRQA['VDiv'] = 1./resultRQA['VMax']

# Calculating the divergence of white vertical lines
    resultRQA['WVDiv'] = 1./resultRQA['WMax']

    RR.append(resultRQA['RecurrenceRate'])
    DET.append(resultRQA['Determinism'])
    DIV.append(resultRQA['Divergence'])
    AVG_DIAG_LINE.append(resultRQA['L'])
    ENT_DIAG.append(resultRQA['LEn'])
    LAM.append(resultRQA['Laminarity'])
    TT.append(resultRQA['TrappingTime'])
    ENT_VERT.append(resultRQA['VEn'])
    ENT_WHITE_VERT.append(resultRQA['WEn'])
    AVG_WVERT_LINE.append(resultRQA['W'])
    VERT_DIV.append(resultRQA['VDiv'])
    WHITE_VERT_DIV.append(resultRQA['WVDiv'])
    RATIO_DET_REC.append(resultRQA['DeteRec'])
    RATIO_LAM_DET.append(resultRQA['LamiDet'])
    DIAG_RR.append(resultRQA['DiagRec'])
```

Saving the results to text files:

```
name =f"RQA_classic_name={symbol}_window={window}_ \
    step={tstep}_rettype={ret_type}_m={m}_ \
    tau={tau}_eps={eps}.txt"

np.savetxt("RR"+ name, RR)
np.savetxt("DIAG_RR"+ name, DIAG_RR)
np.savetxt("DET"+ name, DET)
np.savetxt("DIV"+ name, DIV)
np.savetxt("VERT_DIV"+ name, VERT_DIV)
np.savetxt("WHITE_VERT_DIV"+ name, WHITE_VERT_DIV)
np.savetxt("LAM"+ name, LAM)
np.savetxt("TT"+ name, TT)
np.savetxt("AVG_DIAG_LINE"+ name, AVG_DIAG_LINE)
np.savetxt("AVG_WRITE_VERT_LINE"+ name, AVG_WVERT_LINE)
np.savetxt("ENT_DIAG"+ name, ENT_DIAG)
np.savetxt("ENT_VERT"+ name, ENT_VERT)
np.savetxt("ENT_WHITE_VERT"+ name, ENT_WHITE_VERT)
np.savetxt("RATIO_DET_REC"+ name, RATIO_DET_REC)
np.savetxt("RATIO_LAM_DET"+ name, RATIO_LAM_DET)
```

2.5.2 Recurrence measures of complexity

Now let’s plot and interpret the results. To visualize the figures, let’s define the following function:

```
def plot_recurrence_measure(measure, label, clr="magenta"):

    fig, ax = plt.subplots()

    ax2 = ax.twinx()

    ax2.spines.right.set_position(("axes", 1.03))

    p1, = ax.plot(time_ser.index[window:length:tstep],
                  time_ser.values[window:length:tstep],
"b-", label=fr"{ylabel}")
    p2, = ax2.plot(time_ser.index[window:length:tstep],
                   measure,
                   color=clr,
                   label=fr'${label}$')

    ax.set_xlabel(xlabel)
    ax.set_ylabel(f"{ylabel}")

    ax.yaxis.label.set_color(p1.get_color())
    ax2.yaxis.label.set_color(p2.get_color())

    tkw=dict(size=2, width=1.5)

    ax.tick_params(axis='x', rotation=45, **tkw)
    ax.tick_params(axis='y', colors=p1.get_color(), **tkw)
    ax2.tick_params(axis='y', colors=p2.get_color(), **tkw)
    ax2.legend(handles=[p1, p2])

    plt.savefig(label +
f" RQA_classic_name={symbol}_window={window}_step={tstep}_ \
        rettype={ret_type}_m={m}_tau={tau}_eps={eps}.jpg")

    plt.show();
```

2.5.2.1 Recurrence rate

The simplest indicator is the **recurrence rate** (RR), which determines the density of recurrent points on the plot, ignoring the LOI:

$$RR = \frac{1}{N^2} \sum_{i,j=1}^{N} R\,(i,j),$$

where N is the number of points on the phase space trajectory.

The recurrence rate corresponds to the probability that a certain state will be repeated.

Fig. 2.5 demonstrates the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their *RR* measure.

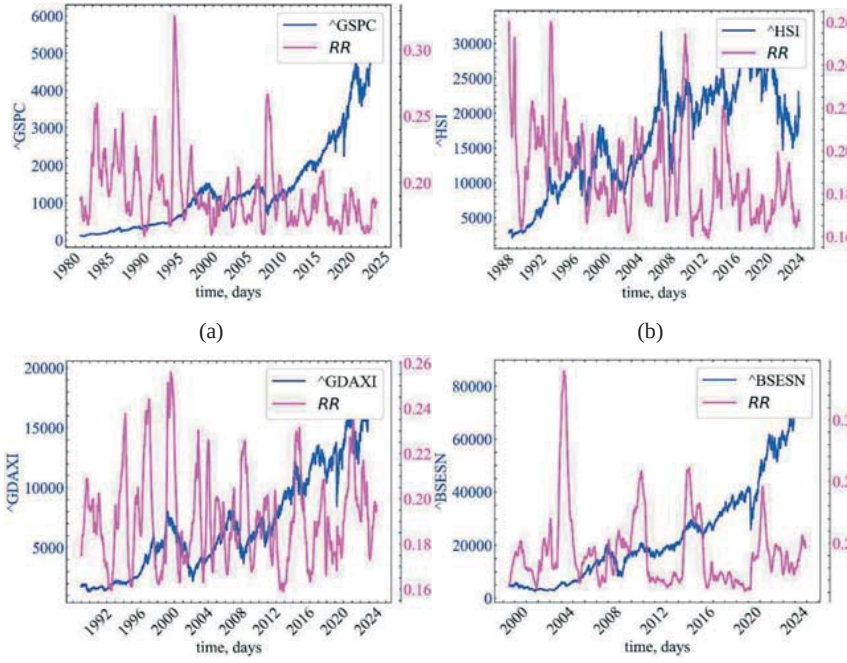


Fig. 2.5: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their *RR* measure

As we can see from Fig. 2.5, the degree of recurrence increases during crash events, indicating an increase in the degree of self-organization and coherence of trading activity among traders in this market.

2.5.2.2 Diagonal recurrence rate

This approach is based on diagonal recurrence profiles of the time series [22]. The diagonal recurrence profile determines the number of recurrence points at different lags similar to the autocorrelation function. To obtain the diagonal recurrence profile, the proportion of recurrence points on the diagonals located in

the lower right or lower left corner of the chart is simply counted and plotted as a function of distance from the main diagonal, i.e. lag.

In other words, the **diagonal recurrence rate** captures the amount of autocorrelation at different lags.

Fig. 2.6 demonstrates the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their diagonal *RR* index.

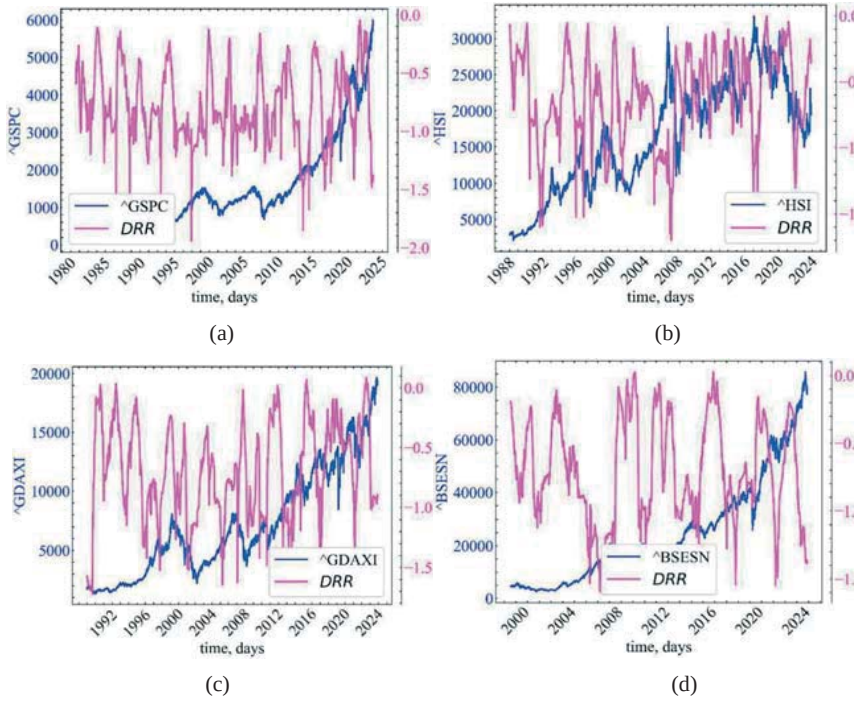


Fig. 2.6: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their diagonal *RR* measure

Fig. 2.6 shows that the diagonal recurrence rate increases in the pre-crisis and crisis periods, indicating an increase in the magnitude of autocorrelation, which in turn demonstrates an increase in the degree of self-organization.

2.5.2.3 Measure of determinism

The next indicator represents the proportion of recurrent trajectories that form diagonal lines of minimum length $\ell_{\min}$. This measure is called **determinism** and is related to the predictability of a dynamic system:

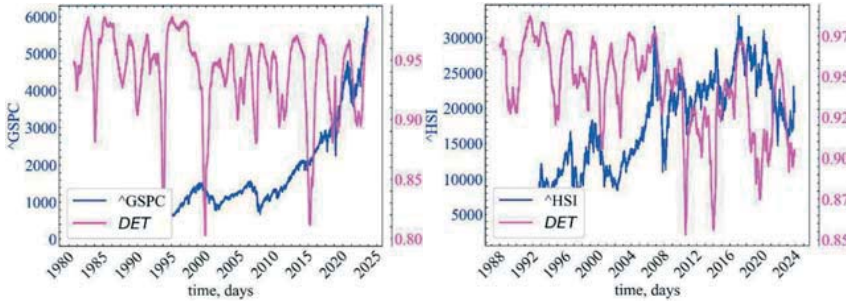
$$DET = \sum_{\ell=\ell_{min}}^{N} \ell \cdot P(\ell) \Big/ \sum_{\ell=1}^{N} \ell \cdot P(\ell),$$

where $P(\ell)$ is the frequency distribution of the diagonal lines with lengths ℓ.

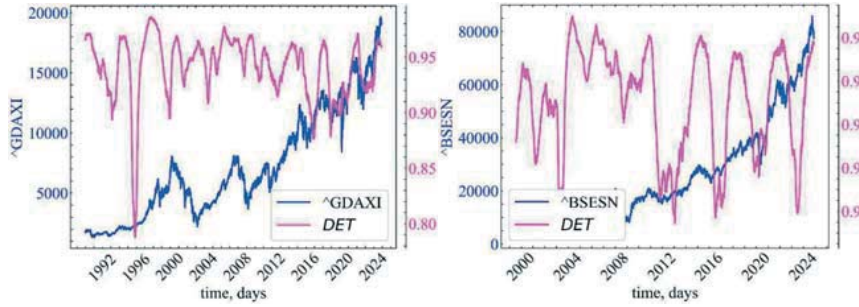
Deterministic systems are characterized by a significant variation of diagonal lines of different lengths. Periodic signals are characterized by long diagonal lines, while for chaotic signals the diagonal lines will be short. For stochastic systems, there will be no diagonal lines at all, except for random patterns that will form very short diagonal lines.

White noise, for example, would have a recurrence pattern with almost isolated recurrence points and a very small percentage of diagonal lines, while a deterministic process would show a very small number of single recurrences but a high density of long diagonal lines.

Fig. 2.7 demonstrates the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their determinism measure.



(a) (b)



(c) (d)

Fig. 2.7: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their DET

As we can see from Fig. 2.7, in the pre-crisis periods, DET begins to increase, which also indicates an increase in the degree of predictability (orderliness) of system fluctuations.

2.5.2.4 Laminarity

The indicator characterizing the number of recurrent states that form vertical lines is called **laminarity** and is related to the number of laminar phases in the system:

$$LAM = \sum_{v=v_{min}}^{N} v \cdot P(v) \Big/ \sum_{v=1}^{N} v \cdot P(v),$$

and $P(v)$ is the frequency distribution of the lengths v of vertical lines that have a length of at least $v_{\min}$. Laminarity characterizes the probability of a system to remain in an unchanged state. As the number of isolated recurrent points in the system increases, the degree of laminarity will decrease.

Fig. 2.8 demonstrates the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their laminarity index.

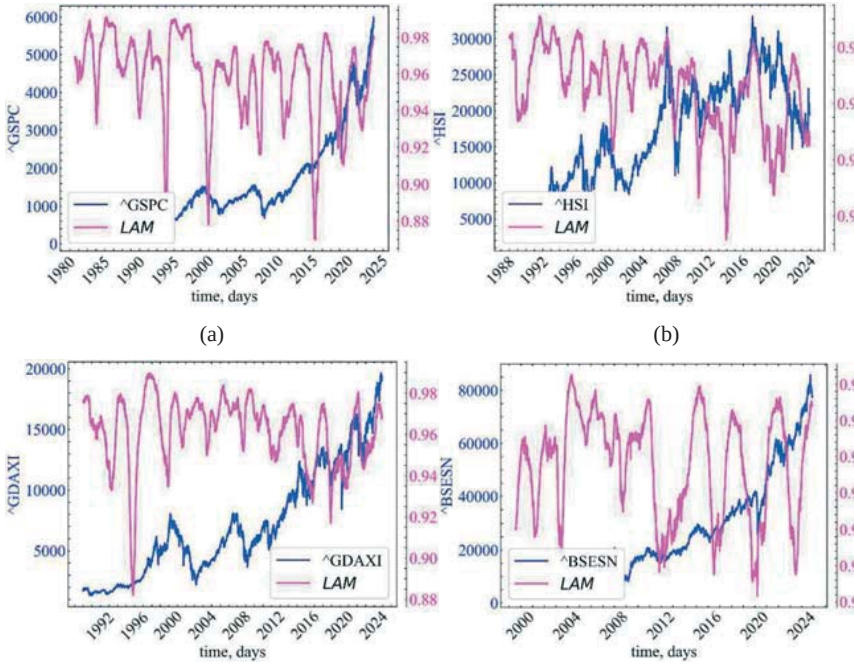


Fig. 2.8: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their *LAM*

It can be seen that the degree of laminarity increases during crisis states. Both the density of diagonal points and the overall number of recurrent trajectories in the phase space increases. Crises are characterized by trend stability, persistence and determinism of their behavior.

2.5.2.5 Average diagonal line length

You can also measure the **average diagonal lines length**. The average diagonal lines length is defined as

$$L = \sum_{\ell=\ell_{min}}^{N} \ell \cdot P(\ell) \Big/ \sum_{\ell=\ell_{min}}^{N} P(\ell).$$

In general, this indicator characterizes the average period of time when two phase space trajectories are sufficiently close to each other. The average length of the diagonal lines determines the average time at which the system remains predictable.

In Fig. 2.9 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their average length of diagonal lines

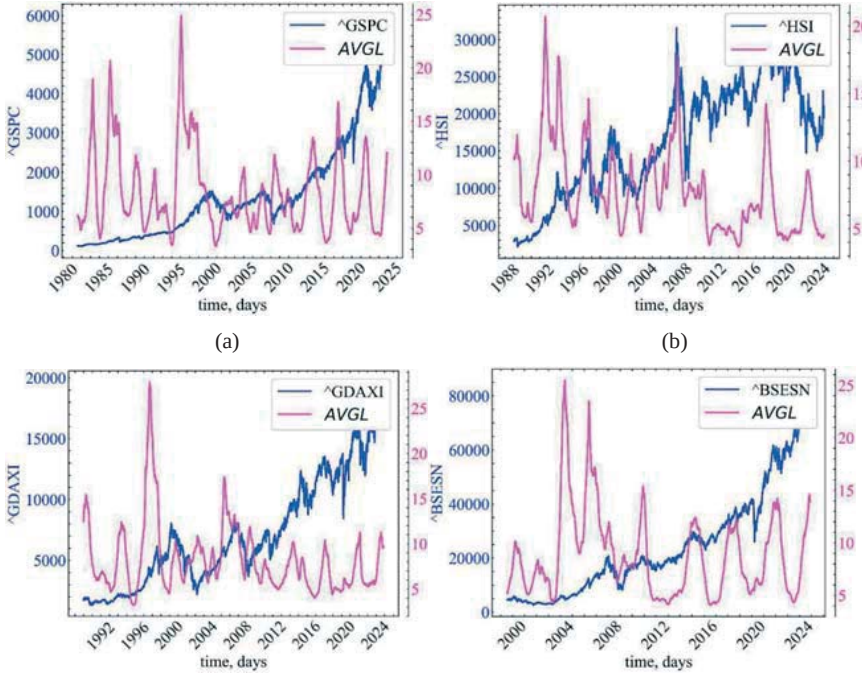


Fig. 2.9: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their average length of diagonal lines

As before, we can see that the average time the studied stock indices stays in the deterministic state increases before crisis periods, which indicates an increase in the degree of collectivization of traders in the market.

2.5.2.6 Trapping/delay time

The average length of the vertical line is related to the predictability time of the dynamic system and the **trapping time**:

$$TT = \sum_{v=v_{min}}^{N} v \cdot P(v) \Big/ \sum_{v=v_{min}}^{N} P(v).$$

The average length of the vertical lines determines the average time the system stays in the laminar state. That is, it corresponds to the average period of time when the system "freezes" in a certain state. Obviously, the growth of this value characterizes the longer and longer delay time of the system under study in a certain state.

In Fig. 2.10 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their trapping time indicator.

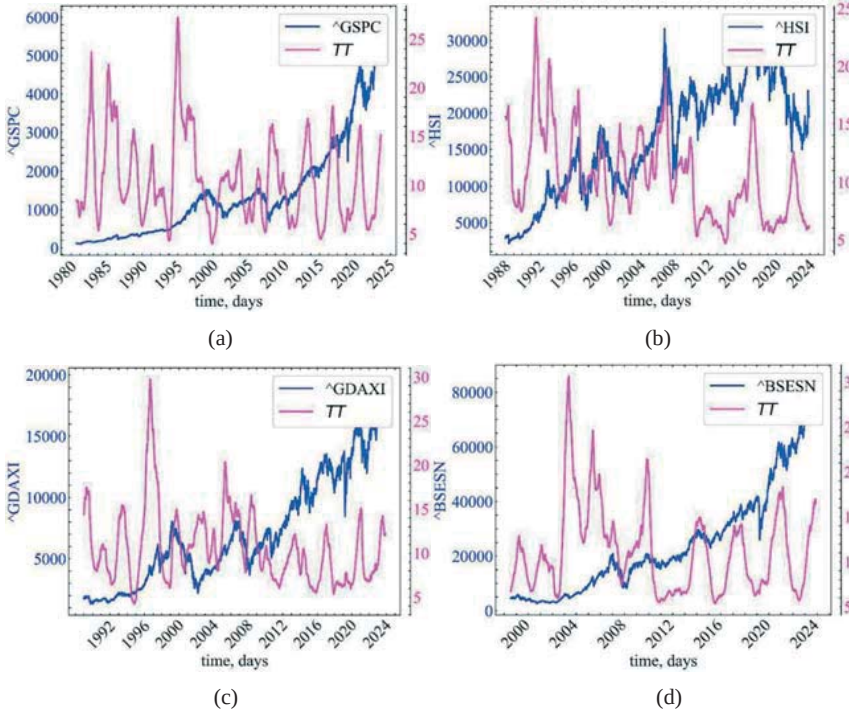


Fig. 2.10: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their *TT*

Fig. 2.10 shows that TT increases in (pre-)crisis states, indicating that the system is trying to stay in a state of crisis for some time.

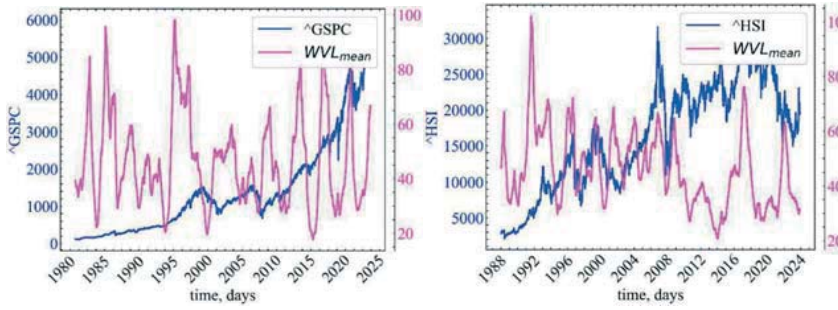
2.5.2.7 Average white vertical lines length

The average white vertical lines length can be defined as

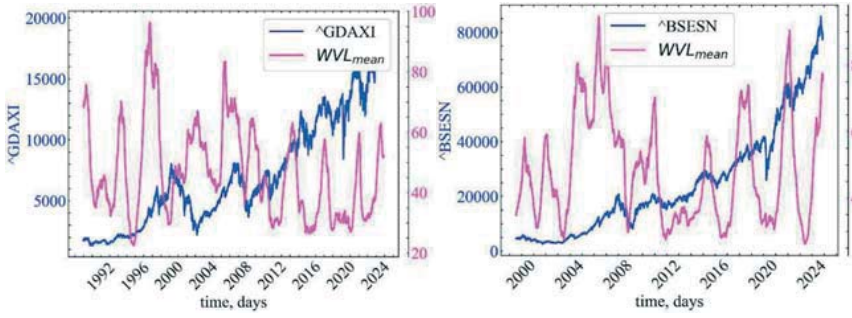
$$WVL_{mean} = \sum_{w=w_{min}}^{N} w \cdot P(w) \Big/ \sum_{w=w_{min}}^{N} P(w),$$

where $P(w)$ is the frequency distribution of white vertical lines of length w, and w_{min} corresponds to the shortest length of white vertical lines (the shortest period of return to the recurrence state). The presented measure can be characterized as the average system **unpredictability** horizon.

In Fig. 2.11 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their average length of white vertical lines.



(a) (b)



(c) (d)

Fig. 2.11: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their average length of white vertical lines

2.5.2.8 Diagonal lines entropy

Given the appropriate diagonal segments, the amount of information needed to describe the entire distribution of this type of line can be calculated. The probability $p(\ell)$ that a diagonal line has length ℓ can be estimated from the frequency distribution $P(\ell)$ with $p(\ell) = P(\ell)/\sum_{\ell=\ell_{\min}}^{N} P(\ell)$. The Shannon entropy of the probability of occurrence of such diagonal lines (diagonal lines entropy) can be defined as follows:

$$DLEn = -\sum_{\ell=\ell_{min}}^{N} p(\ell) \ln p(\ell).$$

This indicator reflects the complexity of the structure under study.

For uncorrelated noise or oscillations, we would get small value of the entropy, which would indicate an asymmetric distribution of diagonal lines: there would be a small fraction of diagonal lines of a particular length, which would characterize the recurrence of the system under study. An increase in this entropy would indicate an increase in the symmetry of the distribution of diagonal line lengths.

In Fig. 2.12 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their entropy of diagonal lines.

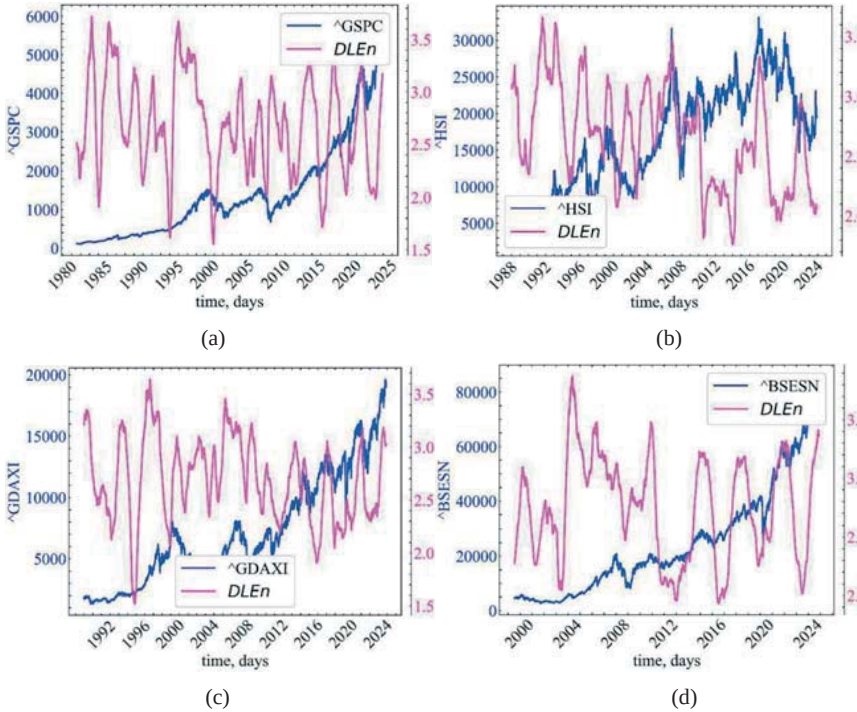


Fig. 2.12: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their entropy of diagonal lines

Fig. 2.12 shows that the entropy of the diagonal lines increases during crises, indicating the growing influence of deterministic processes with varying degrees of predictability.

2.5.2.9 Vertical lines entropy

We can define the **Shannon entropy for the distribution of vertical structures** (vertical lines entropy) of a recurrence plot. The probability $p(v)$ that a vertical line has length v can be estimated from the frequency distribution $P(v)$ with $p(v) = P(v)/\sum_{v=v_{\min}}^{N} P(v)$. The Shannon entropy of this probability is defined as

$$VLEn = -\sum_{v=v_{min}}^{N} p(v) \ln p(v).$$

This measure, similar to the previous entropy, is also a measure of system complexity.

For a sinusoidal process, we would expect a small value of this entropy, since it is a simple periodic process. For a complex process with memory, we expect a high value of this type of recurrent entropy. This would mean that the laminarity of the process is characterized by different periods of long-term memory of the system.

In Fig. 2.13 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their entropy of vertical lines.

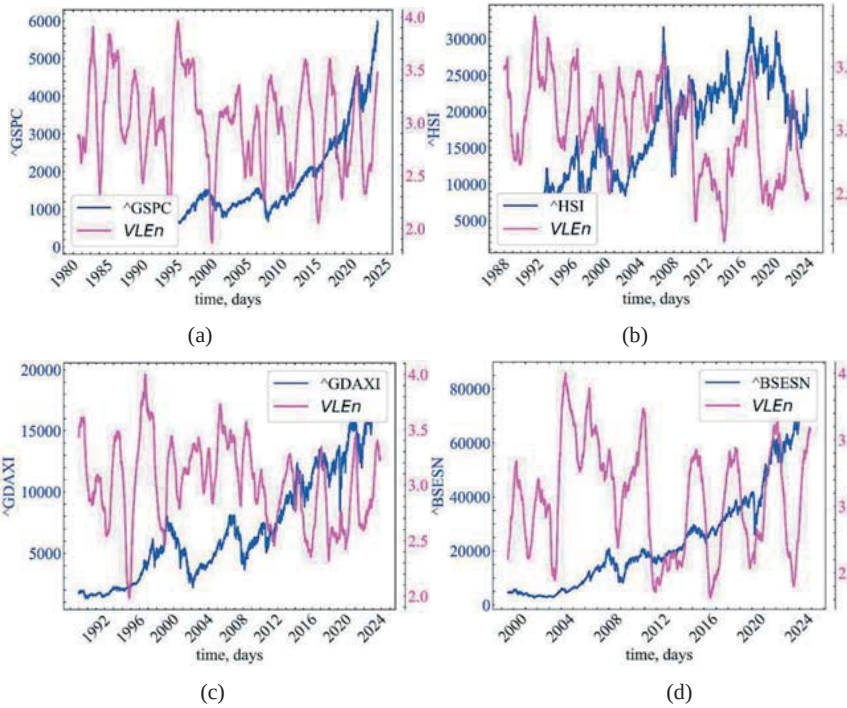


Fig. 2.13: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their entropy of vertical lines

Fig. 2.13 shows that the entropy of vertical lines begins to increase during the crash, indicating an increase in the degree of laminarity, i.e., an increase in the uniformity of the distribution of vertical lines of different lengths.

2.5.2.10 Divergence

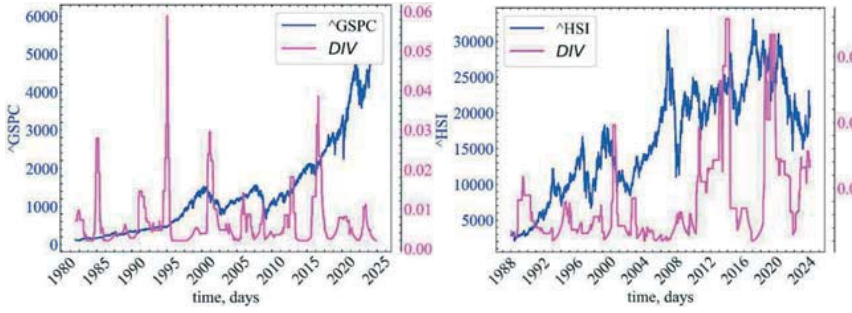
The $L_{\max}$ indicator can provide us with information about the maximum degree of predictability of the period under study. The inverse value of the **maximum length of the diagonal lines** $L_{\max}$ or **divergence** can indicate the speed and duration of the divergence of the studied trajectories. This indicator can be defined as

$$DIV = 1/L_{max}.$$

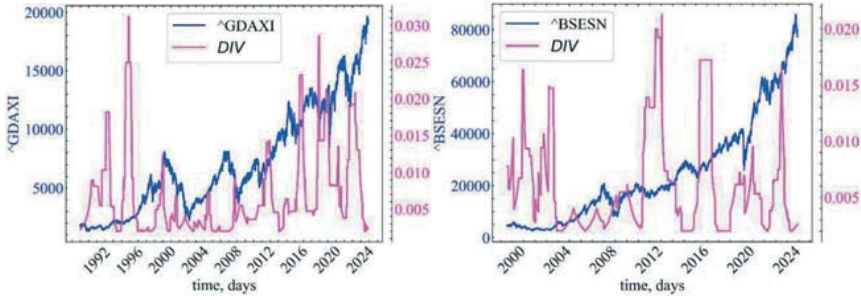
This measure is similar to the largest Lyapunov exponent [93]. However, the relationship between this measure and the positive maximum Lyapunov exponent is much more complicated (to calculate the Lyapunov exponent from RP, the entire frequency distribution of the diagonal lines must be taken into account).

The higher the divergence value, the faster the phase space trajectories diverge. And vice versa, the lower the divergence value, the closer the trajectories under study are to each other.

In Fig. 2.14 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their divergence indicator.



(a) (b)



(c) (d)

Fig. 2.14: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their DIV indicator

Fig. 2.14 shows that the divergence of the diagonal lines begins to decline in the crisis and pre-crisis periods, which also indicates an increase in the degree of orderliness of the system's dynamics during these periods.

2.5.2.11 Divergence of vertical lines

The inverse value of the maximum vertical line length V_{max} or vertical line divergence can be defined as

$$VDIV = 1/V_{max}.$$

The maximum length of the vertical lines provided us with information about the maximum degree of system invariance. Vertical divergence allows us to characterize the rate of onset or decay of laminarity in the system. The higher the value of $VDIV$, the faster the system leaves the laminar state and vice versa.

In Fig. 2.15 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and the divergence index of their vertical lines.

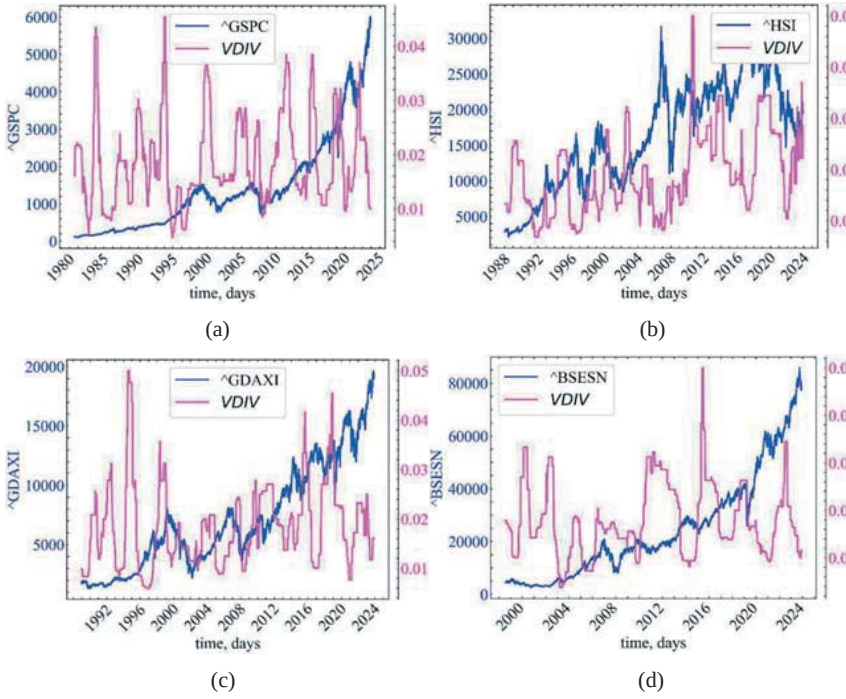


Fig. 2.15: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their divergence of vertical lines

Fig. 2.15 shows that periods of crises are characterized by a decline in vertical divergence, i.e., an increase in the number of vertical structures that characterize an even greater degree of laminarity of states.

2.5.2.12 White vertical lines divergence

The inverse value of the **maximum length of the white vertical lines** (WVL_{max}) can be described as **white vertical lines divergence**. It can be defined as follows:

$$WVDIV = 1/WVL_{max}.$$

The increase of this indicator should indicate an increase in the degree of recurrence of the system, and its decline should demonstrate an increase in unpredictability.

In Fig. 2.16 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their divergence of white vertical lines.

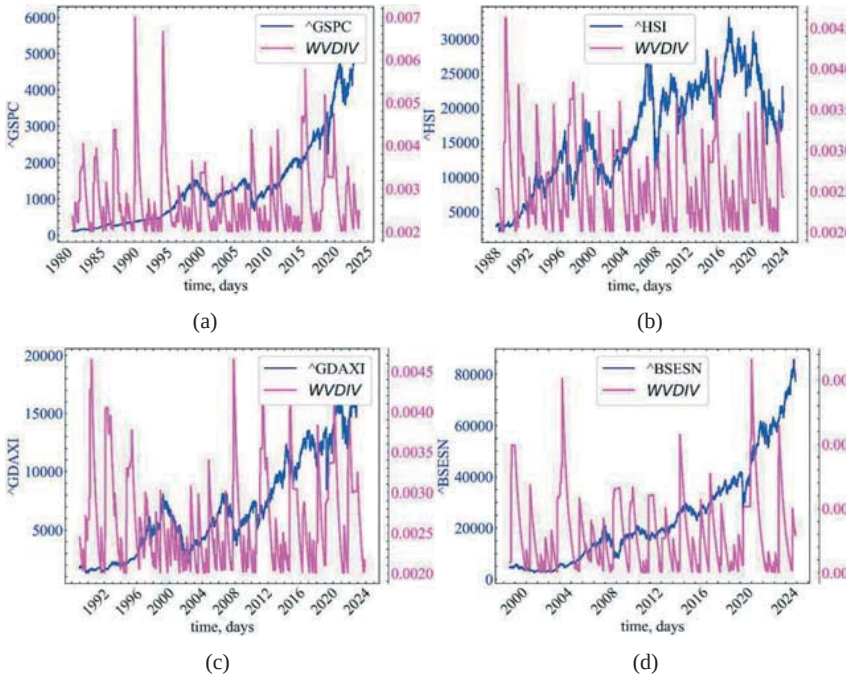


Fig. 2.16: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their divergence of white vertical lines

Fig. 2.16 shows that the divergence of the white vertical lines begins to increase in the pre-crisis periods of stock indices, indicating an increase in the degree of determinism of the system and a decrease in the time spent by the stock market phase trajectories in a divergent state.

2.5.2.13 Entropy of white vertical lines

The probability $p(\omega)$ that a white vertical line has length ω can be estimated from the frequency distribution $P(\omega)$ with $p(\omega) = P(\omega)/\sum_{\omega=\omega_{\min}}^{N} P(\omega)$. The Shannon entropy of the probability of white vertical lines is defined as

$$WVertEn = -\sum_{\omega=\omega_{min}}^{N} p(\omega) \ln p(\omega),$$

where ω_{min} is the minimum length of the white vertical line.

In Fig. 2.17 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their entropy of white vertical lines.

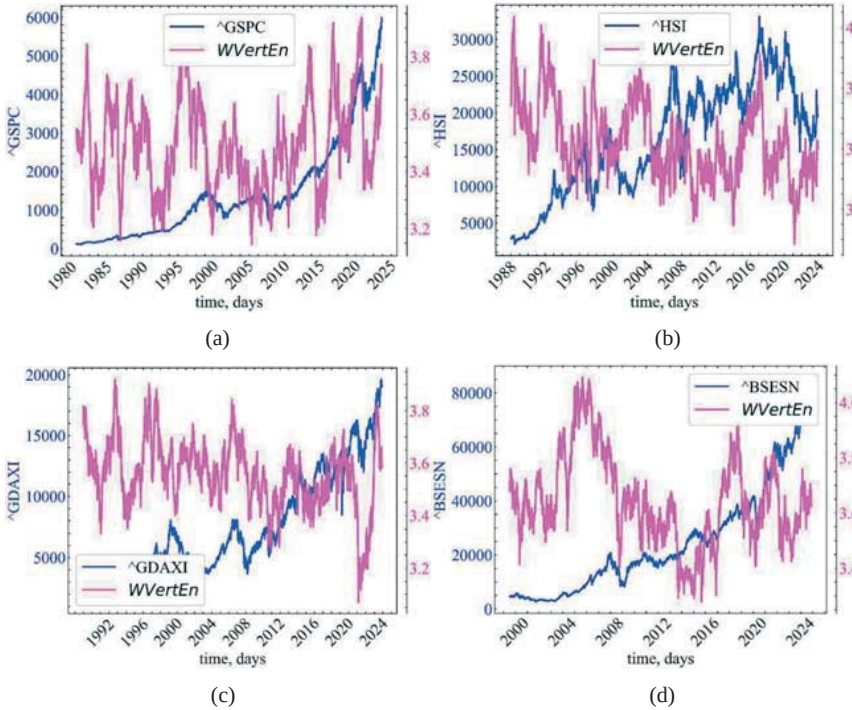


Fig. 2.17: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their entropy of white vertical lines

It can be seen that the entropy of the white vertical lines decreases in crisis and pre-crisis periods of the stock market and indicates an increase in the overall predictability of the system and a shift in the distribution of white vertical lines to specific lengths. That is, their distribution in times of crisis becomes less symmetrical and signals a gradual replacement of white vertical lines with black ones.

2.5.2.14 Recurrence rate to determinism ratio DET/RR

The **ratio** between DET and RR ($RATIO_1$) can be used to detect hidden phase transitions in a system:

$$RATIO_1 = DET/RR = N^2 \cdot \left(\sum_{l=l_{min}}^{N} l \cdot P(l) \right) \Bigg/ \left(\sum_{l=1}^{N} l \cdot P(l) \right)^2 .$$

In Fig. 2.18 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and ratio between DET and RR measures.

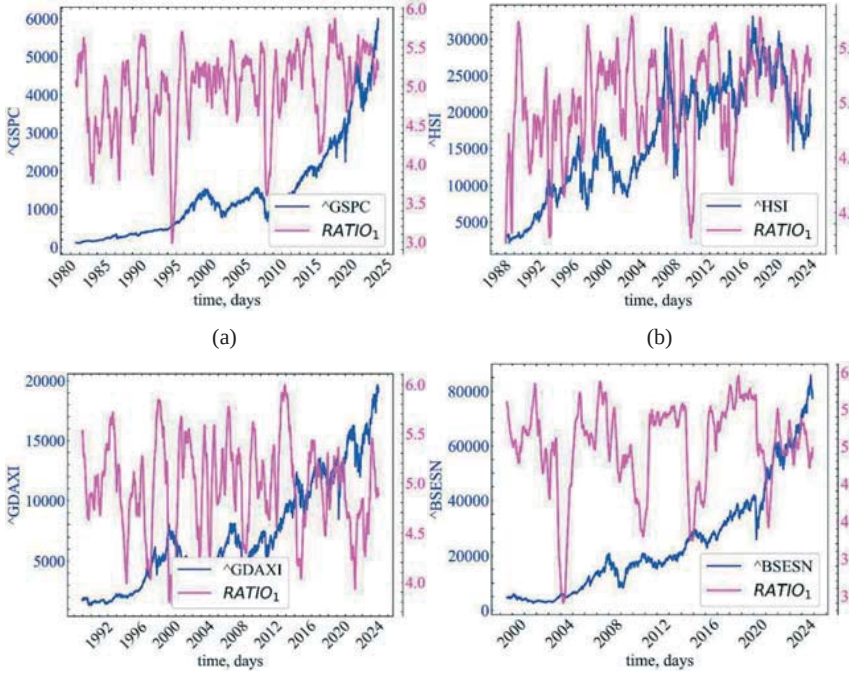


Fig. 2.18: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their ratio between the measure of DET and RR

This indicator decreases before stock market crises. This suggests that the overall density of recurrent points, both isolated and the entire distribution of vertical structures, should increase. In crisis periods, RR is higher than DET.

2.5.2.15 The ratio of laminarity to determinism LAM/DET

Just like the previous measure, the ratio of laminarity to determinism can allow us to identify hidden transitions in the signal under study:

$$RATIO_2 = LAM/DET.$$

In Fig. 2.19 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and ratio between LAM and DET measures.

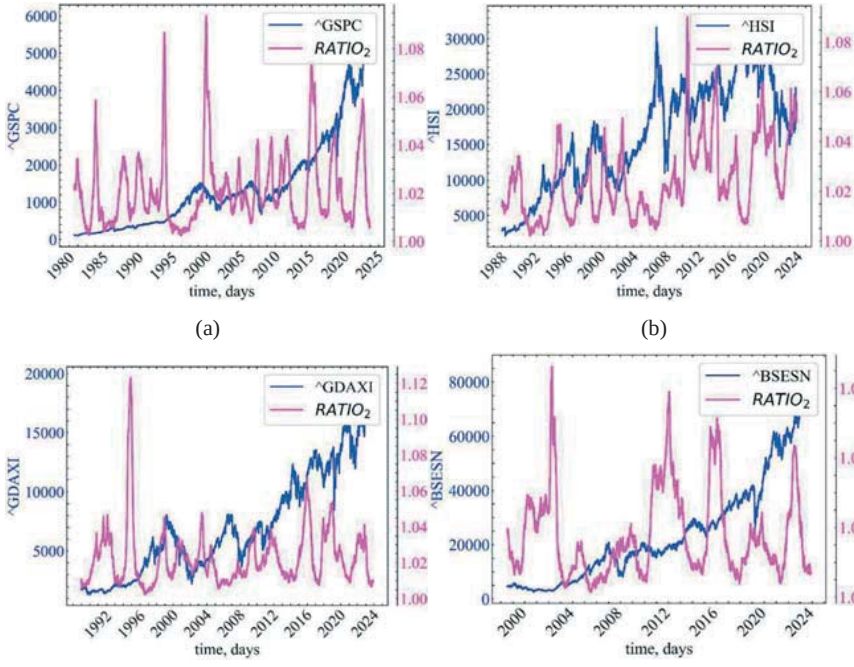


Fig. 2.19: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their ratio between the measure of LAM and DET

Regarding the dynamics of the $RATIO_2$ indicator, we can say that the overall degree of determinism begins to prevail over laminarity during crises.

2.6 Conclusions on recurrence analysis

In this section, quantitative recurrence measures were presented to study the evolution of the system. These measures were applied to a time series representing the closing prices of S&P 500, Hang Seng index, DAX, and BSE Sensex stock indices. It has been demonstrated that quantitative indicators are able to detect transitions between chaotic and periodic states (and vice versa), allow identifying laminar states (chaos-chaos transitions), states of determinism and the time until the onset of a state of predictability. Based on the results of the presented indicators, we can say that the studied collapse and pre-collapse events are characterized by an increase in recurrence, and this kind of behavior can be used as a harbinger of possible crisis phenomena.

3 Non-extensive Tsallis statistics

3.1 Non-equilibrium thermodynamics and non-extensive statistical mechanics

The great challenge of complexity theory, which is the basis of the modern scientific paradigm, originates from old and important problems such as the arrow of time, the existence of a simple and fundamental physical level for a single description of macroscopic and microscopic levels, the relationship between the observer and the object under study, etc. In general, with regard to the theory of complexity and each new level of reality, new concepts and new classifications are needed.

In particular, the theory of complexity includes: chaotic dynamics in the space of states, far from equilibrium phase transitions, long-term correlations, self-organization and multiscale, fractal processes in space and time, and other significant phenomena [4]. Complexity theory is considered the third scientific revolution of the last century (after relativity and quantum theory). However, complexity theory is still far from its academic maturity. In this direction, a significant contribution to the question of "what is complexity" can be found in the book by G. Nikolis and I. Prigogine [70]. Generally, we can summarize the basic concept of complexity theory as follows:

1. Complexity theory is a generalization of statistical physics for critical states of thermodynamic equilibrium and for processes far from equilibrium.
2. Complexity is the extension of dynamics to nonlinearity and strange dynamics.
3. Also, according to Ilya Prigogine, complexity theory is related to the dynamics of correlations instead of the dynamics of trajectories or wave functions.

According to complexity theory, various physical phenomena occurring in distributed physical systems, such as cosmic plasma, liquids or solids, chemistry, biology, ecosystems, DNA dynamics, socio-economic or information systems, networks can be described and understood in a similar way. This description is based on the principle of entropy maximization. Also, according to the theory of complexity, these systems are holistically stable dissipative structures formed by a general natural process aimed at maximizing entropy. From the point of view of complexity, there is no significant differentiation between a group of galaxies, stars, animals, flowers, or elementary particles, because everywhere we have open, dynamic, and self-organized systems and everywhere nature works to maximize entropy.

In the study of complex physical systems and phenomena, such as self-organizing and fractal structures, subdiffusion, turbulence, chemical reactions, and various economic, social, and biological systems, the Gibbs distribution does not provide a good fit to the observed phenomena. Many studies have shown that such systems are characterized by power distributions [46]. They are not derived from the Gibbs-Shannon maximum entropy principle, which is the basis of both **equilibrium** and **non-equilibrium statistical thermodynamics** [43-45]. This has led to numerous attempts to construct a generalized statistic that would provide power law asymptotics of the distribution function. Such generalized statistics can be constructed on the basis of several entropies. Among them, an important place is occupied by the **Tsallis entropy**.

Research in the field of mechanics of **non-extensive (non-additive)** systems has recently become a subject of considerable interest in connection with the manifestations of non-additive properties in anomalous physical phenomena. This is due to both the novelty of the general theoretical problems that arise here and the importance of practical applications (see the bibliography presented at (https://tsallis.cbpf.br/biblio.htm), which is constantly updated). The beginning of a systematic study in this area is associated with the work of Tsallis, in which the

author introduced a parametric formula for the statistical q-entropy, which depends on some real number q (the so-called deformation parameter) and is non-additive for a set of independent complex systems. The theory of non-extensional systems based on the Tsallis entropy is currently being intensively developed. These works have become a significant step in the development of the information theoretic approach and in the development of the principles of **non-extensive statistical mechanics** and equilibrium thermodynamics of open systems. It is important to note that the range of applications of these and many other non-extensive parametric entropies is currently constantly expanding, covering various areas of science, such as cosmology and cosmogony, plasma theory, quantum mechanics and statistics, nonlinear dynamics and fractals, geophysics, biomedicine, and many others.

From a physical point of view, economic dynamics can be viewed as spatially distributed dynamics and is related to the general category of nonlinear distributed systems. The analysis of economic time series demonstrates complex and chaotic dynamics in phase space. Taken's embedding theorem (using the method of delays) allows us to reconstruct a topological equivalent to the original phase space that preserves the basic geometric and dynamic properties, such as degrees of freedom, fractal dimension, multifractality, Lyapunov exponents, prediction matrix, etc. The reconstructed phase space can be used to estimate all of the above quantities, as well as phase transitions, statistical behavior, entropy generation, etc. In addition, the phase space can have multifractal properties and discontinuous turbulence characteristics, which indicate the existence of long-range interactions in space and time, as well as multiscale interactions.

These characteristics also indicate the existence of fractional dynamics in phase space, which can be described by the Fokker-Planck fractional differential equations and anomalous diffusion equations. The solutions of these equations are fractional space-time functions and non-Gaussian distribution functions, which belong to the category of Levy distributions and Tsallis distributions. Non-

equilibrium steady states of economic dynamics originate from processes of strong self-organization corresponding to local maxima of the Tsallis entropy, while changes in the control parameters of the economic system can cause a phase transition and a shift of economic dynamics to a new stable equilibrium, a steady state with maximum Tsallis entropy. This phase transition leads to a multifractal change in the formation of the phase space and to a change in the phenomenology of the economic system. Finally, the statistics of the dynamics in the multifractal phase space can be described by means of power functions of the Tsallis distribution with “heavy” tails, which can be used to improve forecasting methods.

In recent years, statistical mechanics has expanded its original purpose: the application of statistics to large systems whose states are governed by some kind of Hamiltonian functionals [46]. Their ability to relate the microscopic states of individual system components to macroscopic properties is now widely used [43]. Undoubtedly, the most important of these connections is still the determination of thermodynamic properties through the correspondence between the concept of entropy, originally introduced by Rudolf Clausius in 1865, and the number of allowed microscopic states, introduced by Ludwig Boltzmann around 1877 when he studied the approach to the equilibrium of an ideal gas [44]. This relationship can be expressed as

$$S = k \ln W, \tag{3.1}$$

where k is a positive constant, and W is the number of microstates compatible with the macroscopic state of an isolated system. This equation, known as Boltzmann's principle, is one of the cornerstones of standard statistical mechanics. When the system is not isolated, but instead is in contact with some large reservoir, we can modify Eq. (3.1) to obtain the Boltzmann-Gibbs entropy (BG):

$$S_{BG} = -k \sum_{i=1}^{W} p_i \ln p_i, \tag{3.2}$$

where p_i is the probability of a microscopic configuration [46]. BG statistical mechanics is still based on hypotheses such as molecular chaos [70] and ergodicity [42]. Despite the absence of an actual fundamental derivation, BG statistics has undoubtedly been successful in studying systems dominated by short spacetime interactions. Thus, it is quite possible that other physical entropies besides BG can be defined to properly describe anomalous systems for which the simplified ergodicity and/or independence hypothesis does not hold. Inspired by such concepts, in 1988 Constantino Tsallis proposed a generalization of BG statistical mechanics to cover systems that violate ergodicity, systems whose microscopic configurations cannot be considered independent. This generalization is based on non-additive entropies, characterized by an index and leading to non-extensive statistics:

$$S_q = -k\left(1 - \sum_{i=1}^{W} p_i^q\right)\Big/(1-q), \tag{3.3}$$

where p_i are the probabilities associated with microscopic configurations, W – their total number, q – a real number, and k – Boltzmann's constant. The value is a measure of the non-extensive nature of the system. It corresponds to the standard BG statistic. Eq. (3.3) modifies S_{BG} ($\lim q \to 1, S_q = S_{BG}$) as the basis for a possible generalization of BG statistical mechanics [60, 134]. The value of the entropy index for a particular system should be determined a priori from microscopic dynamics.

Since its introduction, the Tsallis entropy (3.3) has been the source of several important results in both fundamental and applied physics, as well as in other scientific fields such as biology, chemistry, economics, geophysics, and medicine [71].

3.2 Non-extensive entropy and Tsallis triplet

Systems characterized by BG statistical mechanics have the following characteristics: (i) their distribution functions for energies are proportional to an

exponential function; (ii) they have a strong sensitivity to initial conditions that grows exponentially with time (chaos), characterized by a positive maximum Lyapunov exponent; (iii) their relaxation occurs exponentially with a certain relaxation time. In other words, these three behaviors are described by exponential functions (i.e., $q = 1$). However, it has been found that for systems that can be studied within the framework of non-expansive statistical mechanics, the energy probability density function (associated with stationarity or equilibrium), sensitivity to initial conditions, and relaxation are described by three entropy indices called the **Tsallis triplet**, or q-triplet [43, 69].

Non-extensive statistical theory is mathematically based on the nonlinear equation

$$\frac{dy}{dx} = y^q, \tag{3.4}$$

the solution of which is the q-exponential function:

$$\exp_q(x) = \begin{cases} (1 + (1-q)x)^{1/(1-q)}, & \text{if } 1 + (1-q)x > 0, \\ 0, & \text{if } 1 + (1-q)x \leq 0. \end{cases} \tag{3.5}$$

For $q \to 1$, q-Gaussian corresponds to the usual Gaussian distribution.

The solution of Eq. (3.4) can be realized in three different ways included in the Tsallis q-triplet: $(q_{sens}, q_{stat}, q_{rel})$. These quantities characterize the three physical processes that are summarized here, while the q-triplet values characterize the attractor set of dynamics in the phase space of the dynamic, and they can change when the dynamics of the system is attracted to another set of attractors.

For a non-extensive system, the value of the q-exponent depends on the estimated properties of the dynamics and phase space of the system. For dynamic systems, a q-triplet is estimated, which reflects three properties of the system (Fig. 3.1). The q_{stat} index is estimated on the basis of an equilibrium model of the rank distribution using nonlinear estimation methods [151]. This index is a parameter of the system's area of attraction. The q_{sens} exponent reflects the sensitivity of the system to initial conditions and entropy production and is determined by the

multifractal spectrum [32]. The relaxation index q_{rel} is found on the basis of autocorrelation and characteristics of diffusion processes [47].

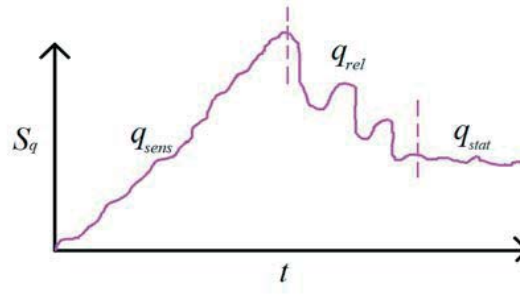


Fig. 3.1: Time periodization of the periods of q-entropy production. The first period corresponds to the production of entropy through the q_{sens} parameter of the Tsallis q-triplet. The second period corresponds to a certain relaxation process through the parameter q_{rel}. The system detects fluctuations due to the q_{stat} parameter

3.2.1 Index q_{stat} and non-extensive physical conditions

The value of q for the steady state is estimated from the yield distribution function, which in turn is obtained by fitting q-Gaussian:

$$P_q(\beta, x) = \left(\sqrt{\beta}/C_q\right) \exp(-\beta r x^2) \tag{3.6}$$

for an empirically constructed histogram $\{p(x_i) \mid i = 1, \ldots, N\}$ and various β values selected by minimizing $\sum_i \left[P_{q_{stat}}(\beta, x_i) - p(x_i)\right]^2$. Depending on the value of q, C_q can take the following forms:

$$C_q = \begin{cases} 2\sqrt{\pi}\,\Gamma\left(\dfrac{1}{1-q}\right)/(3-q)\sqrt{1-q}\,\Gamma\left(\dfrac{3-q}{2(1-q)}\right), & \text{if } -\infty < q < 1, \\ \sqrt{\pi}, & \text{if } q = 1, \\ \sqrt{\pi}\,\Gamma\left(\dfrac{3-q}{2(q-1)}\right)/\sqrt{q-1}\,\Gamma\left(\dfrac{1}{q-1}\right), & \text{if } 1 < q < 3. \end{cases} \tag{3.7}$$

To assess the dynamics of the q value, a graph of the dependence of $\ln_q[p(x)]$ on x^2 is plotted for the selected interval q (for example, from 1 to 5), which provides the best linear approximation (estimated by the maximum

coefficient of determination R^2) [54]. It is clear that the values of $p(x)$ become markedly non-Gaussian along the tails, and can instead be described by a power law.

3.2.2 Study of relaxation processes through the prism of the q_{rel}

The corresponding q-value for the relaxation process is obtained from the autocorrelation coefficient:

$$C(\tau) = \sum_t |g_{t+\tau}| \cdot |g_t| \Big/ \sum_t |g_t|^2. \tag{3.8}$$

For BG statistics, such a correlation should decrease exponentially. The same algorithm as for q_{stat} must be worked out on a graph of the dependence of $\ln_q[C(\tau)]$on τ to determine which q best linearizes empirical data.

3.2.3 Sensitivity to the initial conditions $q = q_{sens}$

Entropy production is related to the general nature of the attractor set. This attractor can be described by multifractality and sensitivity to initial conditions. The sensitivity to initial conditions can be expressed as

$$\frac{d\xi}{dt} = \lambda_1 \xi + \left(\lambda_q - \lambda_1\right)\xi^q, \tag{3.9}$$

where ξ is the deviation of the trajectory in phase space: $\xi \equiv \lim_{\delta \to 0}[\delta(t)/\delta(0)]$, and $\delta(t)$ is the distance between adjacent trajectories due to time t. The solution of Eq. (3.9) can be represented as:

$$\xi = \left[1 - \left(\frac{\lambda_{q_{sens}}}{\lambda_1}\right) + \left(\frac{\lambda_{q_{sens}}}{\lambda_1}\right) exp\left((1 - q_{sens})\lambda_1 t\right)\right]^{1/(1-q_{sens})}. \tag{3.10}$$

First, it was hypothesized and later proved for time series of non-extensive systems of different nature that such a correlation exists [14]:

$$1/(1 - q_{sens}) = 1/\alpha_{min} - 1/\alpha_{max}, \tag{3.11}$$

where α_{min} and α_{max} are the minimum and maximum α values of the corresponding multifractal spectrum $f(\alpha)$.

The spectrum of multifractality, therefore, follows from the procedure of multifractal detrended fluctuation analysis, which allows you to calculate the Hurst exponent for different moments and time scales.

3.2.4 Practical calculations of *q*-triplet

Let's look at how these indicators can be used as indicators of crisis states. First, let's import the necessary libraries:

```
import matplotlib.pyplot as plt
import numpy as np
import yfinance as yf
import pandas as pd
import scienceplots
import neurokit2 as nk
import fathon
import scipy
import statsmodels.api as sm
from fathon import fathonUtils as fu
from scipy.stats import norm
from scipy.special import gamma
from scipy.optimize import curve_fit
from tqdm import tqdm

%matplotlib inline
```

And we will define the necessary functions for further work:

```
# q-exponential function
def np_exp_q(x, q=1):
    if q == 1:
       return np.exp(x)
    else:
       return (1+(1-q)*x)**(1/(1-q))

# q-logarithm
def np_log_q(x, q=1):
    if q == 1:
       return np.log(x)
    else:
       return x**(1-q)-1/(1-q)

# values for q-Gaussian
def C_q(q=1.0):
    if q == 1:
       return np.sqrt(np.pi)
    elif q < 1:
       return 2*np.sqrt(np.pi)*gamma(1/(1-q))/(3-q)*np.sqrt(1-q)*gamma((3-q)/
(2*(1-q)))
    elif q > 1:
       return (np.sqrt(np.pi)*gamma((3-q))/(2*(q-1)))/(np.sqrt(q-1)*gamma(1/(
```

```
q-1)))

# pdf of q-Gaussian for q_stat calculations
def G_q(r, beta, q):
    return np.sqrt(beta)/C_q(q) * np_exp_q(-beta*r, q)

# autocorrelation function for q_rel
def acf(x, maxlag):

    n = len(x)
    a = (x - x.mean()) / (x.std() * n)
    b = (x - x.mean()) / x.std()

    cor = np.correlate(a, b, mode="full")
    acf = cor[n:n+maxlag+1]
    lags = np.arange(maxlag +1)

    return acf, lags

# relaxation function for q_rel
def rel_func(x, q, tau):
    return np_exp_q(-x/tau, q)

# a function for calculating the returns of a series or its standardization
def transformation(signal, ret_type):

    for_rec = signal.copy()

    if ret_type == 1:
       pass
    elif ret_type == 2:
        for_rec = for_rec.diff()
    elif ret_type == 3:
        for_rec = for_rec.pct_change()
    elif ret_type == 4:
        for_rec = for_rec.pct_change()
        for_rec -= for_rec.mean()
        for_rec /= for_rec.std()
    elif ret_type == 5:
        for_rec = for_rec.pct_change()
        for_rec -= for_rec.mean()
        for_rec /= for_rec.std()
        for_rec = for_rec.abs()
    elif ret_type == 6:
        for_rec -= for_rec.mean()
        for_rec /= for_rec.std()

    for_rec = for_rec.dropna().values

    return for_rec

# function for plotting paired charts
def plot_pair(x_values,
              y1_values,
              y2_values,
              y1_label,
```

```
              y2_label,
              x_label,
              file_name, clr="magenta"):

    fig, ax = plt.subplots()

    ax2 = ax.twinx()
    ax2.spines.right.set_position(("axes", 1.03))

    p1, = ax.plot(x_values,
                  y1_values,
"b-", label=fr"{y1_label}")
    p2, = ax2.plot(x_values,
                   y2_values,
                   color=clr,
                   label=y2_label)

    ax.set_xlabel(x_label)
    ax.set_ylabel(f"{y1_label}")
    ax.yaxis.label.set_color(p1.get_color())
    ax2.yaxis.label.set_color(p2.get_color())

    tkw = dict(size=2, width=1.5)

    ax.tick_params(axis='x', rotation=45, **tkw)
    ax.tick_params(axis='y', colors=p1.get_color(), **tkw)
    ax2.tick_params(axis='y', colors=p2.get_color(), **tkw)
    ax2.legend(handles=[p1, p2])

    plt.savefig(file_name +".jpg")
    plt.show();
```

Next, let's configure the output format of the figures:

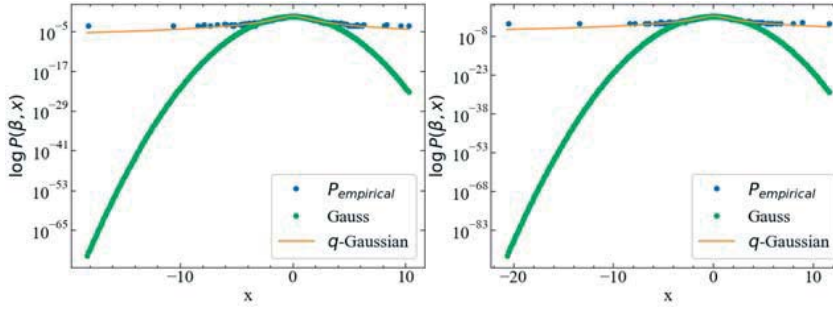
```
plt.style.use(['science', 'notebook', 'grid'])

size = 22
params = {
'figure.figsize': (8, 6),
'font.size': size,
'lines.linewidth': 2,
'axes.titlesize': 'small',
'axes.labelsize': size,
'legend.fontsize': size,
'xtick.labelsize': size,
'ytick.labelsize': size,
"font.family": "Serif",
"font.serif": ["Times New Roman"],
'savefig.dpi': 300,
'axes.grid': False
}

plt.rcParams.update(params)
```

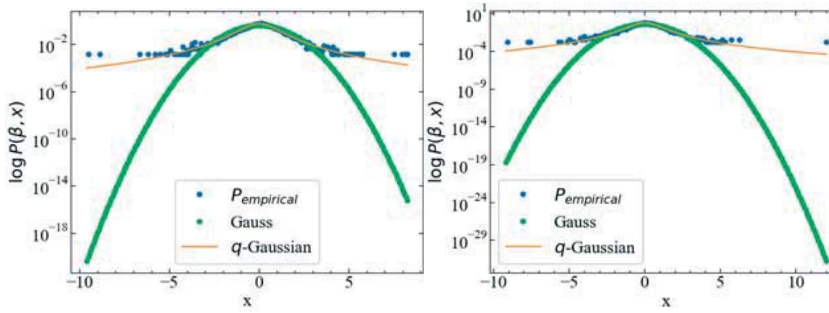
3.2.5 Calculations of the q_{stat} exponent

3.2.5.1 q-Gaussian estimation for the whole time series

```
q_stat_time_ser = time_ser.copy()
ret_type = 4                          # type of a series
q_stat_time_ser = transformation(q_stat_time_ser, ret_type)

hist, bin_edg = np.histogram(q_stat_time_ser, bins=250, density=True)

mu, std = norm.fit(q_stat_time_ser)
x = np.linspace(q_stat_time_ser.min(), q_stat_time_ser.max(), len(bin_edg[1:]
))
p = norm.pdf(x, mu, std)

xval = bin_edg[1:]**2
yval = hist

popt, pcov = curve_fit(G_q, xdata=xval, ydata=yval, bounds=([0.0, 0.0], [np.i
nf, 3.0]))

fig, ax = plt.subplots(1, 1)
ax.plot(bin_edg[1:], hist, 'o', label=r"$P_{empirical}$")
ax.plot(x, p, 'o', label="Gauss")
ax.plot(x, G_q(x**2, popt[0], popt[1]), label=r"$q$-Gaussian")
ax.set_yscale('log')
ax.set_xlabel("x")
ax.set_ylabel(r"$\log{P(\beta, x)}$")

plt.legend()
plt.show();
```

Fig. 3.2 demonstrates distribution function of normalized returns for S&P 500, Hang Seng index, DAX, and BSE Sensex compared to theoretical Gaussian and q-Gaussian distributions





(c) (d)

Fig. 3.2: Distribution function of normalized returns for S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d) compared to theoretical Gaussian and q-Gaussian distributions

Fig. 3.2 shows that the standardized returns for the studied stock indices go beyond $\pm 10\sigma$. As can be seen, the theoretical Gaussian distribution significantly underestimates the occurrence of extremely high and low returns. If the logarithm of the empirical probability for such returns is, approximately, at the level of 10^{-5} for S&P 500, 10^{-8} for Hang Seng, 10^{-2} for DAX, and 10^{-4} for BSE Sensex, then the Gaussian distribution is, approximately, 10^{-65} for S&P 500, 10^{-83} for Hang Seng, 10^{-18} for DAX, and 10^{-30} for BSE Sensex. That is, as an example, the Gaussian distribution underestimates the empirical probability of positive returns by a factor of 10^{25}. Although the q-Gaussian also does not seem ideal for describing such returns, the underestimation of heavy tails in the case of non-extensive statistics is much smaller compared to the normal Gaussian.

3.2.5.2 q_{stat} calculations within the sliding window procedure

```
window = 500   # sliding window width
tstep = 1      # sliding window step
ret_type = 4   # type of a series:

length = len(time_ser)

q_stats = []

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)
```

```
    hist_fragm, bin_edg_fragm = np.histogram(fragm, bins=100, density=True)

    xval = bin_edg_fragm[1:]**2
    yval = hist_fragm

    popt, pcov = curve_fit(G_q, xdata=xval, ydata=yval, bounds=([0.01, 1.0],
[np.inf, 5.0]))
    q_stat = popt[1]

    q_stats.append(q_stat)
```

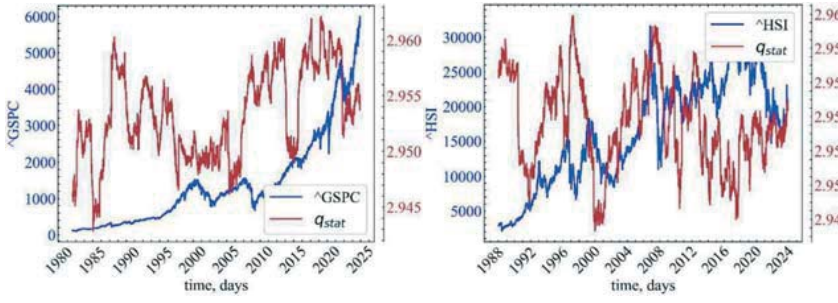
Saving the results to a text file:

```
name = f"q_stat_name={symbol}_window={window}_step={tstep}_rettype={ret_type}
.txt"

np.savetxt(name, q_stats)
```

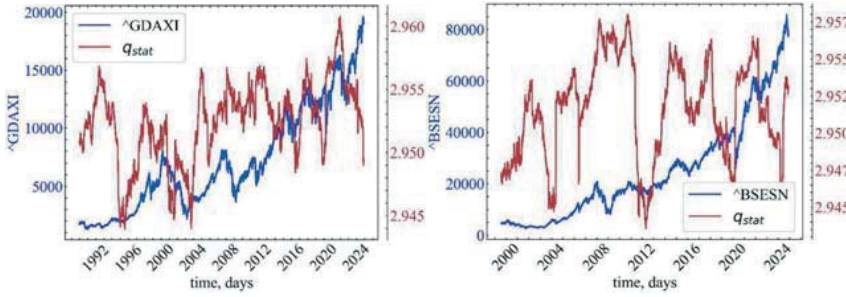
Defining the parameters for saving figures:

```
# labeling of the q_stat indicator in the figure legend
label_q_stat = r'$q_{stat}$'

# figure title
file_name = f"q_stat_name={symbol}_window={window}_step={tstep}_rettype={ret_
type}"

# color of the indicator
color = 'brown'

plot_pair(time_ser.index[window:length:tstep], time_ser.values[window:length:
tstep], q_stats, ylabel, label_q_stat, xlabel, file_name, color)
```

Fig. 3.3 represents the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their q_{stat} exponent.



(a) (b)



(c) (d)

Fig. 3.3: Comparative dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d), and their q_{stat} exponent

3.2.6 Calculation of the q_{rel} exponent

```
window = 500   # sliding window width
tstep = 1      # sliding window time step
ret_type = 1   # type of a series:

max_lag = 100

length = len(time_ser)

q_rels = []

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)
    autocor, lags = acf(x=fragm, maxlag=max_lag)
    lags = lags
    autocor = autocor

    popt, pcov = curve_fit(rel_func, xdata=lags[1:], ydata=autocor[1:], bound
s=(1, [np.inf, 10]))
    q_rel = popt[0]

    q_rels.append(q_rel)
```

Saving the results to a text file:

```
name = f"q_rel_name={symbol}_window={window}_step={tstep}_rettype={ret_type}_
maxlag={max_lag}.txt"

np.savetxt(name, q_rels)
```

Defining the parameters for saving figures:

```
# labeling of the q_rel indicator in the figure legend
label_q_rel = r'$q_{rel}$'
```

```
# figure title
file_name = f"q_rel_name={symbol}_window={window}_step={tstep}_rettype={ret_t
ype}_maxlag={max_lag}"

# color of the exponent
color = 'red'
```

Plot the results:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          q_rels,
          ylabel,
          label_q_rel,
          xlabel,
          file_name,
          color)
```

In Fig. 3.4 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their q_{rel} exponent.

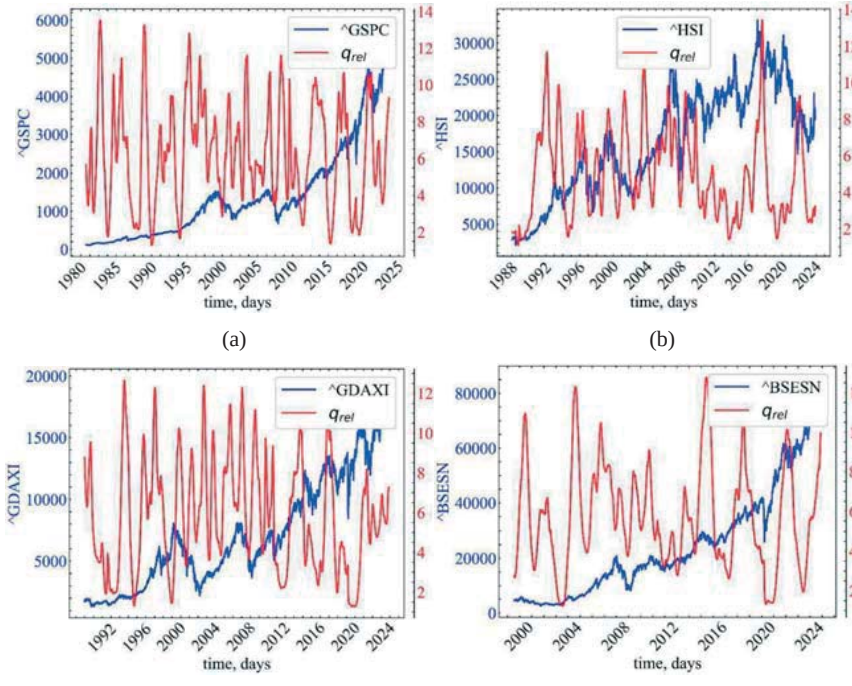


Fig. 3.4: Comparative dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their q_{rel} exponent

For the studied indicator, Fig. 3.4 shows that the degree of relaxation increases in the pre-crisis state of the system, which is an indicator of the growth of traders' self-organization through certain external indicators.

3.2.7 Calculation of the q_{sens} exponent

```
window = 500   # sliding window width
tstep = 1      # sliding window step
ret_type = 4   # type of a series:

rev = True # whether to repeat the calculation of the fluctuation function fr
om the end of the series
accumulate = False # re-accumulation of a detrended series to work with highl
y uncorrelated series

q_min = -5 # minimum value of q
q_max = 5  # maximum value of q
q_inc = 1  # increment step of q

win_beg = 10 # Initial segment width
win_end = window-1 # Final segment width

length = len(time_ser)

q = np.arange(q_min, q_max+q_inc, q_inc)
q = np.round_(q, decimals =1)

order = 3 # polynomial order for detrending (MF-DFA)

q_sens_values = []

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()

    fragm = transformation(fragm, ret_type)

    if accumulate == True:
        fragm = np.cumsum(fragm-np.mean(fragm))

    a = fu.toAggregated(fragm)

    pymfdfa = fathon.MFDFA(a)

    wins = fu.linRangeByStep(win_beg, win_end)

    n, F = pymfdfa.computeFlucVec(wins, q, revSeg=rev, polOrd=order)
    list_H, list_H_intercept = pymfdfa.fitFlucVec()

    if accumulate == True:
        list_H = list_H - 1
```

```
# calculation of tau(q) values
    tau = q * list_H - 1

# calculation of singularity values
    alpha = np.gradient(tau, q, edge_order=2)

# maximum value of the singularity
    maximal_alpha = alpha.max()

# minimum value of the singularity
    minimal_alpha = alpha.min()

# q_sens calculations
    q_sens = (maximal_alpha-minimal_alpha-maximal_alpha*minimal_alpha)/(maxim
al_alpha-minimal_alpha)

    q_sens_values.append(q_sens)
```

Saving the results to a text file:

```
name = f"q_sens_name={symbol}_ret={ret_type}_qmin={q_min}_qmax={q_max}_qinc={
q_inc}_wind={window}_step={tstep}.txt"

np.savetxt(name, q_sens_values)
```

Defining the parameters for saving figures:

```
# labeling of the q_rel indicator in the figure legend
label_q_sens = r'$q_{sens}$'

# figure title
file_name = f"q_sens_name={symbol}_ret={ret_type}_qmin={q_min}_qmax={q_max}_q
inc={q_inc}_wind={window}_step={tstep}"

# color of the indicator
color = 'green'
```

Plot the results:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          q_sens_values,
          ylabel,
          label_q_sens,
          xlabel,
          file_name,
          color)
```

Fig 3.5 presents the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their q_{sens} exponent.

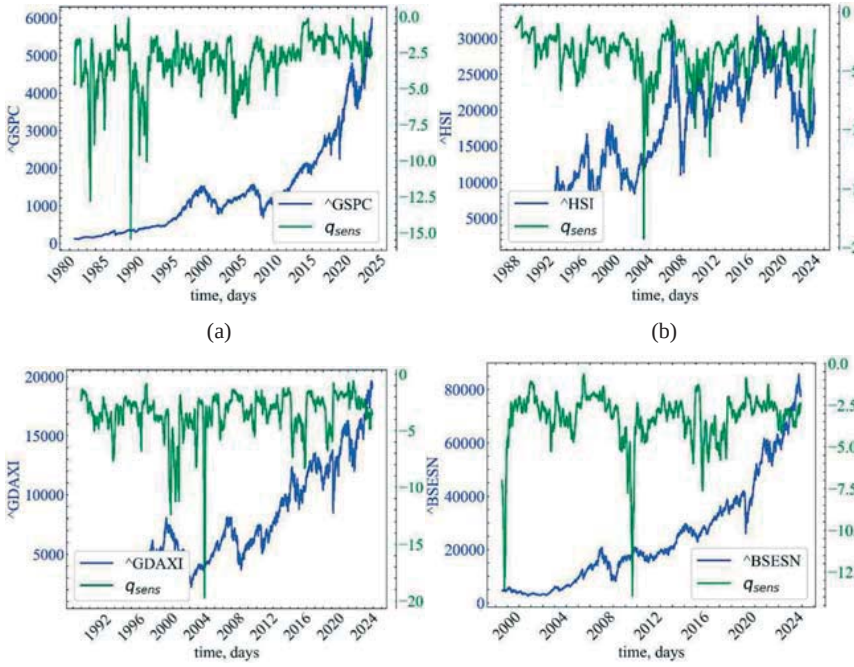


Fig. 3.5: Comparative dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their q_{sens} exponent

The q_{sens} indicator shows a decline in the pre-crisis periods, indicating that the market is particularly sensitive at these moments. For completely identical and independently distributed values, q_{sens} would remain close to 1. In pre-crisis states, it tends to negative values, which indicates the convergence of the system's attractor to singularity, i.e. the convergence of trajectories to each other.

3.2.8 Tsallis entropy calculations

```
window = 500   # sliding window width
tstep = 1      # sliding window time step
ret_type = 1   # type of a series

length = len(time_ser)

tsallis_en = []
```

```
for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)
    p, be = np.histogram(fragm, bins='auto',
                         density=True)
    r = be[1:] - be[:-1]
    P = p * r
    P = P[P!=0]

    tsen, _ = nk.entropy_tsallis(freq=P,
                                 q=1,
                                 base=np.exp(1))
    tsen /= np.log(len(P))

    tsallis_en.append(tsen)
```

Saving the results to a text file:

```
name = f"tsen_name={symbol}_ret={ret_type}_wind={window}_step={tstep}.txt"

np.savetxt(name, tsallis_en)
```

Defining the parameters for saving figures:

```
# Tsallis entropy notation in the figure legend
label_ts_en = r'$TsEn$'

# figure title
file_name = f"tsen_name={symbol}_ret={ret_type}_wind={window}_step={tstep}"

# color of the indicator
color = 'purple'
```

Plot the results:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          tsallis_en,
          ylabel,
          label_ts_en,
          xlabel,
          file_name,
          color)
```

Fig. 3.6 shows the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their Tsallis entropy

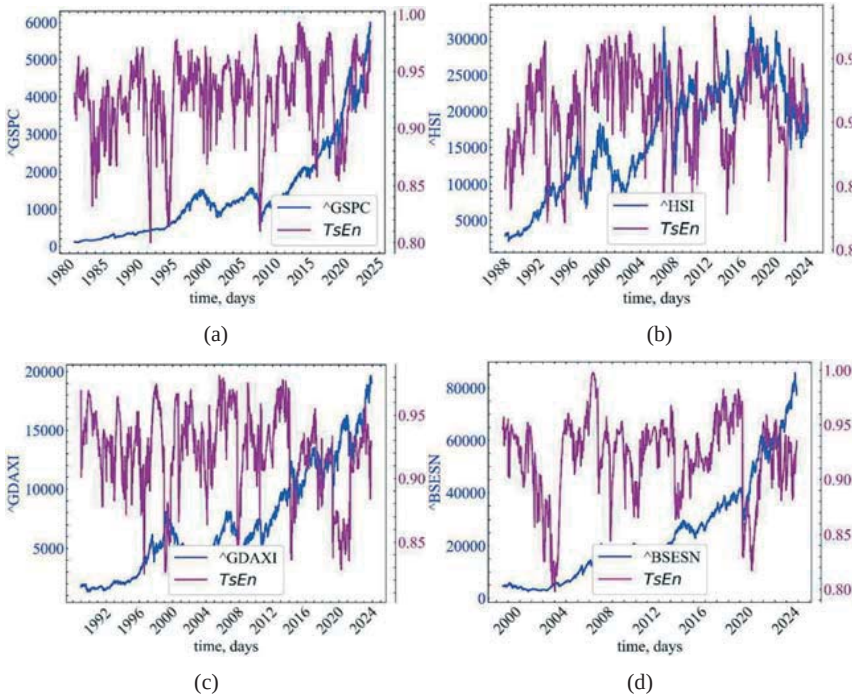


Fig. 3.6: Comparative dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their Tsallis entropy

Fig. 3.6 shows that the non-extensive Tsallis entropy decreases in the pre-crisis periods, indicating an increase in the degree of non-additivity (self-organized dynamics) of the market.

3.3 Conclusions on non-extensive statistics and q-triplet

In this chapter, a non-extensive statistical mechanics approach to the dynamics of daily historical values of the major stock indices and their returns is presented. It was found that the stock indices obey the Tsallis statistics. The time dynamics of the q-triple was modeled, which made it possible to obtain the reaction of the components of the triple to the formation and course of crisis phenomena when compared with the original time series. The q_{stat} value increases

in times of crises, as price fluctuations increase. The q_{rel} value increases in pre-crisis periods, which is obviously due to the system's transition to a non-equilibrium state and subsequent relaxation. Finally, q_{sens} has a minimal value in the pre-crisis period, indicating a special sensitivity of the system near the bifurcation point, which is the crisis itself.

It seems promising to study the features of the q-triplet for complex network structures obtained by transforming a time series into a network using one of the known methods. It is also interesting to search for alternative components of non-extensivity, such as a measure of irreversibility of the time series, or a measure of recurrence, etc. Obviously, these approaches can provide the necessary progress both at the fundamental and applied levels to achieve a deeper understanding of the nature of complex systems.

4 Fractal and multifractal measures of complexity

4.1 Definition of a fractal

Fractals are geometric objects: lines, surfaces, spatial bodies that have a highly rough surface or shape and are characterized by the property of *self-similarity* [25, 26, 76]. The word fractal comes from the Latin word *fractus* and is translated as fractional, broken. Self-similarity as the main characteristic of a fractal means that it varies more or less homogeneously over a wide range of scales. Thus, when zoomed in, small fractal fragments become very similar to large ones. In the ideal case, this self-similarity leads to the fact that the fractal object is invariant with respect to stretching, i.e., it is said to have dilatational symmetry. It implies that the main geometric features of a fractal remain unchanged when the scale changes.

Obviously, fractal objects in the real world are not infinitely self-similar, and there is a minimum scale l_{min} such that the self-similarity property disappears at a scale $l \approx l_{min}$. In addition, at sufficiently large length scales $l > l_{max}$, where l_{max} is the characteristic geometric size of objects, this self-similarity property is also violated. Therefore, the properties of natural fractals are considered only on scales l that satisfy the relation $l_{min} \ll l \ll l_{max}$. Such restrictions are natural, since when we give an example of a fractal – a broken, nonsmooth trajectory of a Brownian particle – we understand that this image represents an obvious idealization. The fact is that on small scales, the finite mass and size of the Brownian particle, as well as the finite time of the collision, are hidden. When these circumstances are taken into account, the trajectory of a Brownian particle begins to represent a smooth curve.

It is worth noting that the property of exact self-similarity is characteristic only of **regular fractals**. If, instead of a deterministic method of construction, some element of randomness is included in the algorithm of their creation (as is the

case, for example, in many processes of differential cluster growth, electrical breakdown, etc.), then so-called **random fractals** appear. Their main difference from regular fractals is that the self-similarity properties are valid only after appropriate averaging over all statistically independent realizations of the object. In this case, the enlarged part of the fractal is not exactly identical to the original fragment, but their statistical characteristics coincide.

4.2 Coastline length

Initially, the concept of a fractal in physics arose in connection with the task of determining the length of a coastline. When it was measured using an existing map of the area, an interesting detail emerged: the larger the map used, the longer the coastline appeared to be [23, 27, 31, 78]. Let's say, for example, that the straight line distance between points A and B located on the coastline is R (see Fig. 4.1).

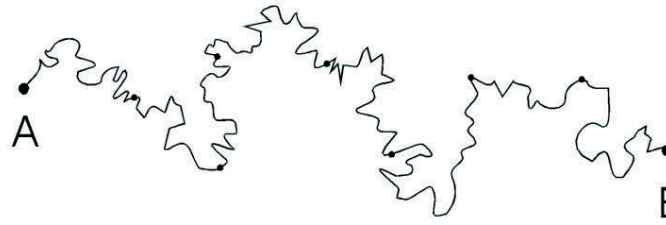


Fig. 4.1: Determining the length of the coastline between points A and B [152]

Then, to measure the length of the coastline between these points, we will place rigidly connected nodes along the shore so that the distance between adjacent nodes would be, for example, $l = 10$ km. The length of the coastline in kilometers between points A and B is then equal to the number of nodes minus 1 multiplied by 10. The next measurement of this length will be done in the same way, but the distance between neighboring nodes will be $l = 1$ km.

It turns out that the result of these measurements will be different. As the scale l decreases, we will get larger and larger values of the length. Unlike a smooth curve, the coastline is often so indented (down to the smallest scale) that as the length of the link l decreases, the value of L – the length of the coastline – does not tend to a finite limit, but increases according to a power law

$$L \approx l(R/l)^D, \tag{4.1}$$

and $D > 1$ is a certain power-law index called the **fractal dimension** of the coastline [31]. The larger the value of D, the more broken or detailed the coastline appears. The origin of the Eq. (4.1) should be intuitive: the smaller the scale we use, the less details of the coastline will be taken into account and the less they will contribute to the measured length. On the contrary, by increasing the scale, we "unfold" the coast, reducing the length L.

Thus, we can see that to determine the length of the coastline L using a rigid scale l, we need to make $N = L/l$ steps, and the value of L varies with l so that N depends on l according to the law $N \approx (R/l)^D$. As a result, the length of the coastline grows unlimitedly as the scale decreases. This circumstance sharply distinguishes a fractal curve from an ordinary smooth curve (such as a circle or ellipse), for which the limit of the length of the approximating broken line L is finite as the length of its link l approaches zero. As a result, for a smooth curve, its fractal dimension is $D = 1$, i.e., it coincides with the topological dimension.

4.3 Fractal dimension of sets

Previously, we introduced the concept of the fractal dimension of the coastline. Now let us give a general definition of this quantity. Let d be the usual Euclidean dimension of the space in which our fractal object is located ($d = 1$ – line, $d = 2$ – plane, $d = 3$ – usual three-dimensional space). Now let's cover this object with entirely d-dimensional "spheres" of radius 1. Suppose that we needed

at least $N(l)$ spheres for this. Then, if for sufficiently small l the value of $N(l)$ varies with l according to the power law

$$N(l) \sim 1/l^D, \tag{4.2}$$

then D is called the **Hausdorff** or **fractal dimension** of this object [155]. Obviously, this formula is equivalent to the relation $N \approx (R/l)^D$, which was used above to determine the length of the coastline. Eq. (4.3) can be rewritten as

$$D = -\lim_{l \to 0} \ln N(l)/\ln l. \tag{4.3}$$

This ratio serves as a general definition of the fractal dimension D. According to it, the value of D represents the **local** characteristic of the object under study.

4.4 Procedures for calculating monofractal dimensions

Currently, there are many definitions and methods for measuring fractal dimension. The most common one-dimensional fractal dimensions are the Hausdorff dimension, the Higuchi dimension, and the Petrosian and Minkowski dimension [95]. The Hausdorff dimension is the simplest fractal dimension. However, its computational complexity is high, which makes it difficult to apply in practice. The Minkowski dimension is relatively simple, and the fractal dimension of the signal can be obtained by adjusting the size of the length of the side of the cell within which the "roughness" of the signal surface is determined. Therefore, it is widely recognized and used. Which of the fractal dimension indicators most accurately describes the complexity of the signal and is able to identify crisis phenomena is the key point of this section.

4.4.1 R/S analysis

The Rescaled range (R/S analysis) method, developed by Mandelbrot and Wallace [25], is based on the previously established Hurst hydrological analysis method [74, 75], and allows for the calculation of the self-similarity parameter H, which measures the intensity of long-term dependencies in a time series. The

coefficient H, called the Hurst coefficient, contains minimal predictions about the nature of the system under study and can classify time series. This indicator distinguishes between random (Gaussian) and nonrandom series; in addition, it is associated with the fractal dimension, which, in turn, characterizes the degree of smoothness of the graph based on the time series. The R/S analysis method can also identify the maximum length of the interval (cycle) at which values retain information about the system's initial data (long-term memory).

The analysis begins with the construction of a series of logarithmic returns, $G(t) \equiv \ln x(t + \Delta t) - \ln x(t)$, where $x(t)$ is the value of the original time series at time t, Δt is the time step. The resulting sequence $G(t)$ is divided into d subsequences of length n.

For each subsequence $m = 1, \ldots, d$:

1) The mean value μ_m and the standard deviation S_m are found.
2) The data is normalized by subtracting the mean of the sequence $X_{i,m} = G_{i,m} - \mu_m$, $i = 1, \ldots, n$.
3) The cumulative sum of the sequence of Xs is found: $Y_{i,m} = \sum_{j=1}^{i} X_{j,m}$, $i = 1, \ldots, n$.
4) Within each subsequence is the range between the maximum and minimum values: $R_m = \max\{Y_{1,m}, \ldots, Y_{n,m}\} - \min\{Y_{1,m}, \ldots, Y_{n,m}\}$, which is standardized by the standard deviation R_m/S_m.
5) The average $(R/S)_n$ of the normalized range values for all subsequences of length n is calculated.

The R/S-statistics calculated in this way corresponds to the ratio $(R/S)_n \cong cn^H$, where the value of H can be obtained by calculating $(R/S)_n$ for sequences of intervals with increasing time horizon:

$$\log(R/S)_n = \log c + H \log n\,. \qquad (4.4)$$

The Hurst coefficient can be found by plotting the relationship $(R/S)_n$ vs. n on a double logarithmic scale and taking the slope of the line interpolating the points of the resulting graph. If $H = 0.5$, the sequence is **white noise**; $0.5 < H \leq 1$

indicates a **persistent** (trending) series, when there is a tendency for large values of the series to follow large values and vice versa; $H < 0.5$ indicates an **anti-persistent** (mean-reversion) series.

As the time horizon increases, the slope of the interpolating line should tend to $H = 0.5$; the transition process itself indicates the loss of the influence of the initial conditions on the current values, and thus we can talk about the long memory horizon – this is the point before which the slope of the interpolating line is different from 0.5, and after – about 0.5.

> ⓘ Note to the R/S analysis
>
> There is also a relationship between fractal dimension and Hurst exponent
>
> $$D_f = 2 - H.$$
>
> While for the coastline we determined the scaling of its length L depending on the change in l, in the case of R/S analysis we determine the change in the normalized range of the series values within the scale n

4.4.2 Detrended fluctuation analysis

Detrended fluctuation analysis (DFA) [49] is based on the hypothesis that a correlated time series can be mapped to a self-similar process by integration. Thus, measuring the self-similarity properties can indirectly indicate the correlation properties of the series. The advantages of DFA compared to other methods (spectral analysis, R/S analysis) are that it reveals long-term correlations of non-stationary time series and also allows to ignore obvious random correlations that are a consequence of non-stationarity [178].

There are DFAs of different orders that differ in the trends that are extracted from the data.

Let’s consider the lowest order DFA.

1) For a time series of length N, the cumulative sum, $y(k) = \sum_{i=1}^{k}(x_i - \bar{x})$, where x_i is the i-th value of the time series, $\bar{x}$ is its average value, $k = 1, \ldots, N$.
2) The resulting series $y(k)$ is divided into m subsequences (windows) of equal width n and for each subsequence (in each window) the following procedures are performed:
   - the local linear trend of $y_t(k)$ is found using the least squares method;
   - the subsequence is detrended by subtracting the value of the local trend $y_t(k)$ from the values of the series $y(k)$ belonging to the sequence t;
   - the average $\bar{y}_t$ of the detrended values is found.

For the values obtained in this way, all the subsequences are

$$F_n = \sqrt{\bar{y}_t \cdot m^{-1}},$$

where n is the number of points in the subsequence (window width), m is the number of subsequences, and $\bar{y}_t$ is the average of the detrended values for subsequence t.

The above procedure is repeated for different values of n, resulting in a set of dependencies F_n on n. Plotting $\log F(n)$ versus $\log n$ and interpolating the obtained values with a regression line allows us to calculate the scaling index α, which is the coefficient of the slope of the interpolation line and characterizes the change in the correlations of fluctuations in the time series F_n with an increase in the time interval n.

Compared to R/S analysis, DFA provides more opportunities for interpreting the scaling factor α:

- for a random series (mixed or "surrogate") $\alpha = 0.5$;
- in the presence of only short-term correlations, α may differ from 0.5, but tends to tend to 0.5 with increasing window size;

- a value of $0.5 < \alpha \leq 1.0$ indicates persistent long-term correlations that follow a power law;
- $0 < \alpha < 0.5$ indicates an anti-persistent series;
- the special case when $\alpha = 1$ means the presence of $1/f$ noise.
- for cases when $\alpha \geq 1$, correlations exist, but no longer reflect power law;
- the case of $\alpha = 1.5$ indicates Brownian noise, which represents integrated white noise.

In the case of power law dependence of the autocorrelation function, there is a decrease in autocorrelation with γ:

$$C(L) \sim L^{-\gamma}.$$

In addition, the spectral density also decreases according to a power law:

$$P(f) \sim f^{-\beta}. \tag{4.5}$$

The corresponding exponents are expressed through the following relations:

- $\gamma = 2 - 2\alpha$;
- $\beta = 2\alpha - 1$.

The second-order DFA (DFA_2) calculates the deviations of $F^2(v,s)$ of the profile from the second-order interpolation polynomial. Thus, the effects of possible linear and parabolic trends for scales larger than those under consideration are removed. In general, the DFA of order n calculates the profile deviations from the interpolation polynomial of the n-th order, which removes the influence of all possible trends of orders up to $(n-1)$ for scales larger than the window size.

Then, the nearest polynomial $y_v(s)$ for the profile at each of the $2N_s$ segments v is calculated and the deviation is determined

$$F^2(v,s) \equiv \frac{1}{s}\sum_{i=1}^{s}\left(x_{(v-1)s+i} - y_i(i)\right)^2. \tag{4.6}$$

Next, we find the average value of the fluctuations of all detrended profiles:

$$F_2(s) \equiv \sqrt{\left(\frac{1}{2N_s}\sum_{v=1}^{2N_s} F^2(v,s)\right)}. \quad (4.7)$$

The value of Eq. (4.7) can be interpreted as the root mean square displacement (movement) of the point of random walks in the chain after s steps.

4.4.3 Higuchi fractal dimension

The **Higuchi fractal dimension** [36, 157] is a type of monofractal dimension defined as follows:

Suppose that we have a time series $x(1), x(2), \ldots, x(N)$ and a reconstructed time series $x_m^k = \{x(m), x(m+k), x(m+2k), \ldots, x(m+[(N-m)/k]\cdot k)\}$ for $m = 1, 2, \ldots, k$, where m represents the original time; $k = 2, \ldots, k_{max}$ represents the degree of time shift. The notation $[\cdot]$ represents the integer part of x. For each reconstructed time series x_m^k, the average length of the time sequence $L_m(k)$ is calculated:

$$L_m(k) = \frac{\sum_{i=1}^{[(N-m)/k]} \left|x(m+ik) - x\left(m+(i-1)\cdot k\right)\right| \cdot (N-1)}{[(N-m)/k]\cdot k}.$$

Then, for all average lengths $L_m(k)$, the general average $L(k) = (k)^{-1}\sum_{m=1}^{k} L_m(k)$ is found. According to Higuchi's method, the generalized average value of $L(k)$ is proportional to the scale k, i.e. $L(k) \propto k^{-D}$. Next, we find a logarithm of both sides and obtain the equation $\ln L(k) \propto D \cdot \ln(1/k)$. By interpolating the regression line through the dependence of $\ln L(k)$ on $\ln(1/k)$, we can obtain the fractality index D as the slope of this line. The index D will represent the Higuchi fractal dimension.

4.4.4 Petrosian fractal dimension

First, for the time series $\{x_1, x_2, \ldots, y_N\}$, we create its discretized (binary) version, z_i:

$$z_i = \begin{cases} 1, & x_i > \langle x \rangle, \\ -1, & x_i \le \langle x \rangle. \end{cases}$$

The **Petrosian fractal dimension** [20, 37, 135] can be defined as

$$D = \log_{10} N / \left(\log_{10} + \log_{10} [N/(N + 0.4N_\Delta)] \right),$$

where N_Δ is the number of total changes in the sign of z_i : $N_\Delta = \sum_{i=1}^{N-2} |(z_{i+1} - z_i)/2|$.

4.4.5 Katz fractal dimension

Suppose that the signal consists of a pair of points (x_i, y_i). Then, the **Katz fractal dimension** [109] is defined as

$$D = \log N / (\log N + \log[d/L]),$$

where $L = \sum_{i=0}^{N-2} \sqrt{(y_{i+1} - y_i)^2 + (x_{i+1} - x_i)^2}$, and the value of $d = \max\left(\sqrt{(x_i - x_1)^2 - (y_i - y_1)^2}\right)$.

4.4.6 Sevcik fractal dimension

First, for the set of values (x_i, y_i) , normalization is performed: $x_i^* = (x_i - x_{min})/(x_{max} - x_{min})$ and $y_i^* = (y_i - y_{min})/(y_{max} - y_{min})$.

The **Sevcik fractal dimension** [40] can be defined as

$$D = 1 + \ln L / \ln(2 \cdot [N - 1]),$$

L is the length of the signal, which can be calculated by the formula $L = \sum_{i=0}^{N-2} \sqrt{(y_{i+1}^* - y_i^*)^2 + (x_{i+1}^* - x_i^*)^2}$.

4.4.7 Fractal dimension via normalized length density

This indicator is calculated as follows [7]:

1) For the time series $\{x_1, x_2, \ldots, x_n\}$, standardization is performed: $y_i = (x_i - \mu)/\sigma$, where μ is the mean value of the series, σ is the standard deviation.

2) The normalized length density is calculated $NLD = N^{-1}\sum_{i=2}^{N}|y_i - y_{i-1}|$. The actual calculation of the **fractal dimension via Normalized Length Density** (FNLD) is based on the construction of a monotonic calibration curve $D = f(NLD)$ using a set of Weierstrass functions for which the values of D are set theoretically.
3) For computational purposes, two models of this relationship have been created:
   - logarithmic model: $D = a \cdot \log(NLD - NLD_0) + C$;
   - a power law model: $D = a \cdot (NLD - NLD_0)^k$. The `neurokit2` Python library uses the power law model. The parameters $a = 1.9079$, $k = 0.18383$, and $NLD_0 = 0.097178$, according to [7].

4.4.8 Fractal dimension and power spectral density

The fractal dimension can be calculated based on the analysis of the **power spectral density slope** (PSD) [62, 136] in signals characterized by a power law frequency dependence.

First, the time series is transformed to the frequency domain and then the signal is divided into harmonic oscillations of a certain amplitude, which are "added together" to represent the original signal. If there is a systematic relationship between the frequencies in the signal and the power of these frequencies, then in logarithmic coordinates this is manifested through a linear relationship. The slope of the regression line is taken as an estimate of the fractal dimension.

A slope of 0 corresponds to white noise, and a slope less than 0 but greater than -1 corresponds to pink noise, i.e. $1/f$ noise. Spectral slopes steeper than -2 indicate fractional Brownian motion, which is a manifestation of random walk processes.

4.4.9 Correlation dimension

The **correlation dimension** (D_2) is a derivative of the correlation integral (correlation sum) and can be represented as [122-124]:

$$C(\varepsilon) = \frac{1}{N^2} \sum_{\substack{i,j=1 \\ i \neq j}}^{N} \Theta\left(\varepsilon - \| \vec{x}(i) - \vec{x}(j) \|\right), \ \vec{x}(i) \in \Re^m.$$

The correlation dimension can be derived from the following power law:

$$C(\varepsilon) \sim \varepsilon^{\nu},$$

or

$$D_2 = \lim_{M \to \infty} \lim_{\varepsilon \to 0} \log(g_\varepsilon / N^2) / \log \varepsilon,$$

where g_ε is the total number of pairs of points whose distance is less than the radius ε.

In the first case, we select the i-th trajectory and all other j-th trajectories, and see if the j-th trajectories fall within the ε-neighborhood of the i-th trajectory. If the distance between them does not exceed the circle of radius ε, we set 1. But if the distance between the trajectories is greater than ε, then we set 0. Then all this is summed up, divided by the total number of trajectories. In essence, the correlation integral is the average probability that the two trajectories in the phase space under consideration will be close enough to each other. The closer the points of the phase space are located, the higher the value of the correlation integral. The more equidistant the trajectories appear from each other, the closer the value of the correlation integral is to zero.

We can find the value of the correlation dimension similarly to the previous fractal indicators: we look for the dependence of the correlation integral on the value of ε. This dependence is plotted on a logarithmic scale.

Here are some interesting examples.

Electrocardiogram (ECG): ECG signals reflect the electrical activity of the heart. The complexity of an ECG signal can be estimated using the correlation dimension. The correlation dimension of a healthy heart ECG is expected to be

higher due to the presence of complex patterns and variability. On the other hand, abnormal ECG signals, for example, from patients with arrhythmias or heart disease, may have a lower correlation dimension due to the loss of signal complexity.

Electroencephalogram (EEG): EEG signals record the electrical activity of the brain. The correlation dimension can be used to analyze the complexity of brain activity, which can vary with different cognitive states, sleep stages, or neurological disorders. In healthy individuals, EEG signals during wakefulness and attention may have a higher correlation dimension compared to signals during sleep, when brain activity is more regular and synchronized.

Respiratory signals: Respiratory signals, such as respiratory rate or airflow, can also be analyzed using correlation dimensionality. The complexity of these signals can vary depending on factors such as stress, exercise, or the presence of respiratory disease. Normal breathing may have a higher correlation dimension, while abnormalities in breathing signals, such as obstructive sleep apnea or respiratory disorders, may result in a lower correlation dimension.

Gait analysis: The correlation dimension can be used to analyze gait patterns. It helps to understand the complexity of a person's movements while walking or running. Changes in the correlation dimension of gait signals can indicate changes in gait stability or gait abnormalities caused by neurological or musculoskeletal conditions.

Heart rate variability (HRV): HRV is a change in the time intervals between successive heartbeats. It is influenced by the autonomic nervous system and reflects the adaptability and complexity of the cardiovascular system. A higher HRV level, corresponding to a higher correlation dimension, is usually associated with a better state of the cardiovascular system and its adaptability to physiological and environmental changes. Its decline may be associated with abnormal cardiac dynamics.

DNA sequences: The correlation dimension can also be used in the analysis of DNA sequences. It helps to identify self-similar or fractal patterns within sequences, which can be important for understanding genetic complexity, evolutionary relationships, and gene regulation. High correlation dimensionality means high complexity of the DNA strand. Low correlation dimension – simplified DNA strand.

Financial markets: Higher correlation dimensionality in financial market time series data indicates greater complexity and the existence of underlying self-similar models or fractal structures. Chaotic behavior of stock prices may be associated with periods of high volatility and unpredictability. On the other hand, a lower value of the correlation dimension may indicate more predictable and less complex price movements, which corresponds to periods of stability or less volatile market conditions.

4.5 Practical estimations of monofractal indicators

Let's consider how these indicators can be used as indicators of crisis states. First, import the necessary libraries:

```
import matplotlib.pyplot as plt
import numpy as np
import neurokit2 as nk
import yfinance as yf
import pandas as pd
import scienceplots
from tqdm import tqdm

%matplotlib inline
```

Next, let's configure the format for displaying figures:

```
plt.style.use(['science', 'notebook', 'grid'])

size = 22
params = {
'figure.figsize': (8, 6),
'font.size': size,
'lines.linewidth': 2,
'axes.titlesize': 'small',
'axes.labelsize': size,
'legend.fontsize': size,
'xtick.labelsize': size,
'ytick.labelsize': size,
```

```
"font.family": "Serif",
"font.serif": ["Times New Roman"],
'savefig.dpi': 300,
'axes.grid': False
}

plt.rcParams.update(params)
```

Let's define the `transformation()` function to standardize the series:

```
def transformation(signal, ret_type):

    for_rec = signal.copy()

    if ret_type == 1:
       pass
    elif ret_type == 2:
        for_rec = for_rec.diff()
    elif ret_type == 3:
        for_rec = for_rec.pct_change()
    elif ret_type == 4:
        for_rec = for_rec.pct_change()
        for_rec -= for_rec.mean()
        for_rec /= for_rec.std()
    elif ret_type == 5:
        for_rec = for_rec.pct_change()
        for_rec -= for_rec.mean()
        for_rec /= for_rec.std()
        for_rec = for_rec.abs()
    elif ret_type == 6:
        for_rec -= for_rec.mean()
        for_rec /= for_rec.std()

    for_rec = for_rec.dropna().values

    return for_rec
```

4.5.1 Calculation of the Hurst exponent using R/S analysis

For further calculations, we will use the `neurokit2` and `fathon` libraries. The second of them can be installed as follows:

```
!pip install fathon
```

Next, import the library itself and related modules:

```
import fathon
from fathon import fathonUtils as fu
```

The `neurokit2` library contains the necessary method for *R/S* analysis — `fractal_hurst`. Its syntax:

```
fractal_hurst(signal, scale='default', corrected=True, show=False)
```

Parameters:

- **signal** (*Union[list, np.array, pd.Series]*) – the signal (i.e., a time series) in the form of a vector of values or dataframe;
- **scale** (*list*) – a list containing the lengths of the windows (number of data points in each subseries) that the signal is divided into;
- **corrected** (*boolean*) – if True, the Anis-Lloyd-Peters correction factor will be applied to the output according to the expected value for the individual (R/S) values;
- **show** (*bool*) – if True, returns a plot.

Returns:

- **h** (*float*) – Hurst exponent;
- **kwargs** – a dictionary containing information regarding the parameters used in the procedure.

Let's consider the degree of trend stability in the dynamics of the gold price using the entire time series. Next, let's find the value of the Hurst exponent using the sliding window procedure.

4.5.1.1 Calculations of R/S analysis for the whole time series

First of all, we will find the value of profitability for our series and standardize them. After that, we will perform the calculations:

```
signal = time_ser.copy()
ret_type = 4 # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

for_rs = transformation(signal, ret_type)
```

Now, let's perform R/S analysis (see Fig. 4.2):

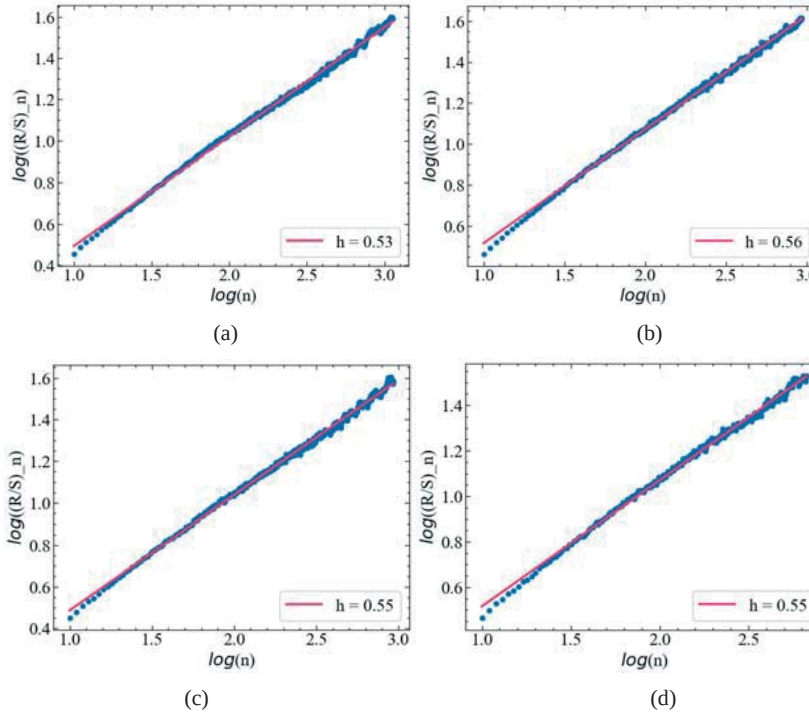


Fig. 4.2: Dependence of R/S values on scaling for S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d)

As we can see from Fig. 4.2, the value of *h* for the studied stock indices close to 0.5, which indicates the similarity of their dynamics to a random walk. But as the laws governing the market change over time, so must the correlations within the system, and therefore the Hurst exponent can also change.

4.5.1.2 Sliding window procedure for R/S analysis

Let's define a function for constructing paired charts:

```
def plot_pair(x_values,
              y1_values,
              y2_values,
              y1_label,
              y2_label,
              x_label,
              file_name, clr="magenta"):
```

```
    fig, ax = plt.subplots()

    ax2 = ax.twinx()
    ax2.spines.right.set_position(("axes", 1.03))

    p1, = ax.plot(x_values,
                  y1_values,
                  "b-", label=fr"{y1_label}")
    p2, = ax2.plot(x_values,
                   y2_values,
                   color=clr,
                   label=y2_label)

    ax.set_xlabel(x_label)
    ax.set_ylabel(f"{y1_label}")
    ax.yaxis.label.set_color(p1.get_color())
    ax2.yaxis.label.set_color(p2.get_color())

    tkw = dict(size=2, width=1.5)

    ax.tick_params(axis='x', rotation=45, **tkw)
    ax.tick_params(axis='y', colors=p1.get_color(), **tkw)
    ax2.tick_params(axis='y', colors=p2.get_color(), **tkw)
    ax2.legend(handles=[p1, p2])

    plt.savefig(file_name +".jpg")

    plt.show();
```

Let's proceed to the sliding window procedure:

```
ret_type = 4 # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series
window = 500 # sliding window width
tstep = 1    # sliding window step
length = len(time_ser.values)  # length of a series
corr = False # Anis-Lloyd-Peters correction factor

H = []                          # array for values of Hurst exponent

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

# calculate hurst exponent
    h, _ = nk.fractal_hurst(fragm, corrected=corr, show=False)

    H.append(h)
```

```
np.savetxt(f"rs_hurst_name={symbol}_window={window}_step={tstep}_ \
           rettype={ret_type}_corrected={corr}.txt" , H)
```

Visualize the result:

```
measure_label = r'$H$'
file_name = f"rs_hurst_name={symbol}_window={window}_step={tstep}_ \
            rettype={ret_type}_corrected={corr}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          H,
          ylabel,
          measure_label,
          xlabel,
          file_name)
```

Fig. 4.3 represents the visualization of the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their Hurst exponent.

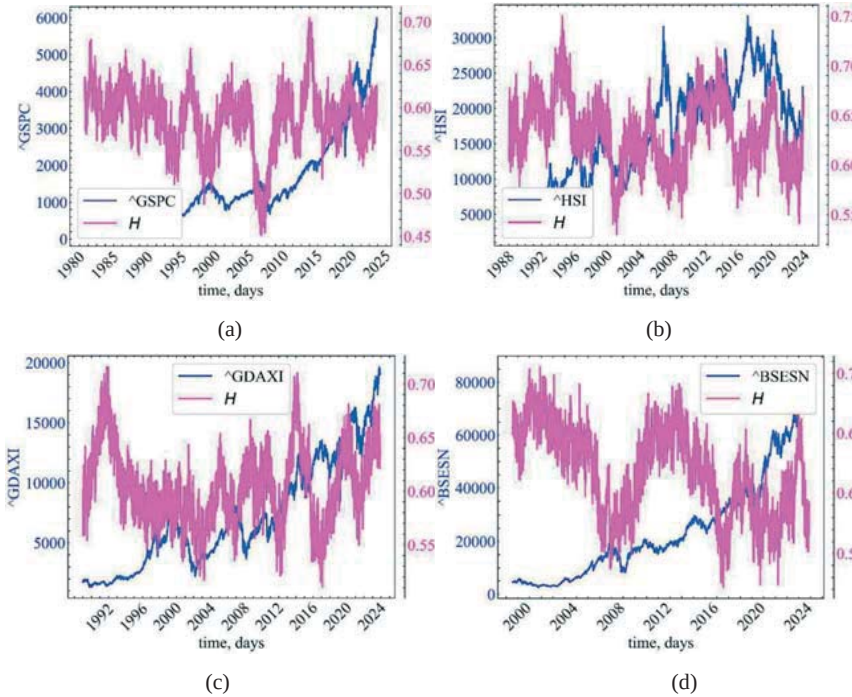


Fig. 4.3: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their Hurst exponent

In Fig. 4.3, we can see that the Hurst exponent increases in the pre-crisis period and decreases during the crisis. Before the crisis, market dynamics are characterized by an increase in trend resistance (persistence), which reflects the growing correlation of traders' actions.

4.5.2 DFA-based calculations

The `fathon` library provides tools for both the classical detrended fluctuation analysis and its multifractal analog.

4.5.2.1 DFA-based calculations for the whole time series

First, let's present the value of α for the entire series. The calculation procedure based on the `fathon` library will look in the following way:

Find the standardized returns of the series:

```
signal = time_ser.copy()
ret_type = 4 # type of a series

for_dfa = transformation(signal, ret_type)

cumulat = fu.toAggregated(for_dfa) # find cumulative values

rev = True # whether to repeat the calculation of the fluctuation function fr
om the end
order = 2  # order of the local trend

pydfa = fathon.DFA(cumulat) # initializing the DFA object to perform further
calculations

win_beg = 100   # initial width of the segments
win_end = 2000  # final width of the segments

wins = fu.linRangeByStep(win_beg, win_end) # generate an array of linearly se
parated elements.

n, F = pydfa.computeFlucVec(wins,
                            polOrd=order,
                            revSeg=rev)    # calculate the fluctuation functi
on

H, H_intercept = pydfa.fitFlucVec()        # calculate alpha exponent
```

We derive the dependence of the fluctuation function on the scale:

```
polyfit = np.polyfit(np.log(n), np.log(F), 1)
fluctfit = np.exp(1)**np.polyval(polyfit, np.log(n))
```

We plot the dependence of the fluctuation function on the scale in a logarithmic scale (see Fig. 4.4):

```
fig, ax = plt.subplots()
fig.suptitle("DFA-based Hurst exponent")

ax.scatter(np.log(n), np.log(F), marker="o", zorder=1, label="_no_legend_")

label = fr"$\alpha$ = {H:.2f}"
ax.plot(np.log(n), np.log(fluctfit), color="#E91E63", zorder=2, linewidth=3,
label=label)

ax.set_ylabel(r'$\ln{F_{2}(n)}$')
ax.set_xlabel(r'$\ln{n}$')

ax.legend(loc="lower right")

plt.show()
```

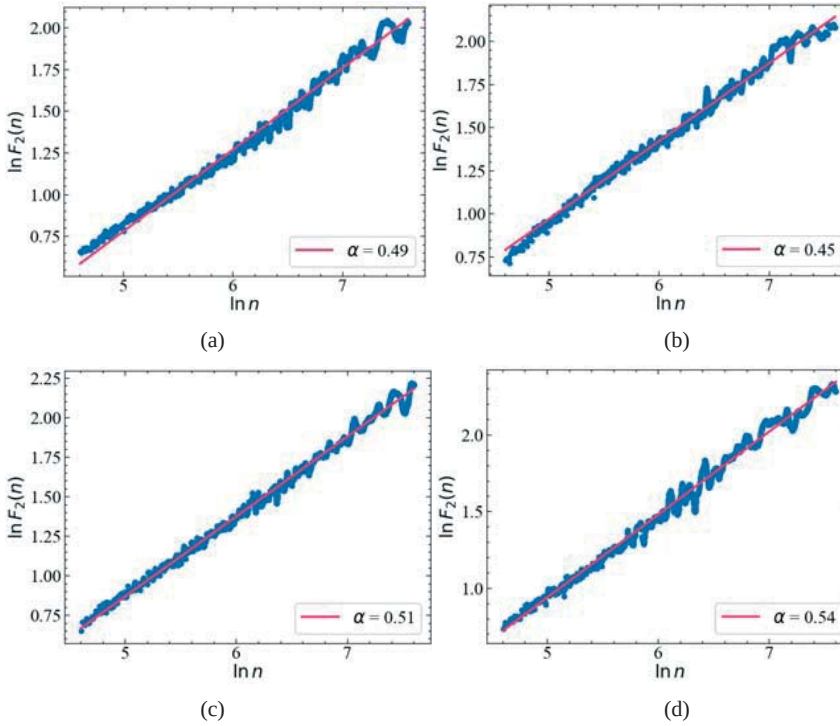


Fig. 4.4: Logarithmic dependence of the fluctuation function values on scaling for S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

The DFA procedure shows that the values of stock indices appear to be rather anti-persistent, but the result presented is quite close to the one obtained using the R/S analysis. Let's consider the values within the sliding window algorithm.

4.5.2.2 DFA-based calculations within the sliding window procedure

Let's define the following parameters:

```
window = 500 # sliding window width
tstep = 1    # sliding window step
ret_type = 4 # type of a series:

rev = True # whether to repeat the calculation of the fluctuation function fr
om the end
order = 2 # order of the polynomial trend

periods = 1

win_beg = 10             # initial scale of segments
win_end = window-1       # the final scale of the segments



length = len(time_ser.values) # time series length

alpha = []               # an array of alpha (Hurst) indicators
D_f = []                 # fractal dimension
beta = []                # spectral density indicator
gamma = []               # autocorrelation indicator
```

Let's find the Hurst exponent (α), the fractal dimension (D_f), the spectral density index (β), and the autocorrelation index (γ):

```
for i in tqdm(range(0, length-window, tstep)):


    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

# calculate cumulative values
    cumulat_wind = fu.toAggregated(fragm)

# initializing the DFA object
    pydfa = fathon.DFA(cumulat_wind)

# generate an array of linearly separated elements
    wins = fu.linRangeByStep(win_beg, win_end)
```

```
# find the fluctuation function
    n, F_wind = pydfa.computeFlucVec(wins, polOrd=order, revSeg=rev)

# find the alpha exponent
    H_wind, _ = pydfa.fitFlucVec()

# find the fractal dimension
    D = 2. - H_wind

# find spectral density indicator
    bi = 2. * H_wind - 1

# find autocorrelation indicator
    gi = 2. - 2. * H_wind

    alpha.append(H_wind)
    D_f.append(D)
    beta.append(bi)
    gamma.append(gi)
```

Save absolute values of indicators to text files:

```
np.savetxt(f"alpha_{symbol}_{window}_{tstep}_ \
{ret_type}_{order}_{win_beg}_{win_end}.txt", alpha)
np.savetxt(f"D_f_{symbol}_{window}_{tstep}_ \
{ret_type}_{order}_{win_beg}_{win_end}.txt", D_f)
np.savetxt(f"beta_{symbol}_{window}_{tstep}_ \
{ret_type}_{order}_{win_beg}_{win_end}.txt", beta)
np.savetxt(f"gamma_{symbol}_{window}_{tstep}_ \
{ret_type}_{order}_{win_beg}_{win_end}.txt", gamma)
```

Define labels for figures and names of saved figures:

```
label_alpha = fr'$\alpha$'
label_d = fr'$D_f$'
label_beta = fr'$\beta$'
label_gamma = fr'$\gamma$'

file_name_alpha = f"alpha_{symbol}_{window}_{tstep}_ \
{ret_type}_{order}_{win_beg}_{win_end}"
file_name_d = f"D_f_{symbol}_{window}_{tstep}_ \
{ret_type}_{order}_{win_beg}_{win_end}"
file_name_beta = f"beta_{symbol}_{window}_{tstep}_ \
{ret_type}_{order}_{win_beg}_{win_end}"
file_name_gamma = f"gamma_{symbol}_{window}_{tstep}_ \
{ret_type}_{order}_{win_beg}_{win_end}"
```

Let’s display the results:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          alpha,
          ylabel,
          label_alpha,
          xlabel,
          file_name_alpha)
```

Fig 4.5 represents the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their α exponent.

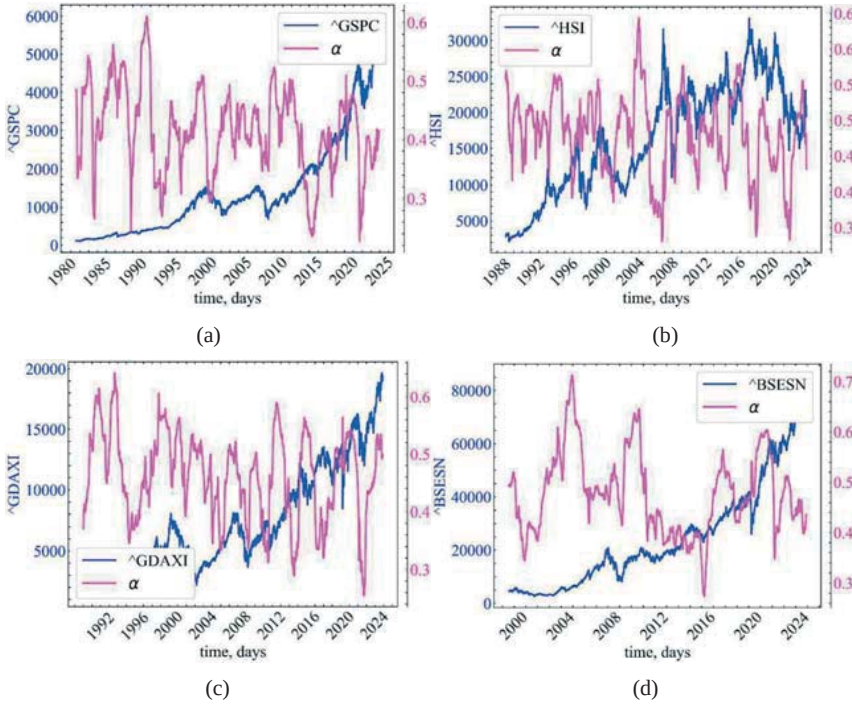


Fig. 4.5: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their α exponent

When compared with R/S analysis, Fig. 4.5 shows that the DFA dynamics of the generalized Hurst exponent is much more stable. We are now able to differentiate a significant proportion of the crash events that took place in the gold market. The generalized Hurst exponent shows that pre-crisis phenomena are characterized by an increase in the trend stability of the market, an increase in the degree of self-organization of the system.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          D_f,
          ylabel,
          label_d,
```

```
          xlabel,
          file_name_d)
```

Fig. 4.6 shows the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their fractal dimension D_f.

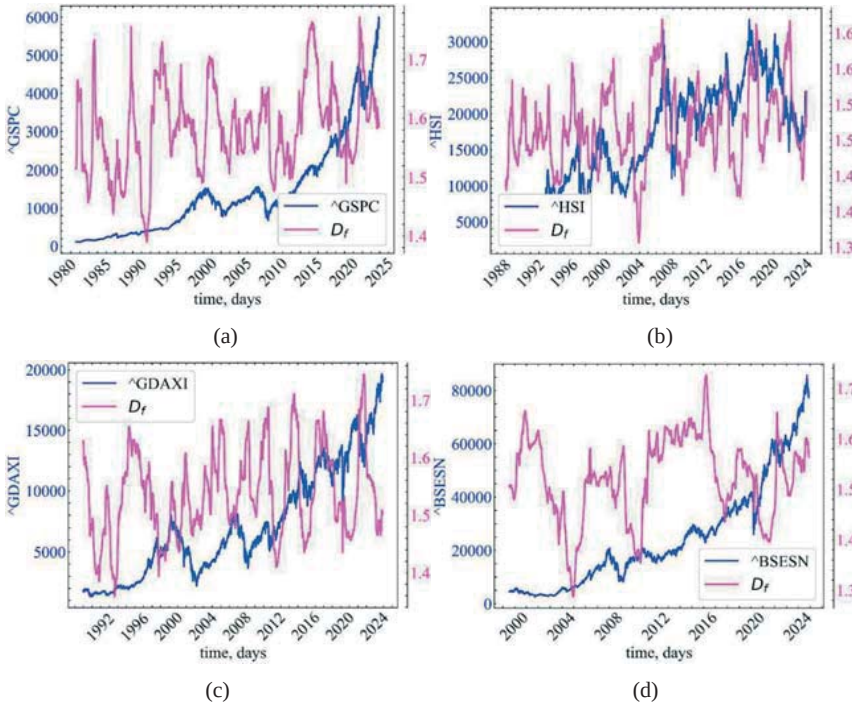


Fig. 4.6: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their fractal dimension D_f

Fig. 4.6 shows that D_f is characterized by a decline in times of crisis. This is an indicator that a higher degree of market organization is reflected in smoother or less volatile fluctuations of the signal under study.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          beta,
          ylabel,
          label_beta,
          xlabel,
          file_name_beta)
```

Fig. 4.7 shows the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their spectral density index β.

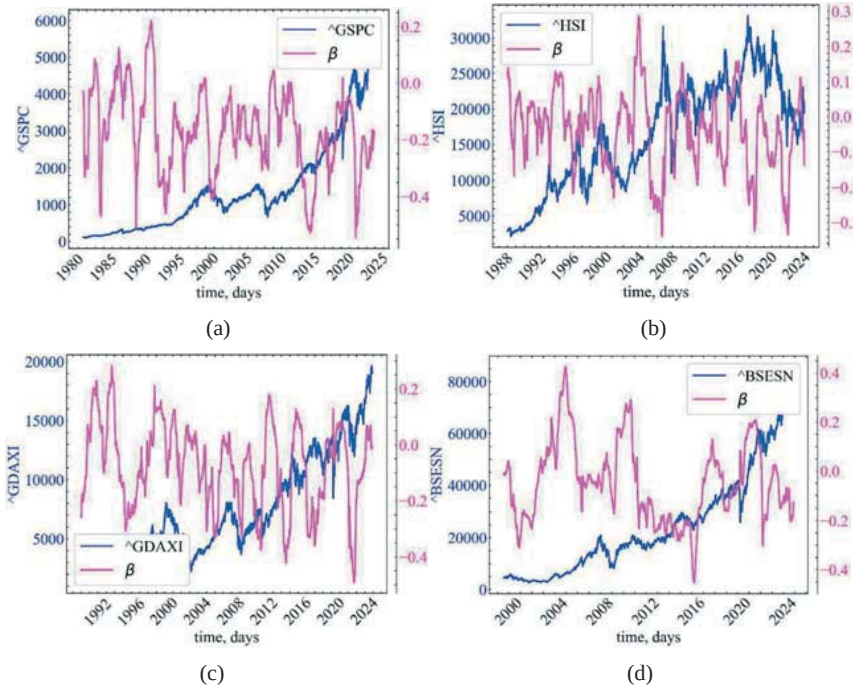


Fig. 4.7: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their spectral density index β

In Fig 4.7, the spectral power density increases during crisis periods, which indicates a decrease in signal power at a unit frequency interval. This is also evidence of an increase in the correlation properties of the system.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          gamma,
          ylabel,
          label_gamma,
          xlabel,
          file_name_gamma)
```

Fig. 4.8 provides the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their autocorrelation index.

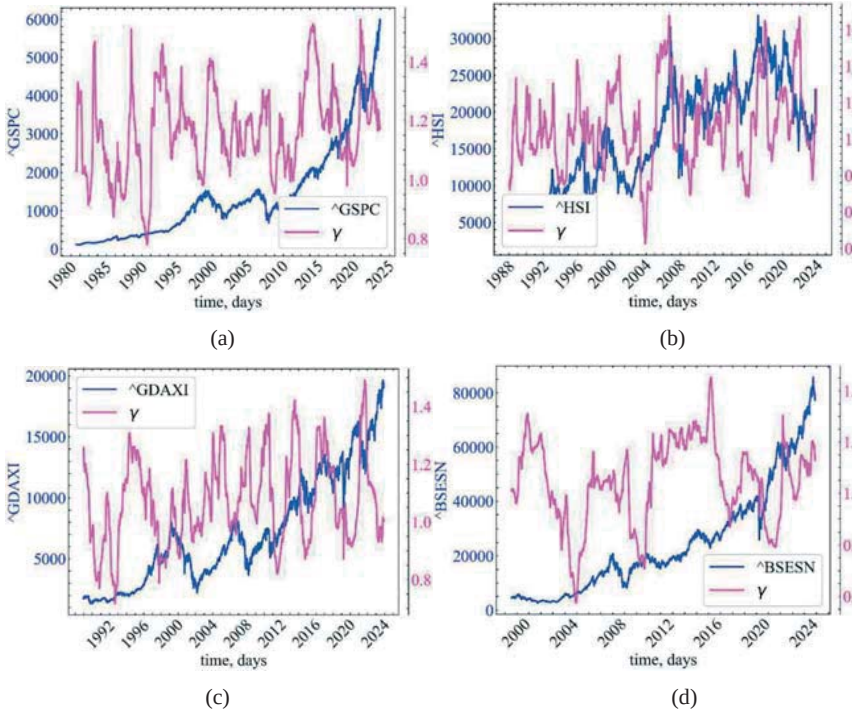


Fig. 4.8: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their autocorrelation index

Fig. 4.8 shows that the γ indicator decreases in crisis and pre-crisis periods. This is an indicator of a slowdown in the decline of the autocorrelation function, which in turn also indicates an increase in the correlation of the system's dynamics.

4.5.3 Calculating the Higuchi fractal dimension

As already mentioned, the Higuchi fractal dimension is a type of fractal dimension for time series. It is calculated by reconstructing a number of new k_{max} data sets. For each reconstructed data set, the length of the curve is calculated and plotted against the corresponding k_{max}-value on a logarithmic scale. The HFD corresponds to the slope of a linear trend using the least squares method.

We calculate the optimal k value for the entire time series. The `neurokit2` library provides a ready-made procedure for automated selection of this parameter. The optimal k_{max} value is calculated based on the point at which the fractal dimension reaches a plateau for a range of k_{max} values [83].

The syntax of this function is as follows:

```
complexity_k(signal, k_max='max', show=False)
```

Parameters:

- **signal** (*Union[list, np.array, pd.Series]*) – the signal (i.e., a time series) in the form of a vector of values;
- **k_max** (*Union[int, str, list], optional*) – maximum number of interval times (should be greater than or equal to 3) to be tested. If max, it selects the maximum possible value corresponding to half the length of the signal;
- **show** (*bool*) – visualize the slope of the curve for the selected k_{max} value.

Returns:

- **k** (*float*) – the optimal k_{max} of the time series;
- **info** (*dict*) – a dictionary containing additional information regarding the parameters used to compute optimal k_{max}.

4.5.3.1 Higuchi fractal dimension for the whole time series

For further calculations, let's first perform a series transformation. We will use the original time series for further calculations:

```
signal = time_ser.copy()
ret_type = 1      # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

for_higuchi = transformation(signal, ret_type)
```

And now we'll get the optimal value according to the above procedure (see Fig. 4.9):

```
k_max, info = nk.complexity_k(for_higuchi, k_max=100, show=True)
```

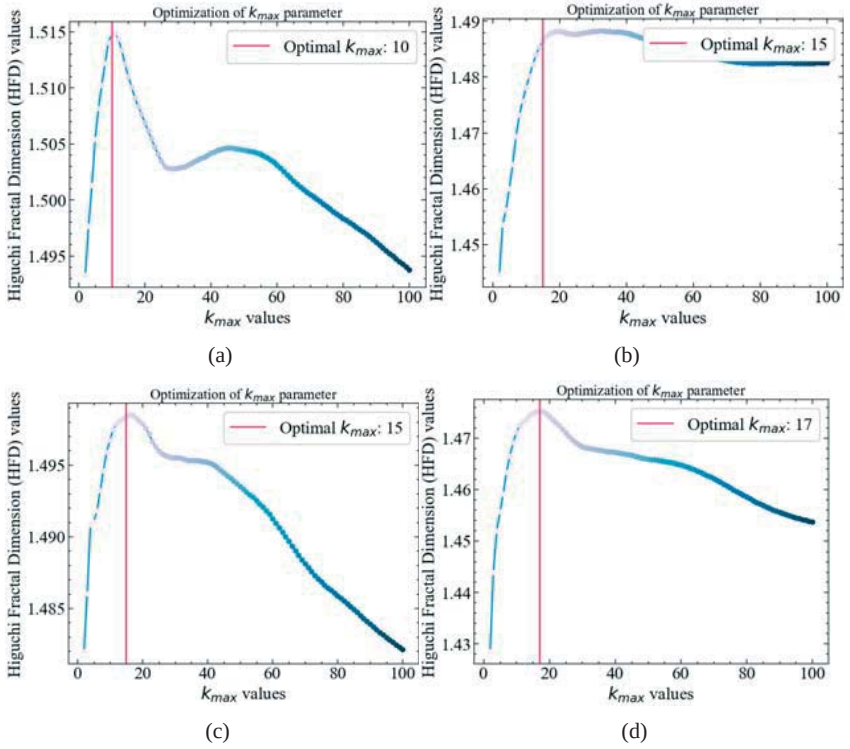


Fig. 4.9: Dependence of the Higuchi fractal dimension on the range of k_{max} values for the whole time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

Now let's plot the dependence of the signal length on the time offset on a logarithmic scale. For a fractal signal, a linear relationship should hold. The `neurokit2` library contains a method for calculating this fractal dimension. The syntax of this procedure is as follows:

```
fractal_higuchi(signal, k_max='default', show=False, **kwargs)
```

Parameters:

- **signal** (*Union[list, np.array, pd.Series]*) – the signal (i.e., a time series) in the form of a vector of values;

- **k_max** (*str or int*) – maximum number of interval times (should be greater than or equal to 2);
- **show** (*bool*) – visualize the slope of the curve for the selected k_{max} value.

Returns:

- **HFD** (*float*) – Higuchi fractal dimension of the time series;
- **info** (*dict*) – a dictionary containing additional information regarding the parameters used to compute Higuchi fractal dimension.

```
hfd, info = nk.fractal_higuchi(for_higuchi, k_max=k_max, show=True)
```

In Fig. 4.10 is presented the dependence of signal length on time delay for the whole time series of S&P 500, Hang Seng index, DAX, and BSE Sensex

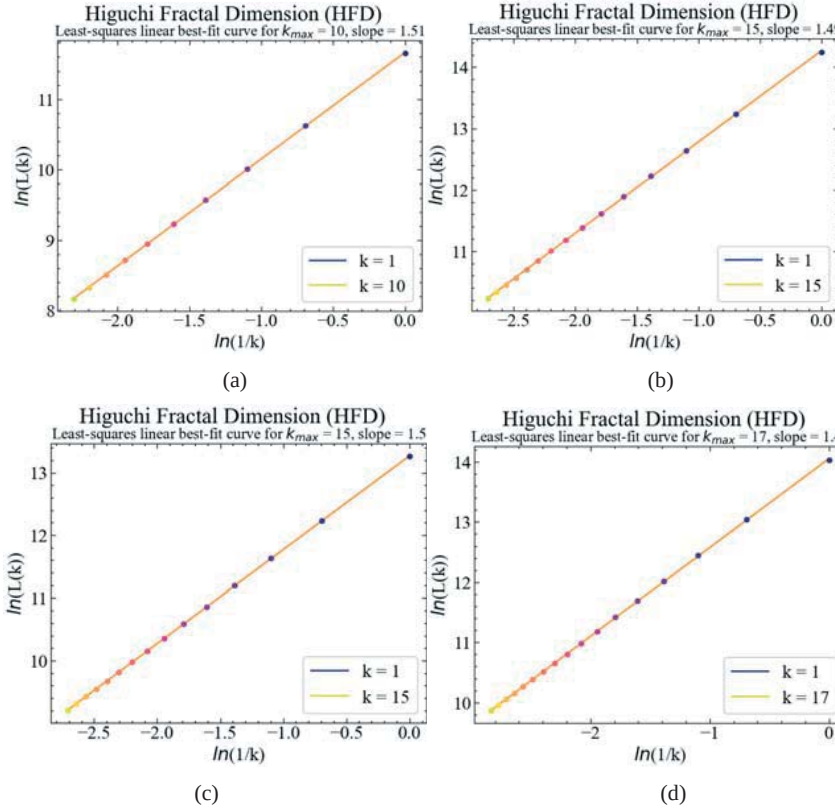


Fig. 4.10: Dependence of signal length on time delay for the whole time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

In the following, we will use the obtained optimal value to calculate the Higuchi fractal dimension within the sliding window algorithm.

4.5.3.2 Calculations of Higuchi fractal dimension within the sliding window algorithm

Let's consider the following parameters:

```
window = 500 # window length
tstep = 1    # time step
ret_type = 1 # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

k_max_wind = 30                    # maximum time delay

length = len(time_ser.values)      # series length

hfd_wind = []                      # array of Higuchi dimensions
```

and start the sliding window procedure:

```
for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()

    fragm = transformation(fragm, ret_type)

# calculate the Higuchi fractal dimension
    higuchi, _ = nk.fractal_higuchi(fragm,
                                    k_max=k_max_wind,
                                    show=False)

# save the result to an array of values
    hfd_wind.append(higuchi)
```

Save the initial values to a text file:

```
np.savetxt(f"fd_higuchi_name={symbol}_kmax={k_max_wind}_\
           wind={window}_step={tstep}.txt", hfd_wind)
```

Define the labels for the figures and the titles of the saved figures:

```
label_higuchi = fr'$HFD$'

file_name_higuchi = f"fd_higuchi_name={symbol}_kmax={k_max_wind}_\
           wind={window}_step={tstep}"
```

Display the result:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          hfd_wind,
          ylabel,
          label_higuchi,
          xlabel,
          file_name_higuchi)
```

Fig. 4.11 represents the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their Higuchi fractal dimension.

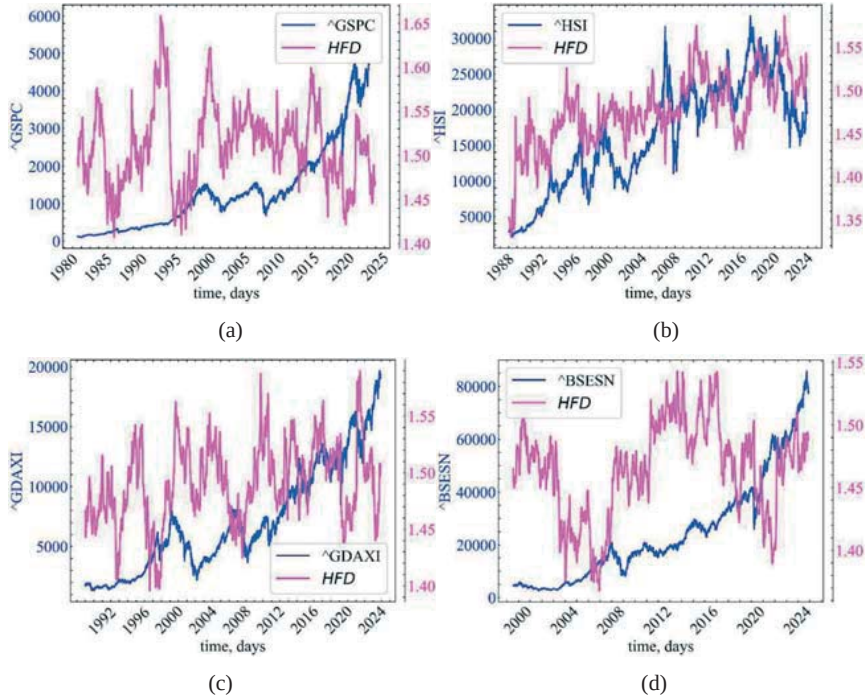


Fig. 4.11: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their Higuchi fractal dimension

As can be seen from the figure, the Higuchi fractal dimension can serve as an indicator or precursor of crisis phenomena. It can be seen that this indicator begins to decline in pre-crisis periods or at the very moment of the crisis,

indicating an increase in the smoothness of the system's dynamics, the degree of correlation and the trend stability of market dynamics.

4.5.4 Calculating the Petrosian fractal dimension

Petrosian [20] proposed a fast method for estimating fractal dimensionality by converting a signal into a binary sequence from which the fractal dimensionality is estimated. There are several variations of the algorithm (`neurokit2`, for example, offers options `"A"`, `"B"`, `"C"`, or `"D"`), which differ primarily in the way the discrete (symbolic) sequence is created (see `complexity_symbolize()` for details). The most common method (`"C"`, by default) binarizes the signal by the sign of consecutive differences.

Most of these sampling methods assume that the signal is periodic (without a linear trend). To remove linear trends, linear detrending can be useful.

The syntax of this procedure is as follows:

```
fractal_petrosian(signal, symbolize='C', show=False)
```

Parameters:

- **signal** (*Union[list, np.array, pd.Series]*) – the signal (i.e., a time series) in the form of a vector of values;
- **symbolize** (*str*) – method to convert a continuous signal input into a symbolic (discrete) signal. By default, assigns 0 and 1 to values below and above the mean. Can be None to skip the process (in case the input is already discrete);
- **show** (*bool*) – if True, will show the discrete the signal.

Returns:

- **pfd** (*float*) – the Petrosian fractal dimension (PFD);
- **info** (*dict*) – a dictionary containing additional information regarding the parameters used to compute PFD.

We won't go into detail about the syntax of the `complexity_symbolize()` function. We will describe only those discretization methods that are related to PFD:

- **Method 'A'** binarizes the signal by higher vs. lower values as compated to the signal's mean. Equivalent method="mean" (method="median" is also valid);
- **Method 'B'** uses values that are within the mean +/- 1 SD band vs. values that are outside this band.
- **Method 'C'** computes the difference between consecutive samples and binarizes depending on their sign;
- **Method 'D'** forms separates consecutive samples that exceed 1 signal's SD from the others smaller changes.

Now let's look at the sliding window dynamics of the indicator.

4.5.4.1 Calculations of Petrosian fractal dimension within sliding window procedure

Since most of these discretization methods require detrending the series, we will perform calculations for the gold price returns. We will use the following parameters:

```
window = 500 # sliding window width
tstep = 1 # sliding window time step
ret_type = 4 # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

symb = "B" # type of series discretization

length = len(time_ser.values) # series length

petr_wind = []                # array for Petrosian FD
```

Start the sliding window procedure:

```
for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)
```

```
# calculate the Petrosian fractal dimension
    petrocian, _ = nk.fractal_petrosian(fragm,
                                        symbolize=symb,
                                        show=False)

# save the result to an array of values
    petr_wind.append(petrocian)
```

Save the initial values to a text document:

```
np.savetxt(f"fd_petrosian_name={symbol}_method={symb}_\
           wind={window}_step={tstep}.txt", petr_wind)
```

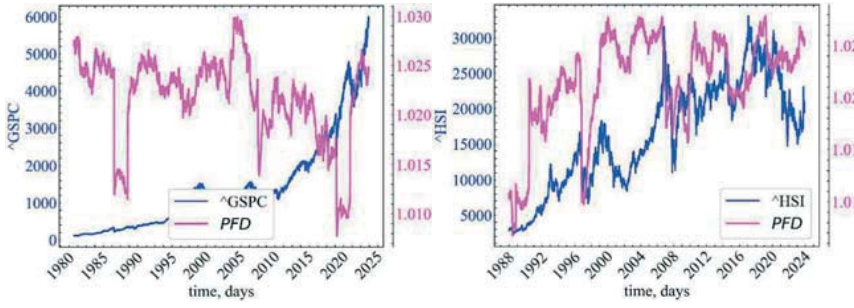
Define labels for figures and titles of saved figures:

```
label_petrocian = fr'$PFD$'

file_name_petrocian = f"fd_petrosian_name={symbol}_method={symb}_\
           wind={window}_step={tstep}"
```

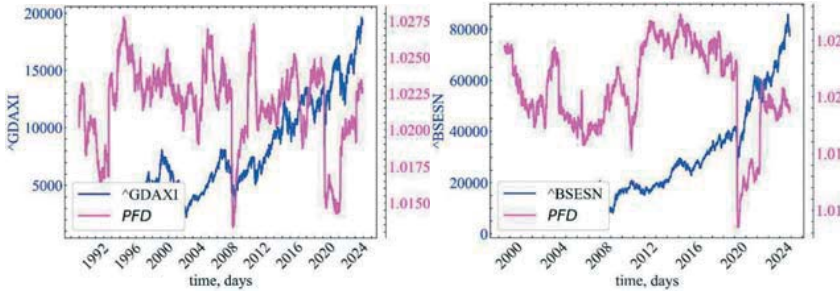
Plot the result:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          petr_wind,
          ylabel,
          label_petrocian,
          xlabel,
          file_name_petrocian)
```

Fig. 4.12 shows the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their Petrosian fractal dimension.



(a) (b)



(c) (d)

Fig. 4.12: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their Petrosian fractal dimension

Fig. 4.12 shows that the Petrosian dimension also decreases during crisis events, indicating an increase in market periodization and synchronization of traders' activity at the relevant moments.

4.5.5 Calculating Katz fractal dimension

Let's calculate the Katz fractal dimension. The Euclidean distances between consecutive signal points are summed and averaged, and the maximum distance between the starting point and any other point in the sample is determined.

The fractal dimension varies from 1.0 for straight lines, to about 1.15 for random walks, and approaches 1.5 for the most "strange" waveforms.

The syntax of the procedure for calculating this dimension is as follows:

```
fractal_katz(signal)
```

Parameters:

- **signal** (*Union[list, np.array, pd.Series]*) – the signal (i.e., a time series) in the form of a vector of values.

Returns:

- **kfd** (*float*) – Katz fractal dimension of the single time series;
- **info** (*dict*) – a dictionary containing additional information (currently empty, but returned nonetheless for consistency with other functions).

4.5.5.1 Calculating Katz fractal dimension within the sliding window procedure

Since this indicator is parameter-independent, we only need the size of the time window, step, and series type:

```
window = 500  # sliding window width
tstep = 1     # sliding window time step
ret_type = 1  # type of a series
# 1 - initial
# 2 - detrending (difference between present and previous values)
# 3 - initial returns
# 4 - standardized returns
# 5 - absolute values (volatility)
# 6 - standardized series
length = len(time_ser.values)

kz_wind = []                        # array for Katz FD
```

Start the sliding window procedure:

```
for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

# calculate Katz FD
    katz, _ = nk.fractal_katz(fragm)

# save results
    kz_wind.append(katz)
```

Save the initial values to a text document:

```
np.savetxt(f"fd_katz_name={symbol}_wind={window}_step={tstep}.txt", kz_wind)
```

Define the labels for the figures and the titles of the saved figures:

```
label_katz = fr'$KFD$'

file_name_katz = f"fd_katz_name={symbol}_wind={window}_step={tstep}"
```

Plot the result:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          kz_wind,
          ylabel,
          label_katz,
          xlabel,
          file_name_katz)
```

In Fig. 4.13 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their Katz fractal dimension.

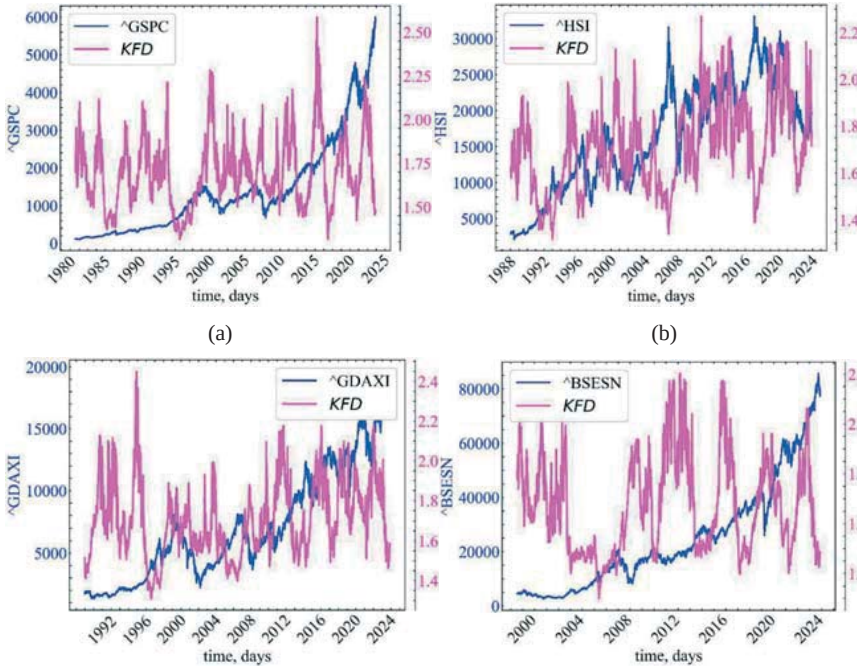


Fig. 4.13: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their Katz fractal dimension

Fig. 4.13 shows that the Katz fractal dimension also decreases in crisis and pre-crisis periods and is also an indicator of the growing degree of correlation of the system during these periods.

4.5.6 Calculating the Sevcik fractal dimension

The algorithm of this fractal dimension was proposed to calculate the fractal dimension of signals by Sevcik [40]. This method can be used to quickly measure the complexity of a signal.

Syntax of the method:

```
fractal_sevcik(signal)
```

Parameters:

- **signal** (*Union[list, np.array, pd.Series]*) – the signal (i.e., a time series) in the form of a vector of values.

Returns:

- **sfd** (*float*) – the sevcik fractal dimension;
- **info** (*dict*) – an empty dictionary returned for consistency with the other complexity functions.

4.5.6.1 Calculating Sevcik fractal dimension within the sliding window procedure

```
window = 500    # sliding window width
tstep = 1       # sliding window time step
ret_type = 1    # type of a series:
# 1 - initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

length = len(time_ser.values)      # series length

sfd_wind = []                      # array for Sevcik FD
```

Start the sliding window procedure:

```
for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

# calculate the fractal dimension of Sevchik
    sevcik, _ = nk.fractal_sevcik(fragm)

# save the result to an array of values
    sfd_wind.append(sevcik)
```

Save the initial values to a text document:

```
np.savetxt(f"fd_cevcik_name={symbol}_wind={window}_step={tstep}.txt", sfd_win
d)
```

Define the labels for the figures and the titles of the saved figures:

```
label_sevcik = fr'$SFD$'

file_name_sevcik = f"fd_cevcik_name={symbol}_wind={window}_step={tstep}"
```

Plot the results:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
```

```
        sfd_wind,
        ylabel,
        label_sevcik,
        xlabel,
        file_name_sevcik)
```

Fig. 4.14 shows the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their Sevcik fractal dimension.

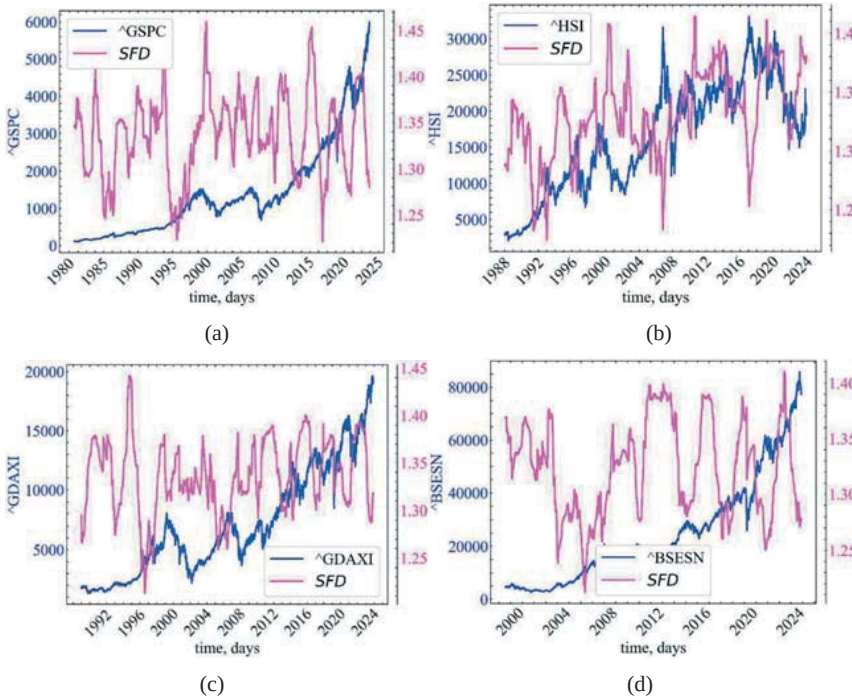


Fig. 4.14: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their Sevcik fractal dimension

We can see that Sevcik fractal dimension reacts with a decline to crash events in the stock market. The downturns during the crises of 2008, 2011, 2015, and 2020 are particularly characteristic. Stock price fluctuations during these crisis events were also characterized by an increase in persistence (correlations).

4.5.7 Calculating the fractal dimension through normalized length density

This is a fairly simple measure corresponding to the average absolute sequential differences of a (standardized) signal (`np.mean(np.abs(np.diff(std_signal)))`). The method was developed for measuring the complexity of signals of very short duration (< 30 samples), and can be used, for example, when continuous changes in the fractal dimension of a signal are of interest when computing within sliding windows.

Procedure syntax:

```
fractal_nld(signal, corrected=False)
```

Parameters:

- **signal** (*Union[list, np.array, pd.Series]*) – the signal (i.e., a time series) in the form of a vector of values;
- **corrected** (*bool*) – if True, will rescale the output value according to the power model estimated by Kalauzi et al. (2009) to make it more comparable with "true" FD range, as follows: `FD = 1.9079*((NLD-0.097178)^0.18383)`. Note that this can result in `np.nan` if the result of the difference is negative.

Returns:

- **NLDFD** (*float*) – fractal dimension;
- **info** (*dict*) – A dictionary containing additional information (currently, but returned nonetheless for consistency with other functions).

4.5.8 Calculating NLD fractal dimension within the sliding window procedure

We also don't need anything extra for this indicator:

```
window = 500    # window length
tstep = 1       # time step
ret_type = 4    # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
```

```
# 5 – absolute values (volatility)
# 6 – standardized series

nld_corrected = True # FD normalization

length = len(time_ser.values)      # series length

nldfd_wind = []                    # array of NLDFD
```

Start the sliding window procedure:

```
for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

# calculate NLDFD
    nld, _ = nk.fractal_nld(fragm,
                            corrected=nld_corrected)

# save the result to an array of values
    nldfd_wind.append(nld)
```

Save the initial values to a text document:

```
np.savetxt(f"fd_nld_name={symbol}_wind={window}_\
           step={tstep}_corrected={nld_corrected}.txt", nldfd_wind)
```

Define the labels for the figures and the titles of the saved figures:

```
label_nld = fr'$NLDFD$'

file_name_nld = f"fd_nld_name={symbol}_wind={window}_\
                step={tstep}_corrected={nld_corrected}"
```

Plot the results:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          nldfd_wind,
          ylabel,
          label_nld,
          xlabel,
          file_name_nld)
```

Fig. 4.15 presents the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their fractal dimension dynamics through the normalized length density.

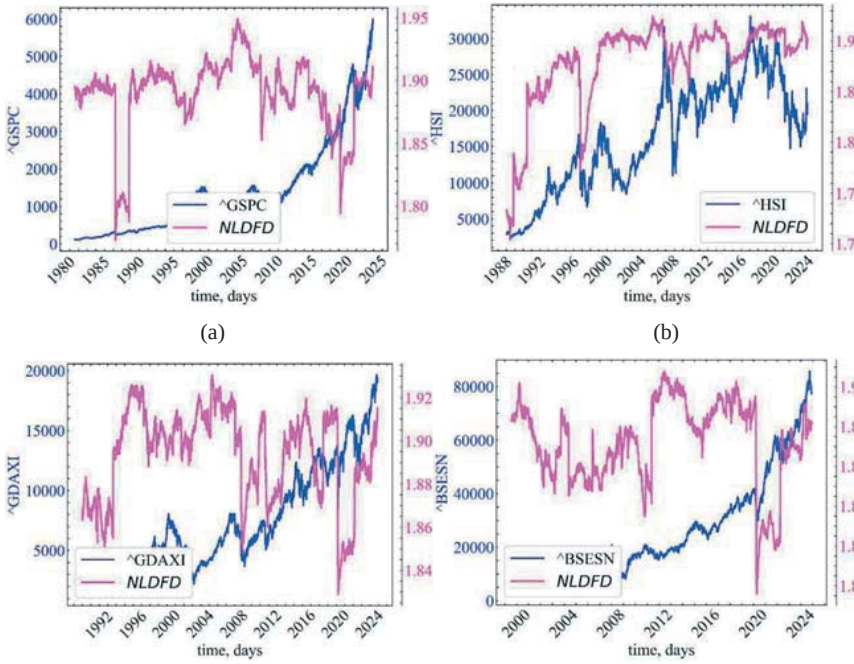


Fig. 4.15: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their fractal dimension dynamics through the normalized length density

Fig. 4.15 shows that *NLDFD* decreases during crisis and pre-crisis events, indicating that correlations increase during these periods.

4.5.9 Calculation of fractal dimension through the slope of the power spectral density

Let's use the following method of the `neurokit2` library:

```
fractal_psdslope(signal, method='voss1988', show=False, **kwargs)
```

Parameters:

- **signal** (*Union[list, np.array, pd.Series]*) – the signal (i.e., a time series) in the form of a vector of values;
- **method** (*str*) – method to estimate the fractal dimension from the slope, can be `"voss1988"` (default) or `"hasselman2013"`;

- **show** (*bool*) – if True, returns the log-log plot of PSD versus frequency;
- ****kwargs** – other arguments to be passed to `signal_psd()`.

Returns:

- **slope** (*float*) – estimate of the fractal dimension obtained from PSD slope analysis;
- **info** (*dict*) – a dictionary containing additional information regarding the parameters used to perform PSD slope analysis.

4.5.9.1 Calculating the PSD fractal dimension for the whole time series

For further calculations, let's first perform a series transformation. We will use the initial time series for further calculations:

```
signal = time_ser.copy()
ret_type = 1 # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

for_psd = transformation(signal, ret_type)
```

And now let's plot the power spectral density versus frequency on a logarithmic scale (see Fig. 4.16):

```
psdslope, info = nk.fractal_psdslope(for_psd, method="voss1988", show=True)
```

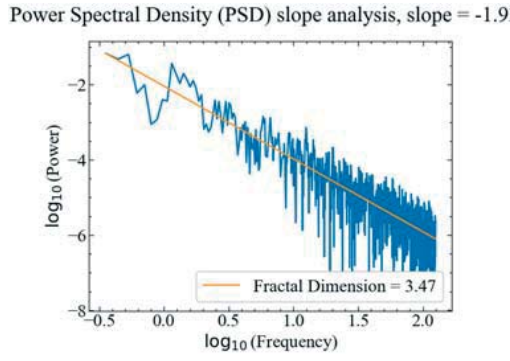


Fig. 4.16: Dependence of power spectral density on frequency in logarithmic scale for S&P 500 index

Obviously, the slope of the power spectral density at different frequencies has a linear dependence, and the slope of the line constructed from the spectrum is close to -2, indicating that the dynamics of the S&P 500 index is close to fractional Brownian motion.

Now let's consider the variation of the slope of the spectrum within the sliding window algorithm.

4.5.9.2 Calculating the PSD fractal dimension within the sliding window procedure

Let's set the following parameters:

```
window = 500    # sliding window width
tstep = 1       # sliding window time step
ret_type = 4    # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

method_psd = "voss1988" # method for calculating spectral density

length = len(time_ser.values)  # series length

psd_wind = []                  # array for PSDFD
```

Start the sliding window procedure:

```
for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

# calculate PSDFD
    psd, _ = nk.fractal_psdslope(fragm, method=method_psd)

# save results to an array
    psd_wind.append(psd)
```

Save the initial values to a text document:

```
np.savetxt(f"fd_psd_name={symbol}_method{method_psd}_\
           wind={window}_step={tstep}.txt", psd_wind)
```

Define the labels for the figures and the titles of the saved figures:

```
label_psd = fr'$PSDFD$'

file_name_psd = f"fd_psd_name={symbol}_method{method_psd}_\
                wind={window}_step={tstep}"
```

Plot the results:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          psd_wind,
          ylabel,
          label_psd,
          xlabel,
          file_name_psd)
```

In Fig. 4.17 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their fractal dimension by the slope of the power spectral density.

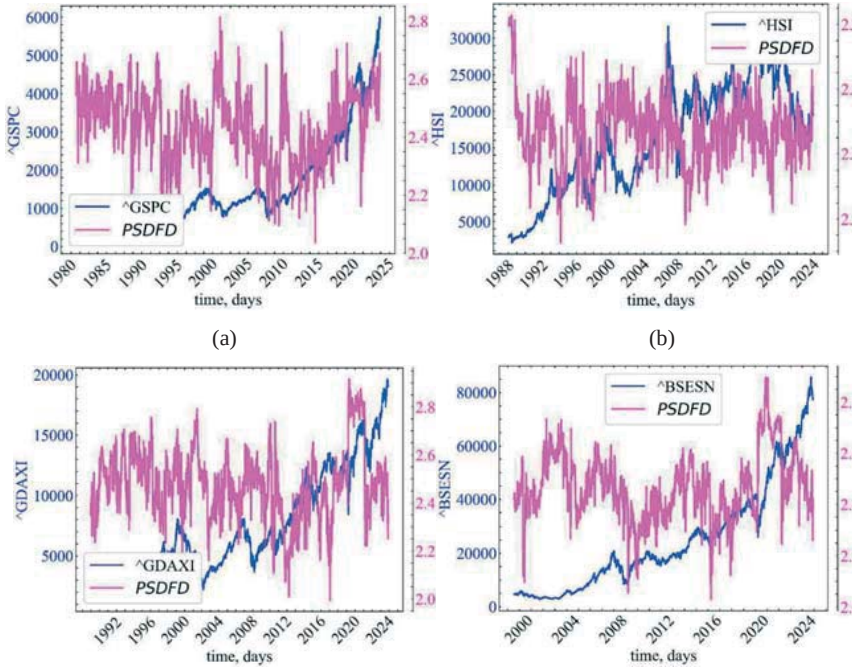


Fig. 4.17: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their fractal dimension by the slope of the power spectral density

The figure above shows that this indicator also reacts with a decline to crisis and pre-crisis events, indicating an increase in the autocorrelation of the time series. It is also clear that there are variations in the slope of the power density spectrum. At some points in time, the signal dynamics can be similar to Brownian motion, and at others to white noise.

4.5.10 Calculation of the correlation dimension

The correlation dimension is a lower bound for estimating the fractal dimension of the investigated phase space.

First, the phase space of the signal is reconstructed according to the time-delay method, and then the distances between all points of the trajectory are calculated. Then, the "correlation sum" is calculated, which is the proportion of pairs of points whose distance is less than a given radius. The final correlation dimension is approximated by a graph of the correlation sum versus the radius of the multidimensional neighborhood of the trajectories under study on a logarithmic scale.

This dimension can be called with `fractal_correlation()`. Its syntax is as follows:

```
fractal_correlation(signal, delay=1, dimension=2, radius=64, show=False, **kwargs)
```

Parameters:

- **signal** (*Union[list, np.array, pd.Series]*) – the signal (i.e., a time series) in the form of a vector of values;
- **delay** (*int*) – time delay (τ) in samples;
- **dimension** (*int*) – embedding dimension (d_E);
- **radius** (*Union[str, int, list]*) – the sequence of radiuses to test. If an integer is passed, will get an exponential sequence of length radius ranging from 2.5% to 50% of the distance range. Methods implemented in other packages can be used via `"nolds"`, `"Corr_Dim"` or `"boon2008"`;

- **show** (*bool*) – plot of correlation dimension if True. Defaults to False.

Returns:

- **cd** (*float*) – the correlation dimension (CD) of the time series;
- **info** (*dict*) – a dictionary containing additional information regarding the parameters used to compute the correlation dimension.

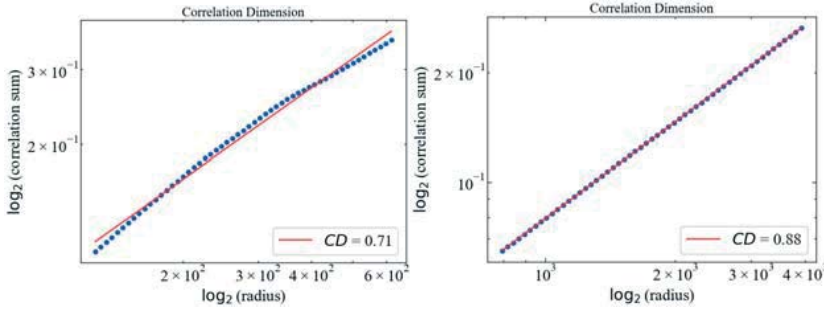
4.5.10.1 Calculating the correlation dimension for the whole time series

Let's consider the dependence of the correlation sum on the radius for the entire time series. First of all, let's transform the series:

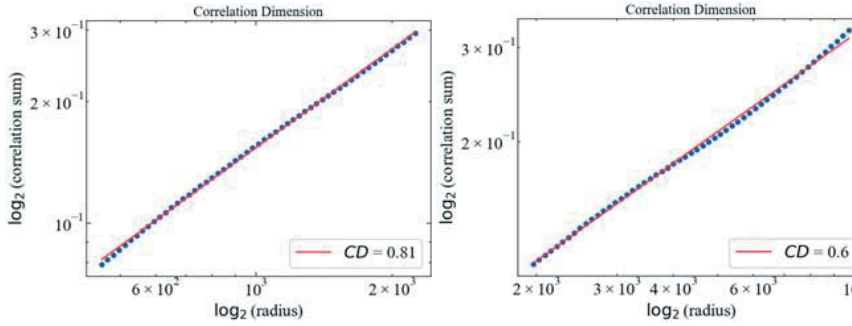
```
signal = time_ser.copy()
ret_type = 6 # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

for_corr = transformation(signal, ret_type)
```

Now let's calculate the correlation dimension by plotting the correlation sum against the radius on a logarithmic scale (see Fig. 4.18):

```
cd, info = nk.fractal_correlation(for_corr,
                                  delay=1,
                                  dimension=1,
                                  radius="nolds",
                                  show=True)
```



(a) (b)



(c) (d)

Fig. 4.18: Dependence of the correlation sum on the radius of the multidimensional neighborhood of the studied trajectories for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

As we can see, the correlation sum does indeed have a linear dependence for different values of the radius of the neighborhood of a particular trajectory, which indicates the fractal nature of the system. Now let's see how the value of the correlation dimension varies during periods of turbulence.

4.5.10.2 Calculating the correlation dimension within the sliding window algorithm

For this indicator, we define the following parameters:

```
window = 500    # sliding window width
tstep = 1       # sliding window time step
ret_type = 6    # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

d_wind = 3          # embedding dimension
tau_wind = 1        # time delay
rad_wind = "nolds" # method for determining an array of radii

length = len(time_ser.values)      # series length

corr_wind = []                     # array for CD
```

Start the sliding window procedure:

```
for i in tqdm(range(0, length-window, tstep)):
```

```
    fragm = time_ser.iloc[i:i+window].copy()

    fragm = transformation(fragm, ret_type)

# calculate correlation dimension
    cd_wind, _ = nk.fractal_correlation(fragm,
                                        delay=tau_wind,
                                        dimension=d_wind,
                                        radius=rad_wind)

# save results
    corr_wind.append(cd_wind)
```

Save the initial values to a text file:

```
np.savetxt(f"fd_correlation_name={symbol}_wind={window}_\
                step={tstep}_dim={d_wind}_tau={tau_wind}_\
                radius={rad_wind}.txt", corr_wind)
```

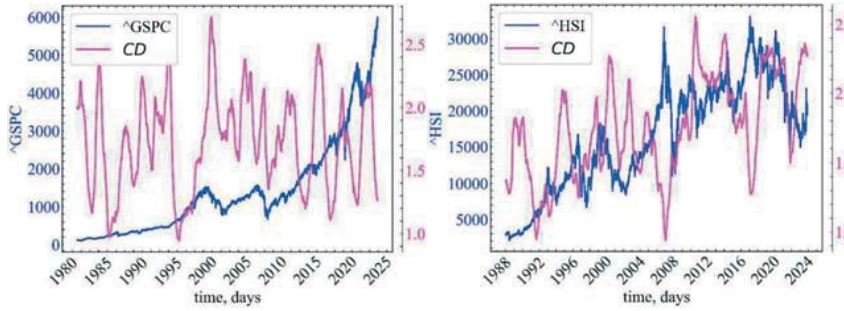
Define the labels for the figures and the titles of the saved figures:

```
label_cd = fr'$CD$'

file_name_cd = f"fd_correlation_name={symbol}_wind={window}_\
                step={tstep}_dim={d_wind}_tau={tau_wind}_\
                radius={rad_wind}"
```

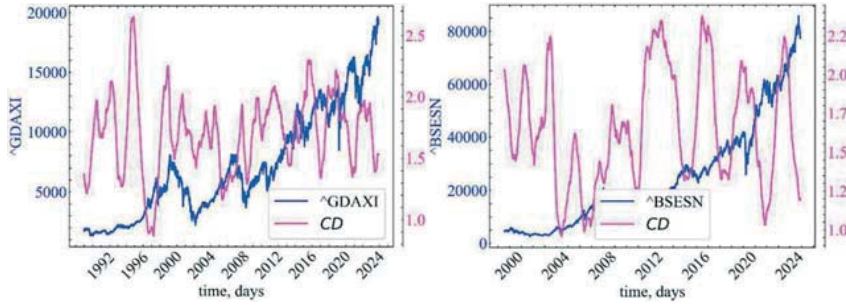
Plot the results:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          corr_wind,
          ylabel,
          label_cd,
          xlabel,
          file_name_cd)
```

Fig. 4.19 provides the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their correlation fractal dimension.



(a) (b)



(c) (d)

Fig. 4.19: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their correlation fractal dimension

Fig. 4.19 shows that the correlation dimension for the stock indices also decreases in crisis and pre-crisis periods, indicating that current stocks prices are more correlated with previous ones. Another way to put it is that during crises, traders begin to self-organize and collectively buy or sell the asset in question; in other words, their dynamics become more synchronized. Since the correlation dimension is measured for the trajectories of the phase space, a decline in this indicator indicates an increase in the density of the trajectories under study. That is, the phase space becomes sparser, and all its trajectories are concentrated in only one specific area, which is an indicator of the cohesion of the hidden variables of the system under study.

4.6 Definition of multifractals

In this laboratory, we will present the basics of the theory of multifractals — inhomogeneous fractal objects, for a complete description of which, in contrast to regular fractals, it is not enough to introduce only one quantity, its fractal dimension D, but a whole range of such dimensions is needed, the number of which, generally speaking, is infinite. The reason for this is that, along with purely geometric characteristics, which are determined by the value of D, such fractals are also characterized by some specific statistical properties. The easiest way to

explain what is meant by a “non-uniform fractal” is using the example of the Sierpinski triangle obtained using the method of random iterations (see Fig. 4.20).

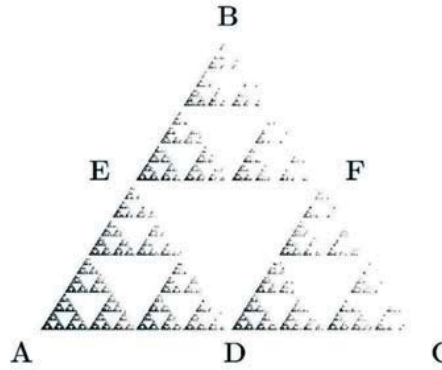


Fig. 4.20: Sierpinski triangle whose regions are generated with different probabilities

Suppose that in the random iteration method, we have now for some reason preferred one of the vertices of the triangle, for example, vertex A, and began to choose it with a probability of 90%. The other two vertices B and C are equivalent for us, but they now account for only 5% each. The result of such an “asymmetrical game” is shown in the figure above. It can be seen that the points inside the ABC triangle are now extremely unevenly distributed. Most of them are located near the top of A and its prototypes. At the same time, vertices B and C (and their prototypes) have very few of them. However, according to the usual terminology, this set of points (provided that the number of iterations tends to infinity) is a fractal, because the main property of the fractal — self-similarity — has been preserved. Indeed, the triangle DFC, although it has 20 times fewer points, is completely similar in its statistical and geometric properties to the large triangle ABC. As in the great triangle, the points in it are concentrated mostly near the vertex D, an analogue of the vertex A.

Fig. 4.21 shows in more detail the resulting distribution of points along the Sierpinski triangle. The numbers in each of the small triangles show its relative population of set points.

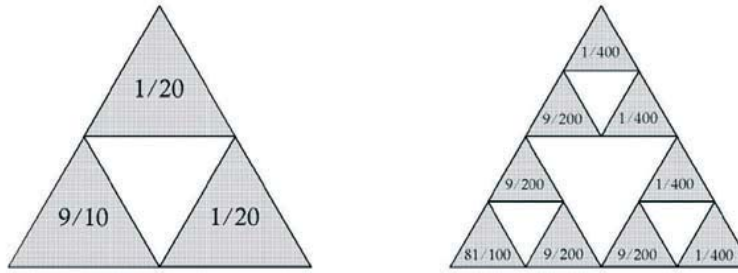


Fig. 4.21: Distribution of points along the Sierpinski triangle shown in the previous figure

However, despite the uneven distribution of fractal points, the fractal dimension remained the same, $D = \ln 3/\ln 2$. Covering this set with smaller and smaller triangles can be carried out according to the same algorithm as before. Such a coincidence makes us think about the search for new quantitative characteristics that could distinguish an uneven distribution of points from a uniform one. Another, more complex example of a non-uniform fractal, which we would like to cite, is shown in the following figure. On the left is a large square with a side equal to one, which at this (zero) stage completely covers some fractal set of points M. In the next (first) stage, in the center of the figure, it is shown how the same set can be covered by three smaller squares with sides $l_1 = 1/2$, $l_2 = l_3 = 5/16$, which, respectively, contain the quotient $p_1 = 1/2$, $p_2 = 1/3$ and $p_3 = 1/6$ of all points.

The next stage of coverage (shown in the figure on the right) already contains 9 squares with sides $l_1^2 = 1/4$, $l_1 l_2 = l_1 l_3 = 5/32$ (in the lower right corner) and $l_2 l_1 = 5/32$, $l_2^2 = l_2 l_3 = 25/256$ (top right and left). The relative population of these squares by the points of the set is shown in the figure. It corresponds to the product of population factors (probabilities): $p_1^2 = 1/4$, $p_1 p_2 = 1/6$, $p_1 p_3 = 1/12$ for the lower right group, $p_2 p_1 = 1/6$, $p_2^2 = 1/9$, $p_2 p_3 = 1/18$ for the upper left and $p_3 p_1 = 1/12$, $p_3 p_2 = 1/18$, $p_3^2 = 1/36$ for the upper right group. Note that there is a clear correspondence between the population of the square $p_j p_i$ and its size $l_i l_i$.

The further process of partitioning and covering the set M is carried out according to this renormalization scheme. Each square having size l and population p at the n -th step is replaced at $n+1$ step by three squares with dimensions of ll_1, ll_2, ll_3 and populations of pp_1, pp_2, pp_3 respectively, placed in the same way relative to each other, as shown in Fig. 4.22.

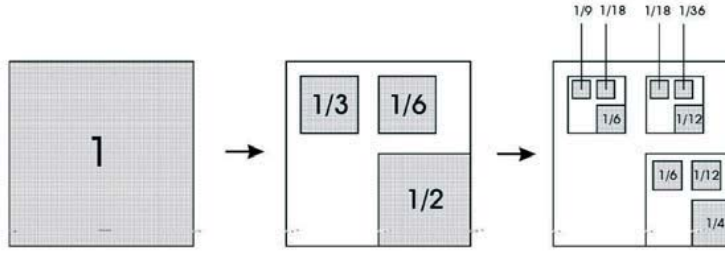


Fig. 4.22: An example of a multifractal obeying a renormalization scheme

Two of the cases discussed above are examples of heterogeneous fractals. By the word "heterogeneous" we mean the uneven distribution of the points of the set across the fractal or the uneven distribution of small and large fluctuations in the time series. The reason for the heterogeneity in the previous cases is the same: different probabilities of filling geometrically identical elements of the fractal, or, in the general case, the discrepancy between the filling probabilities and the geometric dimensions of the corresponding regions. Such inhomogeneous fractal objects are called **multifractals** in the literature, and we will study them in the future.

4.7 Generalized fractal dimensions D_q

Let's give a general definition of a multifractal. Consider a fractal object occupying some bounded area Ω size L in Euclidean space with dimension d. Suppose at some stage of its construction it is a set of $N \gg 1$ points somehow distributed in this region. We will assume that in the end, $N \to \infty$. An example of

such a set is the Sierpinski triangle constructed by random iterations. Each step of the iterative procedure adds one new point to this set.

Let's divide the entire Ω area into cubic cells with side $\varepsilon \ll L$ and volume ε^d. Let the number of occupied cells i vary between $i = 1,2,\ldots, N(\varepsilon)$, where $N(\varepsilon)$ is the total number of occupied cells, which, of course, depends on the size of cell ε.

Let $n_i(\varepsilon)$ represent the number of points in cell number i, then the value $p_i(\varepsilon) = \lim_{N\to\infty} n_i(\varepsilon)/N$ represents the probability that a randomly selected point in our set is in cell i. In other words, the probabilities p_i characterize the relative population of cells. From the condition of probability normalization, it follows that

$$\sum_{i=1}^{N(\varepsilon)} p_i\,(\varepsilon) = 1.$$

Let us now introduce into consideration the **generalized partition function** $Z(q,\varepsilon)$, which is characterized by an exponent of the degree q, which can acquire any values in the interval $-\infty < q < +\infty$

$$Z(q,\varepsilon) = \sum_{i=1}^{N(\varepsilon)} p_i^q\,(\varepsilon).$$

The spectrum of **generalized fractal dimensions** of D_q, which characterizes a given distribution of points in the Ω region, is determined by the relation $D_q = \tau(q)/(q-1)$, where the function $\tau(q)$ has the form

$$\tau(q) = \lim_{\varepsilon\to 0} \ln Z(q,\varepsilon) \,/\, \ln \varepsilon.$$

As we will show below, if $D_q = D = \text{const}$, i.e., independent of q, then a given set of points is an ordinary, regular fractal characterized by only one quantity, the fractal dimension D. Conversely, if the function D_q somehow changes with q, then the set of points in question represents a multifractal.

Thus, the multifractal in the general case is characterized by some **nonlinear** function $\tau(q)$, which determines the behavior of the statistical sum $Z(q,\varepsilon)$ at $\varepsilon \to 0$:

$$Z(q,\varepsilon) = \sum_{i=1}^{N(\varepsilon)} p_i^q\,(\varepsilon) \approx \varepsilon^{\tau(q)}. \qquad (4.8)$$

It should be borne in mind that in a real situation we always have a finite, albeit very large, number of discrete points N, so in computer simulation of a particular set, the limit transition $\varepsilon \to 0$ must be performed with caution, remembering that it is always preceded by a limit $N \to 0$.

Let us now show how the generalized statistical sum behaves in the case of an ordinary regular fractal with fractal dimension D. In this case, all occupied cells contain the same number of points, $n_i(\varepsilon) = N/N(\varepsilon)$, i.e. the fractal appears to be **homogeneous**. Then it is obvious that the relative populations of the cells, $p_i(\varepsilon) = 1/N(\varepsilon)$, are also the same, and the generalized statistical sum takes the form

$$Z(q,\varepsilon) = N^{1-q}(\varepsilon). \qquad (4.9)$$

Let us now consider that, according to the definition of the fractal dimension D, the number of occupied cells at a sufficiently small ε behaves as follows:

$$N(\varepsilon) \approx \varepsilon^{-D}. \qquad (4.10)$$

Substituting (4.10) for (4.9), and comparing with (4.8), we get

$$\varepsilon^{\tau(q)} = \varepsilon^{-D(1-q)} \to \tau(q) = (q-1)D. \qquad (4.11)$$

We conclude that in the case of an ordinary fractal, the function (4.11) is linear. Then all D_q are really independent of q. A fractal in which all generalized fractal dimensions do D_q coincide is called a **monofractal**.

If the distribution of points among the cells is not the same, then the fractal is called inhomogeneous, that is, it is a multifractal, and to characterize it, a whole range of generalized fractal dimensions D_q is required, the number of which, in the general case, is infinite.

Thus, for example, in $q \to +\infty$ the main contribution to the generalized statistical sum (4.8) is made by the cells containing the largest number of n_i particles in them and, accordingly, characterized by the highest probability of filling them p_i. On the contrary, at $q \to -\infty$, the main contribution to the generalized statistical sum is made by the most sparse cells with the lowest probability of filling them p_i. Thus, the function D_q shows how heterogeneous the studied set of points Ω seems to be.

Further, to characterize the distribution of points, it is necessary to know not only the function $\tau(q)$, but also its derivative:

$$\frac{d\tau(q)}{dq} = \lim_{\varepsilon \to 0} \sum_{i=1}^{N(\varepsilon)} p_i^q \ln p_i \Big/ \left(\sum_{i=1}^{N(\varepsilon)} p_i^q \right) \ln \varepsilon.$$

This derivative has an important physical content that will be demonstrated later. Now again note that for a multifractal system it does not remain constant and changes with q.

4.8 Multifractal spectrum function $f(\alpha)$

4.8.1 Spectrum of fractal dimensions

In the previous paragraph, we introduced the concept of a multifractal — an object that is an inhomogeneous fractal. To describe it, we introduced a set of generalized fractal dimensions D_q, where q takes any values in the interval $-\infty < q < +\infty$. However, the quantities of D_q are not, strictly speaking, fractal dimensions in the general sense of the word.

Therefore, the so-called **multifractal spectrum** $f(\alpha)$ (the spectrum of multifractal singularities) is often used to characterize a multifractal set, to which, as we will see later, the term fractal dimension is more suitable. We will show that the magnitude $f(\alpha)$ is actually equal to the Hausdorff dimension of some homogeneous fractal subset from the original set Ω, which gives the dominant contribution to the statistical sum for a given value q.

One of the main characteristics of a multifractal is a set of probabilities p_i that show the relative population of the cells of the ε with which we cover this set. The smaller the size of the cell, the smaller the value of its population. For self-similar sets, the dependence of p_i on the size of the cell ε has a power character:

$$p_i(\varepsilon) \approx \varepsilon^{\alpha_i},$$

where α_i represents some exponent (different for different cells i).

Additionaly on α

By directing the ε value to zero, the fractality can be considered locally for each point (element) of the system under study, and thus the α indicator is the **local fractal dimension**. It is also called the **Hölder exponent** or the **singularity strength**. We can observe a power dependence, since, obviously, the distribution of mass (fluctuations) is concentrated with different "strengths" α , so the probabilistic measure changes in proportion to the size of the windows ε

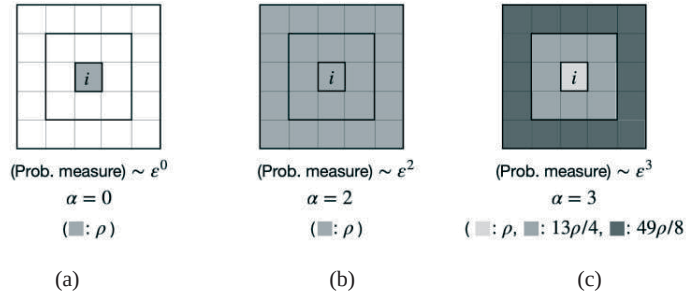


Fig. 4.23: Schematic representation of the relationship between singularity strength and density compared to neighborhoods

The gray scale represents a probabilistic measure for each location, as shown in each panel. In Fig. 4.23 (a), only the i-th location has a non-zero density, the rest of the spaces are empty. The probabilistic measure on the cell remains ρ even when the cell size ε increases, which is emphasized by a bold line. However, due to the fact that we do not observe an increase in density further, the α indicator remains zero. In Fig. 4.23 (b) all cells have the same density. The probabilistic

measures of the cells are ρ, 9ρ, and 25ρ for the smallest, second, smallest, and largest cell (highlighted in bold line). Thus, the singularity strength of the i-th cell is 2. Fig. 4.23 (c) The i-th cell is sparse compared to the surrounding cells. The probabilistic measure of the cells is ρ, 27ρ, and 125ρ for the smallest, second, smallest, and largest cells (highlighted with a bold line). Thus, the singularity strength of the i-th cell is 3.

Additionally on α

It can be said that the smoother the surface of the system appears, the fewer elements are involved in its development, the smaller the singularity indicator. The more elements of the system interconnect with each other, the more processes take place during the evolution of the system, the greater the singularity index

It is known that for a regular (homogeneous) fractal, all exponents of the α_i degree are the same and equal to the fractal dimension D:

$$p_i = 1/N(\varepsilon) \approx \varepsilon^D.$$

In this case, the statistical sum (4.8) takes the following form:

$$Z(q,\varepsilon) = \sum_{i=1}^{N(\varepsilon)} p_i^q\,(\varepsilon) = N(\varepsilon)\varepsilon^{Dq} = \varepsilon^{-D}\varepsilon^{Dq} \approx \varepsilon^{D(q-1)}.$$

Therefore, $\tau(q) = D(q-1)$ and all generalized fractal dimensions $D_q = D$ in this case coincide and do not depend on q.

However, for such a complex object as a multifractal, due to its heterogeneity, the probabilities of filling cells p_i generally vary, and the α_i degree indicator for different cells can take different values. It is quite typical that these values continuously fill some closed interval $(\alpha_{min}, \alpha_{max})$, with $p_{min} \approx \varepsilon^{\alpha_{max}}$ and $p_{max} \approx \varepsilon^{\alpha_{min}}$.

Now let's move on to the question of the probability distribution of different values of α_i. Let $n(\alpha)d\alpha$ be the probability that α_i is in the range α to $\alpha + d\alpha$. In other words, $n(\alpha)d\alpha$ is the relative number of cells i characterized by the same

degree of p_i with α_i lying in this interval. In the case of a monofractal for which all α_i are the same (and equal to the fractal dimension D), this number is obviously proportional to the total number of cells $N(\varepsilon) \approx \varepsilon^{-D}$ power-dependent on the size of cell ε. The exponent in this relation is determined by the fractal dimension of the set D.

For a multifractal, however, this is not the case, and different values of α_i meet with a probability characterized not by the same value D, but by different (depending on α) values of the exponent $f(\alpha)$:

$$n(\alpha) \approx \varepsilon^{-f(\alpha)}. \tag{4.12}$$

Thus, the physical meaning of the function $f(\alpha)$ is that it represents the Hausdorff dimension of some homogeneous subset Ω_α from the original set of Ω, characterized by the same probabilities of filling cells $p_i \approx \varepsilon^\alpha$. Since the fractal dimension of the subset is obviously always less than or equal to the fractal dimension of the original set D_0, there is an important inequality for the function $f(\alpha)$:

$$f(\alpha) \leq D_0.$$

As a result, we can conclude that the set of different values of the function $f(\alpha)$ (at different α) represents the spectrum of fractal dimensions [94, 95] of homogeneous subsets Ω_α, into which the original set of Ω can be divided, each of which is characterized by its own value of fractal dimension $f(\alpha)$.

4.8.2 Legendre transformation

Let us establish the connection of the function $f(\alpha)$ with the previously introduced function $\tau(q)$. For this, let's calculate the partition function $Z(q, \varepsilon)$. Substituting the probabilities $p_i \approx \varepsilon^{\alpha_i}$ in $Z(q, \varepsilon)$, and moving from summation by i to integration by α with the probability density (7.5), we get

$$Z(q,\varepsilon) = \sum_{i=1}^{N(\varepsilon)} p_i^q\,(\varepsilon) \approx \int d\alpha n(\alpha)\varepsilon^{q\alpha} \approx \int d\alpha \varepsilon^{q\alpha - f(\alpha)}. \tag{4.13}$$

Since the magnitude of ε is very small, the main contribution to this integral will be made by those values $\alpha(q)$ at which the exponent $q\alpha - f(\alpha)$ is minimal (and the subintegral function is maximum). This contribution will be proportional to the value of the subintegral function at the maximum point. The value of $\alpha(q)$ itself is determined from the following condition:

$$\frac{d}{d\alpha}[q\alpha - f(\alpha)]\Big|_{\alpha=\alpha(q)} = 0.$$

Also, from the minimum condition, we have

$$\frac{d^2}{d\alpha^2}[q\alpha - f(\alpha)]\Big|_{\alpha=\alpha(q)} > 0.$$

As a result, we get that the dependence $\alpha(q)$ is implicitly defined from $q = df(\alpha)/d\alpha$, and that the function $f(\alpha)$ is convex everywhere:

$$f''(\alpha) > 0.$$

This means that the value $f\big(\alpha(q)\big)$ is indeed the fractal dimension of the subset $\Omega_{\alpha(q)}$ that has the largest dominant contribution to the statistical sum (7.6) for a given value of the exponent q.

Since $Z(q, \varepsilon) = \tau(q)$, we conclude that

$$\tau(q) = q\alpha(q) - f\big(\alpha(q)\big). \tag{4.14}$$

Remembering that $\tau(q) = D(q-1)$, we can find the function D_q:

$$D_q = \frac{1}{q-1}\big[q\alpha(q) - f\big(\alpha(q)\big)\big]. \tag{4.15}$$

Thus, if we know the function of the multifractal spectrum $f(\alpha)$, then from the relation (4.15) and (4.16) we can find the function D_q. On the contrary, knowing D_q, we can reproduce the relationship $\alpha(q)$ using the equation

$$\alpha(q) = \frac{d}{dq}\big[(q-1)D_q\big] \tag{4.16}$$

and then find $f\big(\alpha(q)\big)$ from (4.16). These two equations define the function $f(\alpha)$.

$$\frac{d\tau}{dq}\frac{dq}{d\alpha} = q + \alpha\frac{dq}{d\alpha} - \frac{df}{d\alpha}.$$

Taking into account that $q = df/d\alpha$, and reducing this equation by $dq/d\alpha$, we arrive at the ratio $\alpha = d\tau(q)/dq$, which is equivalent to (4.16).

The expressions for $\tau(q)$ and $\alpha(q)$ define the **Legendre transformation** [124, 154] from the variables $\{q, \tau(q)\}$ to the variables $\{\alpha, f(\alpha)\}$: $\alpha = d\tau/dq$ and $f(\alpha) = q(d\tau/dq) - \tau(q)$. As you know, for a homogeneous fractal $D_q = D =$ const. Therefore, $\alpha = d\tau/dq = D$ and $f(\alpha) = q\alpha - \tau(q) = qD - D(q-1) = D$. In this case, the "graph" of the function $f(\alpha)$ on the plane $\big(\alpha, f(\alpha)\big)$ consists of only one point (D, D).

4.9 Multifractal detrended fluctuation analysis

Monofractal and multifractal structures of financial signals are a special kind of scale-invariant structures. Most often, the monofractal structure of financial time series is defined by a single power law and implies that scale invariance does not depend on time and space. However, we can often observe spatial and temporal variation of the scale-invariant structure of the complex system under study. These space-time variations indicate the multifractal nature of the financial signal, which is defined by the multifractal spectrum. The multifractal spectrum can help to quantify the asymmetry of ups and downs in the stock or cryptocurrency markets, predict a gradually approaching financial crisis, and thus contribute to the success of further trading decisions. The main purpose of this section is to present one of the most accurate procedures for determining a set of fractal indicators, the **multifractal detrended fluctuation analysis** (MF-DFA) [87-89], which is still one of the most powerful methods for analyzing systems of different nature and complexity [61, 63, 72, 82, 101, 113, 125, 140, 146, 179].

There are 9 steps for MF-DFA:

1. "*Noise and random walks in time series*" presents a method for making a time series look like a random walk.

2. "*Calculating the standard deviation of a time series*" introduces the standard deviation, which is the basic procedure for further calculations in MF-DFA and a typical way to calculate the average variation of time series of various nature.
3. "*Local RMS variation of time series*" represents the calculation of the local variation of the time series as the standard deviation of the time series within segments that may or may not overlap.
4. "*Local time series detrending*" is the calculation of the same local standard deviation around trends that are often found in financial time series.
5. "*Monofractal detrended fluctuation analysis*"*:* the amplitudes of local standard deviations are summed into a generalized standard deviation. The total standard deviation for segments with small sample sizes is dominated by fast fluctuations in the time series. In contrast, the total standard deviation for segments with large sample sizes is dominated by slow fluctuations. The power law relationship between the total standard deviation for several sample sizes (i.e., scales) is determined using monofractal detrended fluctuation analysis (DFA) and is called the Hurst exponent (H).
6. "*Multifractal detrended fluctuation analysis*"*:* MF-DFA is obtained by expanding the generalized standard deviation by the q-th order. The qth order standard deviation can distinguish between segments with small and large fluctuations. The power-law relationship between the q-th order standard deviation is numerically defined as the q-th order generalized Hurst exponent.
7. "*Multifractal spectrum of time series*"*:* several multifractal spectra are calculated based on the q-order Hurst index.
8. "*Generalized fractal dimensions*" presents a more detailed description of the D_q indicators, which will be described further.

9. "*Analogies of multifractals with thermodynamics*" shows that the obtained quantitative multifractal indicators have a connection with thermodynamic indicators, which allowed us to derive the multifractal 'heat capacity'.

To further visualize each step of the MF-DFA procedure, we import the following modules:

```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import numpy as np
import neurokit2 as nk
import yfinance as yf
import pandas as pd
import scienceplots
from scipy.integrate import cumulative_trapezoid
from tqdm import tqdm

%matplotlib inline
```

And let's adjust the figures for output:

```
plt.style.use(['science', 'notebook', 'grid'])

size = 22
params = {
'figure.figsize': (8, 6),
'font.size': size,
'lines.linewidth': 2,
'axes.titlesize': 'small',
'axes.labelsize': size,
'legend.fontsize': size,
'xtick.labelsize': size,
'ytick.labelsize': size,
"font.family": "Serif",
"font.serif": ["Times New Roman"],
'savefig.dpi': 300,
'axes.grid': False
}

plt.rcParams.update(params)
```

Here, we will present the description of multifractal analysis algorithm using the example of S&P 500 index. When describing the MF-DFA procedure, we will compare the multifractality of this series with artificially generated monofractal series, the complexity of which should obviously be less.

Let us again plot the time series for further explanation:

```
fig, ax = plt.subplots(1, 1)
ax.plot(time_ser.index, time_ser.values)
ax.legend([symbol])
```

```
ax.set_xlabel(xlabel)
ax.set_ylabel(ylabel)

plt.xticks(rotation=45)

plt.savefig(f'{symbol}.jpg')
plt.show();
```

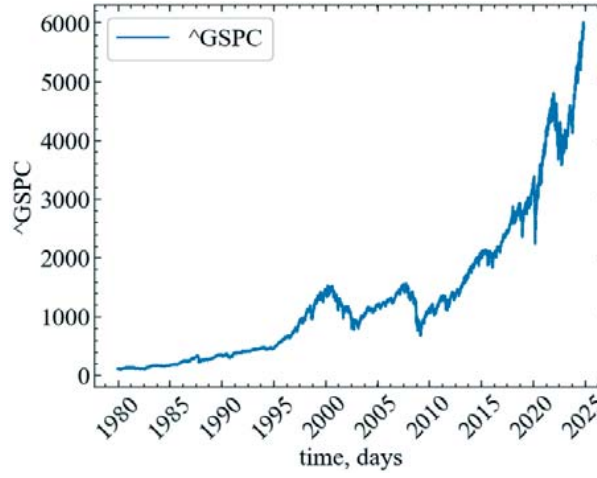


Fig. 4.24: Dynamics of daily changes in S&P 500 index

The last thing we need to do is to transform the original series to the returns. To do this, let's define the `transformation()` function and use it to find the returns:

```
def transformation(signal, ret_type):

    for_rec = signal.copy()

    if ret_type == 1:
        pass
    elif ret_type == 2:
        for_rec = for_rec.diff()
    elif ret_type == 3:
        for_rec = for_rec.pct_change()
    elif ret_type == 4:
        for_rec = for_rec.pct_change()
        for_rec -= for_rec.mean()
        for_rec /= for_rec.std()
    elif ret_type == 5:
        for_rec = for_rec.pct_change()
        for_rec -= for_rec.mean()
        for_rec /= for_rec.std()
```

```
        for_rec = for_rec.abs()
    elif ret_type == 6:
        for_rec -= for_rec.mean()
        for_rec /= for_rec.std()

    for_rec = for_rec.dropna().values

    return for_rec

signal = time_ser.copy()
ret_type = 4 # type of a series:

# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

sp_ret = transformation(signal, ret_type) # calculate returns
sp_length = len(sp_ret)                   # define length of a series
```

As already mentioned, when describing the MF-DFA procedure, we will also use monofractal signals for comparison. For further calculations, we will generate a signal of white and pink noise. The `signal_noise()` function of the `neurokit2` library can help us with this. This function generates pure Gaussian (1/f)**beta noise. The power spectrum of the generated noise is proportional to S(f)=(1/f)**beta. The following categories of noise have been described:

- violet noise: beta $= -2$;
- blue noise: beta $= -1$;
- white noise: beta $= 0$;
- flicker/pink noise: beta $= 1$;
- brown noise: beta $= 2$.

Its syntax is as follows:

`signal_noise(duration=10, sampling_rate=1000, beta=1, random_state=None)`

Parameters:

- **duration** (*float*) – desired length of duration (s);
- **sampling_rate** (*int*) – the desired sampling rate (in Hz, i.e., samples/second);

- **beta** (*float*) – the noise exponent;
- **random_state** (None, int, numpy.random.RandomState or numpy.random.Generator) – seed for the random number generator.

Returns:

- **noise** (*array*) – the signal of pure noise.

Now we can generate 2 types of noises:

```
white_noise = nk.signal_noise(duration=sp_length, # generate white noise
                              sampling_rate=1,
                              beta=0,
                              random_state=123)

pink_noise = nk.signal_noise(duration=sp_length,  # generate pink noise
                              sampling_rate=1,
                              beta=1,
                              random_state=123)
```

4.9.1 Noise and random walks in time series

The multifractal detrended fluctuation analysis of is based on the classical detrended fluctuation analysis (DFA). Classical DFA is applied to time series with a structure similar to random walks [48]. However, most financial time series have fluctuations that are more similar to random walk increments. If a financial time series has a noise-like structure, like returns, it should be converted to a random walk time series before applying DFA. Noise can be converted to random walk by subtracting the mean and integrating the time series (finding its cumulative sum). The white noise time series, monofractal (pink noise), and multifractal are noisy time series and are converted to random walks (see Fig. 4.25):

```
RW1 = np.cumsum(white_noise-np.mean(white_noise)) # random walk of white nois
e
RW2 = np.cumsum(pink_noise-np.mean(pink_noise))   # random walk of the monofr
actal
RW3 = np.cumsum(sp_ret-np.mean(sp_ret))         # random walk for S&P 500

fig, ax = plt.subplots(3, 1, sharex=True)

ax[0].plot(time_ser.index[1:], sp_ret)
ax[0].plot(time_ser.index[1:], RW3, 'r')
ax[0].margins(x=0)
ax[0].set_title('Multifractal time series', fontsize=16)

ax[1].plot(time_ser.index[1:], pink_noise, label= 'Noise-like time series')
```

```
ax[1].plot(time_ser.index[1:], RW2, 'r', label='Random walk')
ax[1].margins(x=0)
ax[1].set_title('Monofractal time series', fontsize=16)
ax[1].legend()

ax[2].plot(time_ser.index[1:], white_noise)
ax[2].plot(time_ser.index[1:], RW1, 'r')
ax[2].margins(x=0)
ax[2].set_title('White noise', fontsize=16)

plt.show();
```

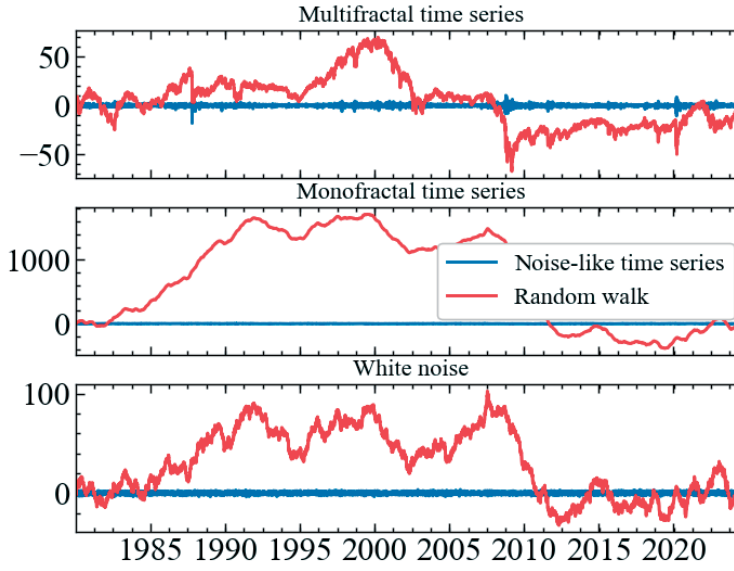


Fig. 4.25: Multifractal (top panel), monofractal (middle panel), and white noise-like (bottom panel) time series

4.9.2 Calculating the standard deviation of time series

The traditional analysis of the variation in a time series is to calculate the average value of the variation as the standard deviation. The reader can use the code below to calculate the standard deviation for time series with white noise, monofractal, and multifractal data:

```
RMS_ordinary = np.sqrt(np.mean(white_noise**2))    # root mean square variati
on of white noise
RMS_monofractal = np.sqrt(np.mean(pink_noise**2))  # root mean square variati
on of monofractal
RMS_multifractal = np.sqrt(np.mean(sp_ret**2))    # root mean square variatio
n of multifractal
```

```
fig, ax = plt.subplots(3, 1, sharex=True)

ax[0].plot(time_ser.index[1:], sp_ret, label='Noise-like time series')
ax[0].axhline(y=np.mean(sp_ret), c='r', linestyle='--', label='Mean')
ax[0].axhline(y=np.mean(sp_ret)+RMS_multifractal, c='r', linestyle='-', label
='+/- 1 RMS')
ax[0].axhline(y=np.mean(sp_ret)-RMS_multifractal, c='r', linestyle='-')
ax[0].set_ylim(-20, 20)
ax[0].margins(x=0)
ax[0].set_title('Multifractal time series', fontsize=16)

ax[1].plot(time_ser.index[1:], pink_noise)
ax[1].axhline(y=np.mean(pink_noise), c='r', linestyle='--')
ax[1].axhline(y=np.mean(pink_noise)+RMS_monofractal, c='r', linestyle='-')
ax[1].axhline(y=np.mean(pink_noise)-RMS_monofractal, c='r', linestyle='-')
ax[1].margins(x=0)
ax[1].set_title('Monofractal time series', fontsize=16)

ax[2].plot(time_ser.index[1:], white_noise)
ax[2].axhline(y=np.mean(white_noise), c='r', linestyle='--')
ax[2].axhline(y=np.mean(white_noise)+RMS_ordinary, c='r', linestyle='-')
ax[2].axhline(y=np.mean(white_noise)-RMS_ordinary, c='r', linestyle='-')
ax[2].margins(x=0)
ax[2].set_title('White noise', fontsize=16)

handles, labels = ax[0].get_legend_handles_labels()
fig.legend(handles, labels, loc='lower center')

plt.show();
```

Fig. 4.26 shows multifractal, monofractal, and white noise-like time series with zero mean (red dashed line) and ±RMS (red solid line).

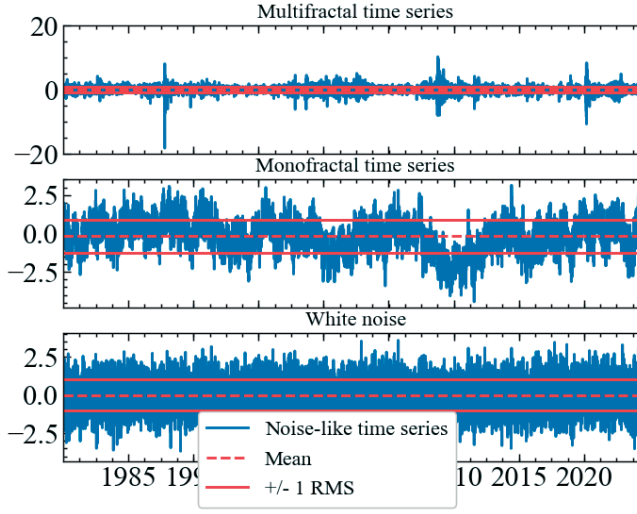


Fig. 4.26: Multifractal (top panel), monofractal (middle panel) and white noise-like (bottom panel) time series with zero mean (red dashed line) and ±RMS (red solid line)

Fig. 4.26 illustrates that the average amplitude of variation (i.e., standard deviation) is the same for all three time series, even though they have quite different structures. MF-DFA can distinguish between these structures.

4.9.3 Local RMS variation of time series

The multifractal time series in the top panel have local fluctuations of different magnitudes. The Root Mean Square Deviation (RMS) in the previous code can be calculated for segments of the time series to distinguish the magnitude of local fluctuations. A simple procedure is to divide the time series into equal-sized, non-overlapping segments and calculate the local RMS for each segment. This can be done with the code below:

```
def calc_rms(arr, scale=1335, m=1):

    # simulate a random walk (X)
    X = np.cumsum(arr - np.mean(arr))

    # transpose the values of X
    X = X.T

    # determine the length of the segments
    scale = scale

    # determine the order of the polynomial
    m = m

    # determining the number of segments
    segments = np.floor(len(X) / scale).astype(int)

    Index = {}  # dictionary of value indices
    fit = {}    # a dictionary for saving the obtained polynomial curves for
each segment
    RMS = []    # list of standard deviations

    for v in range(segments+1):       # go through each segment
        Idx_start = v * scale         # determine the initial value of the se
gment
        Idx_stop = (v+1) * scale    # determine the final value

        # form an array of indices of the values of the segment under study
        Index[v] = np.arange(Idx_start, min(Idx_stop, len(X)))

        # get values by indices
```

```
        X_Idx = X[Index[v]]

        # determine the polynomial coefficients of order m
        C = np.polyfit(Index[v], X_Idx, m)

        # build a polynomial curve according to the determined coefficients
        fit[v] = np.polyval(C, Index[v])

        # determine the variation of the series around the polynomial trend
        RMS.append(np.sqrt(np.mean((X_Idx - fit[v])**2)))

    return fit, RMS, Index, X
```

The first line of code of the `calc_rms()` function converts a noisy time series, a multifractal, into a random walk time series X. The third line of code sets the scale of the parameter that determines the sample size of non-overlapping segments for which the local root mean square deviation, RMS, is calculated. The fifth line is the number of segments into which the time series X can be divided, where `len(X)` is the sample size of the time series X. Thus, `segments = 8` with `len(X) = 11316` and `scale = 1335`. Lines nine through sixteen are a loop that calculates the local rms value around the trend `fit[v]` for each segment, updating the time index. In the first loop, $v = 0$, `Index[0]` goes from 0 to 1335 segment values (not inclusive). In the second cycle, $v = 1$, `Index[1]` goes from 1335 to 2670 of the second segment value. In the last cycle $v = 8$, `Index[8]` goes from 10680 to 12015 (not inclusive).

4.9.4 Local time series detrending

In complex systems, there are slowly changing trends, so to quantify the scale-invariance of fluctuations around these trends, it is necessary to detrend the signal. In the previous code, a polynomial trend `fit[v]` is fitted to X at each segment v. The parameter m determines the order of the polynomial. The polynomial trend is linear if $m = 1$, quadratic if $m = 2$, and cubic if $m = 3$. The line `C = np.polyfit(Index[v], X[Index[v]], m)` defines the coefficients of the polynomial `C` used to create the polynomial trend `fit[v]` for each segment. Then, for the residual variation, `X(Index[v])-fit[v]`, the local root mean

square deviation, RMS[v], is calculated within each segment v. The local root mean square variation, RMS[v], is shown in Fig. 4.27 as the distance between the red dashed trends and the red solid lines.

```
fit_1, RMS_1, Index_1, X = calc_rms(sp_ret, scale=1335, m=1) # estimation of
local deviation for multifractal
fit_2, RMS_2, Index_2, X = calc_rms(sp_ret, scale=1335, m=2) # Estimation of
local deviation for a monofractal
fit_3, RMS_3, Index_3, X = calc_rms(sp_ret, scale=1335, m=3) # local deviatio
n estimation for white noise

fig, ax = plt.subplots(3, 1, sharex=True)


ax[0].plot(time_ser.index[1:], X)
for v in list(fit_1.keys()):
    ax[0].plot(time_ser.index[Index_1[v]], fit_1[v], 'r--')
    ax[0].plot(time_ser.index[Index_1[v]], fit_1[v]+RMS_1[v], c='r', linestyl
e='-')
    ax[0].plot(time_ser.index[Index_1[v]], fit_1[v]-RMS_1[v], c='r', linestyl
e='-')

ax[0].margins(x=0)
ax[0].set_title('Linear detrending '+r'$(m=1)$', fontsize=16)


ax[1].plot(time_ser.index[1:], X, label='Random walk of a multifractal signal
')
for v in list(fit_2.keys()):
    if v == 1:
        ax[1].plot(time_ser.index[Index_2[v]], fit_2[v], 'r--', label='Local
trend')
        ax[1].plot(time_ser.index[Index_2[v]], fit_2[v]+RMS_2[v], c='r', line
style='-', label='+/- 1 RMS')
        ax[1].plot(time_ser.index[Index_2[v]], fit_2[v]-RMS_2[v], c='r', line
style='-')
else:
        ax[1].plot(time_ser.index[Index_2[v]], fit_2[v], 'r--')
        ax[1].plot(time_ser.index[Index_2[v]], fit_2[v]+RMS_2[v], c='r', line
style='-')
        ax[1].plot(time_ser.index[Index_2[v]], fit_2[v]-RMS_2[v], c='r', line
style='-')

ax[1].margins(x=0)
ax[1].set_title('Quadratic detrending '+r'$(m=2)$', fontsize=16)


ax[2].plot(time_ser.index[1:], X)
for v in list(fit_3.keys()):
    ax[2].plot(time_ser.index[Index_3[v]], fit_3[v], 'r--')
    ax[2].plot(time_ser.index[Index_3[v]], fit_3[v]+RMS_3[v], c='r', linestyl
e='-')
    ax[2].plot(time_ser.index[Index_3[v]], fit_3[v]-RMS_3[v], c='r', linestyl
```

```
e='-')

ax[2].margins(x=0)
ax[2].set_title('Cubic detrending '+r'$(m=3)$', fontsize=16)

handles, labels = ax[1].get_legend_handles_labels()
fig.legend(handles, labels, loc='lower center')

plt.show();
```

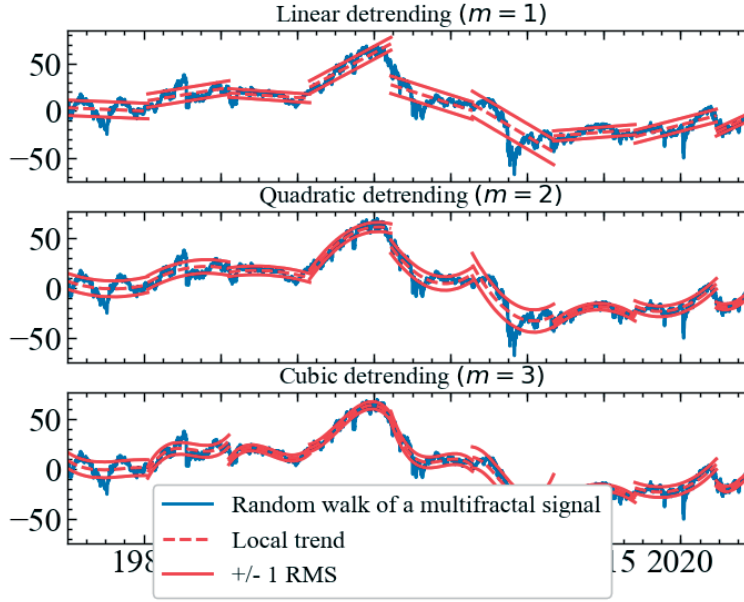


Fig. 4.27: Calculation of local `RMS` fluctuations around linear, quadratic, and cubic trends using the `calc_rms()` function ($m = 1$, $m = 2$, and $m = 3$, respectively). The red dashed line is the fitted trend, `fit[v]`, in seven segments of the 1335 sample. The distance between the red dashed trend and the solid red lines is $\pm$ `RMS`

4.9.5 Monofractal detrended fluctuation analysis

In DFA, variations in local RMS are quantified by the overall RMS (F).

Fast fluctuations in the time series X will affect the total RMS deviation F in segments of short length (scale), while slow fluctuations will affect F in segments of long length (scale). Thus, the fluctuation function F should be calculated for several scales to isolate the influence of both fast and slow fluctuations, which in turn determine the structural transformations of the time series. The fluctuation

function $F(ns)$ can be calculated for several scales by modifying the previous code:

```
def calc_F(arr, scale, m=1):

    X = np.cumsum(arr - np.mean(arr)) # simulate a random walk (X)
    X = X.T                           # transpose the values of X

    scale = scale
    m = m
    segments = np.zeros(len(scale), dtype=int)
    F = np.zeros(len(scale))

    Index = {}  # dictionary of value indices
    fit = {}    # a dictionary for saving the obtained polynomial curves for
each segment
    RMS = {}    # list of standard deviations

    for ns in range(len(scale)):
        segments[ns] = np.floor(len(X) / scale[ns]).astype(int)
        RMS[ns] = np.zeros(segments[ns])

        for v in range(segments[ns]):         # go through each segment
# determine the initial value of the segment
            Idx_start = v * scale[ns]

# determine the final value
            Idx_stop = (v + 1) * scale[ns] if v < segments[ns] - 1 else len(X
)

# form an array of indices of the values of the segment under study
            Index[v, ns] = np.arange(Idx_start, Idx_stop)

# get values by indices
            X_Idx = X[Index[v, ns]]

# determine the polynomial coefficients of order m
            C = np.polyfit(Index[v, ns], X_Idx, m)

# build a polynomial curve according to the determined coefficients
            fit[v, ns] = np.polyval(C, Index[v, ns])

# estimate the standard deviation for the fragment v on the scale ns
            RMS[ns][v] = np.sqrt(np.mean((X_Idx - fit[v, ns])**2))

# estimate the total standard deviation within the scale ns
        F[ns] = np.sqrt(np.mean(RMS[ns]**2))

    return F, RMS, Index, X

scales = [16, 32, 64, 128, 256, 512, 1024][::-1]
F, RMS, Index, X = calc_F(sp_ret, scale=scales) # estimation of the generaliz
ed fluctuation function on different scales
```

```
fig, ax = plt.subplots(len(scales), sharex=True)

for scale, val in enumerate(scales):
    l = [Index[val] for val in Index.keys() if (val[1] == scale)]

    x = np.array([])
    for v in l:
        x = np.concatenate([x, v])

    y = np.array([])
    for idx, v in enumerate(l):
        y = np.concatenate([y, RMS[scale][idx]*np.ones(len(v))])

    if scales[scale] == 16:
        ax[scale].plot(time_ser.index[1:], y, c='b', label=" Local fluctuatio
ns: RMS")
        ax[scale].axhline(y=F[scale], c='r', linestyle='-', label=r"RMS of lo
cal fluctuations: $F$")
        ax[scale].set_title(f"Scale = {scales[scale]}", fontsize=16)
        ax[scale].margins(x=0)
    else:
        ax[scale].plot(time_ser.index[1:], y, c='b')
        ax[scale].axhline(y=F[scale], c='r', linestyle='-')
        ax[scale].set_title(f"Scale = {scales[scale]}", fontsize=16)
        ax[scale].margins(x=0)

handles, labels = ax[-1].get_legend_handles_labels()
fig.legend(handles, labels, loc='upper left', fontsize=14)

fig.tight_layout(pad=0.05)
plt.show();
```

Fig. 4.28 will represent the dynamics of the generalized fluctuation function calculated for different time scales.

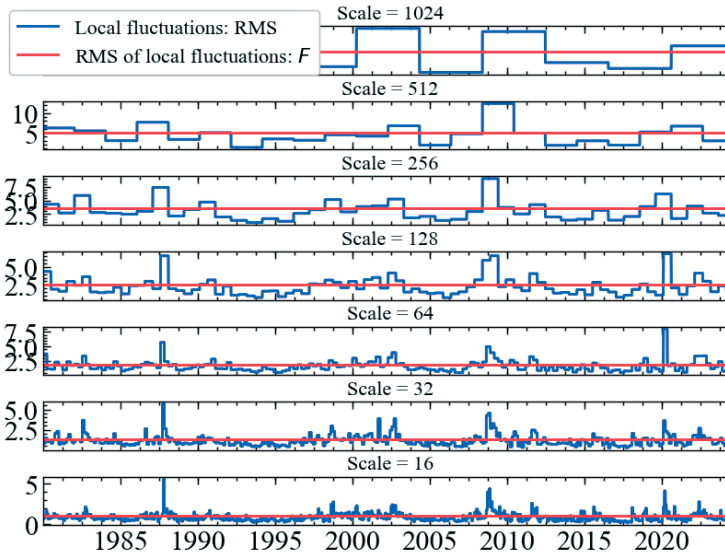


Fig. 4.28: Local fluctuations `RMS[ns]` are calculated for segments with different scales. The fluctuation function `F[ns]` is the total standard deviation of the local fluctuations `RMS[ns]`. Note that `F[ns]` decreases with decreasing scale

DFA determines the monofractal structure of the time series according to the power law relationship between the total standard deviation (i.e., F) calculated for several scales. The power-law relationship between the total RMS deviation is denoted by the slope (H) of the regression line, calculated using the following code:

```
C = np.polyfit(np.log(scales), np.log(F), 1)
H = C[0]
RegLine = np.polyval(C, np.log(scales))
```

Modify the previous code by adding new fragments:

```
def calc_H(arr, scale, m=1):

    X = np.cumsum(arr - np.mean(arr)) # simulate a random walk (X)
    X = X.T                           # transpose the values of X

    scale = scale
    m = m
    segments = np.zeros(len(scale), dtype=int)
    F = np.zeros(len(scale))

    Index = {}  # dictionary of value indices
```

```
    fit = {}    # dictionary for saving the obtained polynomial curves for ea
ch segment
    RMS = {}    # dictionary of standard deviations

    for ns in range(len(scale)):
        segments[ns] = np.floor(len(X) / scale[ns]).astype(int)
        RMS[ns] = np.zeros(segments[ns])

        for v in range(segments[ns]):         # go through each segment
# determine the initial value of the segment
            Idx_start = v * scale[ns]

# determine the final value
            Idx_stop = (v+1) * scale[ns] if v < segments[ns] - 1 else len(X)


# form an array of indices of the values of the segment under study
            Index[v, ns] = np.arange(Idx_start, Idx_stop)

# remove values by indexes
            X_Idx = X[Index[v, ns]]

# determine the polynomial coefficients of order m
            C = np.polyfit(Index[v, ns], X_Idx, m)

# build a polynomial curve according to the determined coefficients
            fit[v, ns] = np.polyval(C, Index[v, ns])

# estimate the standard deviation for the fragment v on the scale ns
            RMS[ns][v] = np.sqrt(np.mean((X_Idx - fit[v, ns])**2))

# estimate the total standard deviation within the scale ns
        F[ns] = np.sqrt(np.mean(RMS[ns]**2))

# find the coefficients of the equation of the line
    C = np.polyfit(np.log(scale), np.log(F), 1)

# take the slope angle of the line as the Hurst exponent
    H = C[0]

# create the equation itself
    RegLine = np.polyval(C, np.log(scale))

    return H, RegLine, F
```

Now let us consider the dependence of the generalized fluctuation function *F* on different lengths (scales) of local segments of the series for the series we are studying (see Fig. 4.29):

```
scmin = 16
scmax = 1024
scres = 19
exponents = np.linspace(np.log(scmin), np.log(scmax), scres)
```

```
scales_exp = np.round(np.exp(1)**exponents).astype(int)

H_multifrac, RegLine_multifrac, F_multifrac = calc_H(sp_ret, scale=scales_exp
, m=1)
H_monofrac, RegLine_monofrac, F_monofrac = calc_H(pink_noise, scale=scales_ex
p, m=1)
H_white_noise, RegLine_white_noise, F_white_noise = calc_H(white_noise, scale
=scales_exp, m=1)

fig, ax = plt.subplots(1, 1)

ax.set_xscale('log')
ax.set_yscale('log')
ax.scatter(scales_exp, F_multifrac,
           label=fr"Multifractal series ($H$={H_multifrac:.2f})",
           color='darkblue')
plt.plot(scales_exp, np.exp(RegLine_multifrac),  color='darkblue')

ax.scatter(scales_exp, F_monofrac,
           label=fr"Monofractal series ($H$={H_monofrac:.2f})",
           color='magenta')
plt.plot(scales_exp, np.exp(RegLine_monofrac), color='magenta')


ax.scatter(scales_exp, F_white_noise,
           label=fr" White noise ($H$={H_white_noise:.2f})",
           color='red')
plt.plot(scales_exp, np.exp(RegLine_white_noise), color='red')

ax.set_xlabel(r'$\log{ns}$')
ax.set_ylabel(r"$\log{F(ns)}$")

plt.legend(fontsize=16)

fig.tight_layout()
plt.show();
```

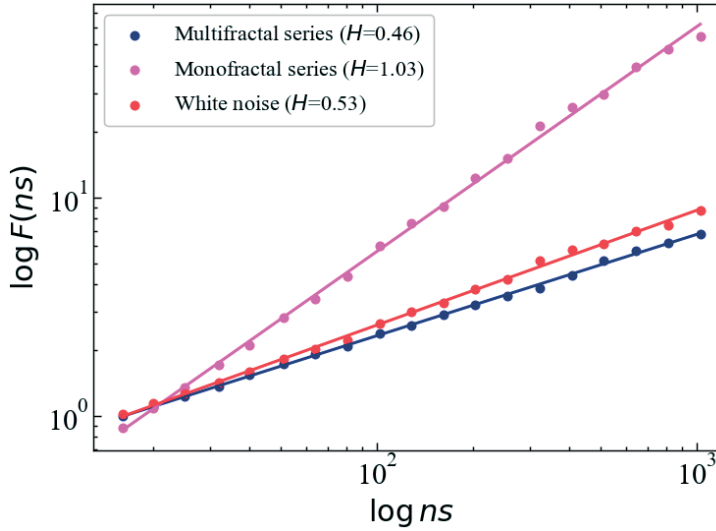


Fig. 4.29: Plot of the dependence of the total standard deviation (i.e., the fluctuation function *F*) on scale. The scale-invariant dependence is indicated by the slope *H* of the regression lines (Hurst exponent)

The Hurst exponent determines the monofractal structure of the time series by indicating how fast the total standard deviation *F* of the local RMS fluctuations increases with the size of the local segments of the series (i.e., the scale). Fig. 4.29 shows that the total RMS value of the local fluctuations *F* compared to S&P 500 index and white noise increases faster with the size of the sample segments for monofractal pink noise.

Fig. 4.30 illustrates that the Hurst exponent defines a continuum between noise-like time series and random walk-like time series. The Hurst exponent is in the range from 0 to 1 for noisy time series, while it is greater than 1 for random walk-like time series. The time series has a long-term dependent (i.e. correlated) structure when the Hurst exponent is in the range 0.5-1, and an anticorrelated structure when the Hurst exponent is in the range 0-0.5. The time series has an independent or short-term dependent structure in the special case when the Hurst exponent is 0.5. According to the previous figure, the time series of white noise

appears to be unpredictable as the Hurst exponent is close to 0.5, while pink noise has a long-term dependent structure with the Hurst exponent close to 1 and S&P 500 index demonstrates more antipersistant dynamics.

```
betas = np.linspace(0.0, 2.0, 12)[::-1]
scmin = 16
scmax = 1024
scres = 19
exponents = np.linspace(np.log(scmin), np.log(scmax), scres)
scales_exp = np.round(np.exp(1)**exponents).astype(int)

color = iter(plt.cm.rainbow(np.linspace(0, 1, len(betas))))

fig, ax = plt.subplots(len(betas), 1, sharex=True)

for idx, beta in enumerate(betas):

    noise = nk.signal_noise(duration=sp_length,  # generate noise with differ
ent beta values
                            sampling_rate=1,
                            beta=beta,
                            random_state=123)

    H_noise, _, _ = calc_H(arr=noise, scale=scales_exp, m=1)

    c = next(color)
    ax[idx].plot(np.arange(len(noise)), noise, label=fr"$H$ = {H_noise:.2f}",
 c=c)
    ax[idx].legend(loc="upper right", fontsize=12)
    ax[idx].margins(x=0)

fig.subplots_adjust(hspace=0)

plt.show();
```

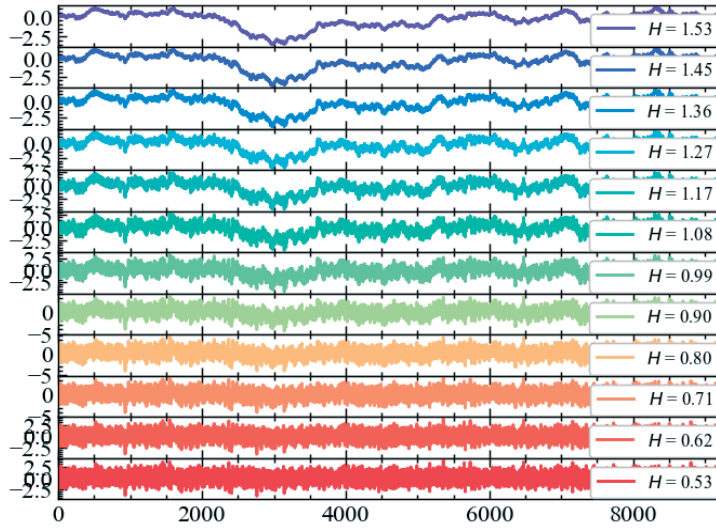


Fig. 4.30: The range of Hurst exponent values defines a continuum of fractal structures between white noise ($H = 0.5$) and brown noise ($H = 1.5$). The pink noise $H = 1$ separates the noise $H < 1$, which has more noticeable fast fluctuations, and the random walks $H > 1$, which have more noticeable slow fluctuations

4.9.6 Multifractal detrended fluctuation analysis

The structures of monofractal and multifractal time series are different, although they have similar overall `RMS` values. Multifractal time series contain local fluctuations with both extremely small and extremely large values, which is not typical for monofractal time series. The absence of fluctuations with extremely large and small values leads to a normal distribution for a monofractal time series, where the variation is described only by the second-order statistical moment (variance). Thus, monofractal DFA is based on the second-order statistic of the total standard deviation (i.e., F). In a multifractal time series, the local fluctuations, `RMS[ns][v]`, will be extremely large for segments v within time periods of large fluctuations and extremely small for segments v within time periods of small fluctuations. Therefore, multifractal time series are not normally distributed and all q-order statistical moments should be taken into account. Thus, it is necessary to

extend the total RMS value of the monofractal DFA to the q-order root mean square fluctuation function of the multifractal DFA (F_q):

```
def calc_Fq(arr, scale, q, m=1):

    X = np.cumsum(arr - np.mean(arr)) # simulate a random walk (X)
    X = X.T                           # transpose the values of X

    scale = scale
    qs = q
    m = m
    segments = np.zeros(len(scale), dtype=int)
    Fq = np.zeros((len(qs), len(scale)))
    Index = {}
    RMS = {}    # dictionary of local standard deviations
    fit = {}    # a dictionary for saving the obtained polynomial curves for
each segment
    qRMS = {}   # is a dictionary of local deviations weighted by q

    for ns in range(len(scale)):
        segments[ns] = np.floor(len(X) / scale[ns]).astype(int)
        RMS[ns] = np.zeros(segments[ns])

# go through each segment
        for v in range(segments[ns]):

# determine the initial value of the segment
            Idx_start = v * scale[ns]

# determine the final value
            Idx_stop = (v+1) * scale[ns] if v < segments[ns] - 1 else len(X)


# form an array of indices of the values of the segment under study
            Index[v] = np.arange(Idx_start, Idx_stop)

# remove values by indexes
            X_Idx = X[Index[v]]

# determine the polynomial coefficients of order m
            C = np.polyfit(Index[v], X_Idx, m)

# build a polynomial curve according to the determined coefficients
            fit = np.polyval(C, Index[v])

# estimate the standard deviation for the fragment v on the scale ns
            RMS[ns][v] = np.sqrt(np.mean((X_Idx - fit)**2))

# convert q values to the float type
        qs = np.asarray_chkfinite(qs, dtype=float)

# for multifractality
# ---------------------------
      for nq, qval in enumerate(qs):
```

```
            if (qval !=0.):
                qRMS[nq, ns] = RMS[ns] ** q[nq]
                Fq[nq, ns] = np.mean(qRMS[nq, ns]) ** (1/ q[nq])
            else:
                Fq[nq, ns] = np.exp(0.5 * np.mean(np.log(RMS[ns] **2)))
        # --------------------------

    return Fq, qRMS, Index
```

The new code block starts a loop that calculates the total root mean square value of the q-order $F_q(nq)$ from negative to positive q. The q-order weighs the influence of segments of the series with large and small fluctuations, `RMS`, as shown in the following figure. For negative q, the value of $F_q(nq)$ is influenced by segments of v with small `RMS(v)`. On the contrary, $F_q(nq)$ for positive q is affected by segments of v with large `RMS(v)`. Local `RMS` fluctuations with large and small magnitudes are classified by the magnitude of negative or positive order q, respectively. The F_q for $q = -3$ and 3 is more affected by the segments of v with the smallest and largest `RMS(v)`, respectively, compared to the F_q for $q = -1$ and 1. The midpoint $q = 0$ is neutral with respect to the influence of segments with small and large RMS. Note that the last line of code in the new section redefines the special case $q(nq) = 0$, since $1/0$ goes to infinity (i.e., $1/q(q = 0) = \infty$). The reader should also note that $F_q[q == 2]$ is equal to the second-order statistic F, since $\sqrt{x} = x^{1/2}$. Monofractal DFA is now extended to MF-DFA (see Fig. 4.31).

```
scales = np.array([32])
nq = np.array([-3, -1, 1, 3])

Fq, qRMS, Index = calc_Fq(sp_ret, scale=scales, q=nq, m=1)
Fq_pink, qRMS_pink, Index = calc_Fq(pink_noise, scale=scales, q=nq, m=1)

fig, ax = plt.subplots((len(nq)+1), 1, sharex=True)

ax[0].plot(time_ser.index[1:], sp_ret, label="Multifractal")
ax[0].plot(time_ser.index[1:], pink_noise, label="Monofractal")
ax[0].grid(False)
ax[0].margins(x=0)
ax[0].legend(loc='upper left', fontsize=12)
ax[0].get_xaxis().set_visible(False)


for idx inrange(1, len(nq)+1):
```

```
    l = [Index[val] for val in Index.keys()]

    x = np.array([])
    for v in l:
        x = np.concatenate([x, v])

    y = np.array([])
    y_pink = np.array([])
    for i, v inenumerate(l):
        y = np.concatenate([y, qRMS[(idx-1, 0)][i]*np.ones(len(v))])
        y_pink = np.concatenate([y_pink, qRMS_pink[(idx-1, 0)][i]*np.ones(len
(v))])

    ax[idx].set_title(fr"Local variations for scale {scales[0]} with $q=${nq[
idx-1]}", fontsize=14)
    ax[idx].plot(time_ser.index[1:], y)
    ax[idx].plot(time_ser.index[1:], y_pink)
    ax[idx].margins(x=0)

handles, labels = ax[0].get_legend_handles_labels()

fig.tight_layout(pad=0.01)
plt.show();
```

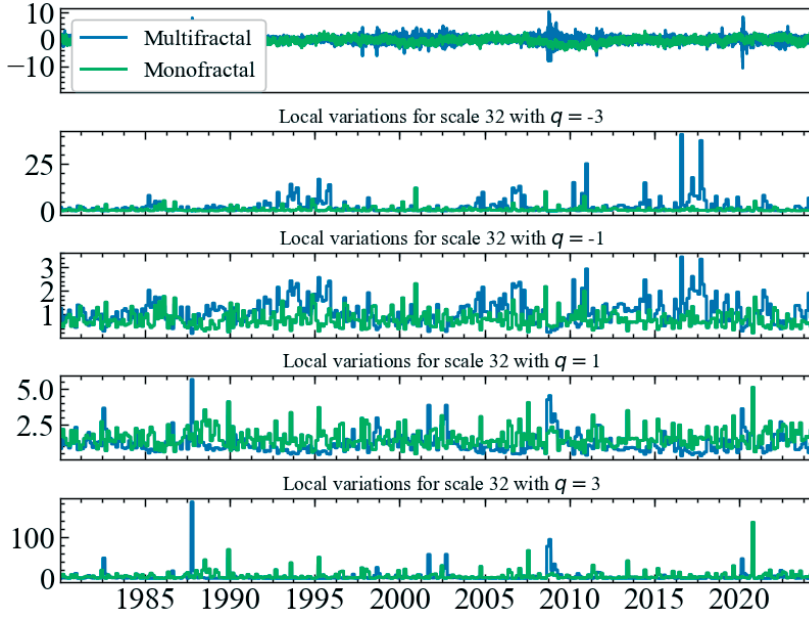


Fig. 4.31: Illustration of the dependence of local fluctuations of qRMS on q at a scale of 32

The qRMS in Fig. 4.31 is the q-order of the local fluctuations (i.e., RMS) and is a component of the overall q-order of the RMS (i.e., F_q). The qRMS is presented for monofractal (green bar) and multifractal (blue bar) time series. A negative order of q ($q = -3$ and -1) enhances segments in the multifractal time series with extremely small RMS, while a positive order of q ($q = 3$ and 1) enhances segments with extremely large RMS. Note that $q = -3$ and $q = 3$ amplify small and large variation, respectively, more than $q = -1$ and $q = 1$. Note also that a monofractal time series has no segments with extremely large or small variations and thus no peaks in the qRMS. The q-order total root mean square deviation is able to distinguish between the structure of small and large fluctuations and, accordingly, monofractal and multifractal time series.

Now we can define the q-order Hurst exponents as the slopes $h(q)$ of the regression lines for each q-order RMS value of F_q. Both $h(q)$ and the regression line are determined in a loop for each q-order:

```
def calc_Hq(arr, scale, q, m=1):

    X = np.cumsum(arr - np.mean(arr)) # simulate a random walk (X)
    X = X.T                           # transpose the values of X

    scale = scale
    qs = q
    m = m
    segments = np.zeros(len(scale), dtype=int)
    Fq = np.zeros((len(qs), len(scale)))       # an array to store the genera
l fluctuation function
    hq = np.zeros(len(qs), dtype=float)        # is an array for Hurst expone
nts of the qth order
    qRegLine = {} # a dictionary for saving regression lines
    Index = {}    # a dictionary for storing serial segment indices
    RMS = {}      # dictionary of local standard deviations
    fit = {}      # a dictionary for saving the obtained polynomial curves fo
r each segment
    qRMS = {}     # is a dictionary of local deviations weighted by q

    for ns in range(len(scale)):
        segments[ns] = np.floor(len(X) / scale[ns]).astype(int)
        RMS[ns] = np.zeros(segments[ns])

# go through each segment
        for v inrange(segments[ns]):

# determine the initial value of the segment
```

```
            Idx_start = v * scale[ns]

# determine the final value
            Idx_stop = (v+1) * scale[ns] if v < segments[ns] - 1 else len(X)


# form an array of indices of the values of the segment under study
            Index[v] = np.arange(Idx_start, Idx_stop)

# get values by indexes
            X_Idx = X[Index[v]]

# determine the polynomial coefficients of order m
            C = np.polyfit(Index[v], X_Idx, m)

# build a polynomial curve according to the determined coefficients
            fit = np.polyval(C, Index[v])

# estimate the standard deviation for the fragment v on the scale ns
            RMS[ns][v] = np.sqrt(np.mean((X_Idx - fit) **2))

# convert q values to the float type
        qs = np.asarray_chkfinite(qs, dtype=float)

# for multifractality
# ---------------------------
        for nq, qval in enumerate(qs):
            if (qval !=0.):
                qRMS[nq, ns] = RMS[ns] ** q[nq]
                Fq[nq, ns] = np.mean(qRMS[nq, ns]) ** (1/ q[nq])
            else:
                Fq[nq, ns] = np.exp(0.5 * np.mean(np.log(RMS[ns] ** 2)))

        for nq, _ in enumerate(qs):
            # if the fluctuation is equal to. 0, log2 will collide with divis
ion by 0
            old_setting = np.seterr(divide="ignore", invalid="ignore")
            C = np.polyfit(np.log(scale), np.log(Fq[nq, :]), m)
            np.seterr(**old_setting)
            hq[nq] = C[0]
            qRegLine[nq] = np.polyval(C, np.log(scale))
        # ---------------------------

    return hq, qRegLine, Fq

scmin = 16
scmax = 1024
scres = 19

q_min = -5.0
q_max = 5.0
q_step = 0.1

nq = np.arange(q_min, q_max+q_step, q_step)
```

```
exponents = np.linspace(np.log(scmin), np.log(scmax), scres)
scales_exp = np.round(np.exp(1)**exponents).astype(int)

Hq_multifrac, qRegLine_multifrac, Fq_multifrac = calc_Hq(sp_ret, scale=scales
_exp, q=nq, m=1)
Hq_monofrac, qRegLine_monofrac, Fq_monofrac = calc_Hq(pink_noise, scale=scale
s_exp, q=nq, m=1)
Hq_white_noise, qRegLine_white_noise, Fq_white_noise = calc_Hq(white_noise, s
cale=scales_exp, q=nq, m=1)

fig, ax = plt.subplots(2, 2)

ax[0][0].set_title("Multifractal")
ax[0][0].set_xlabel(r"$ns$")
ax[0][0].set_ylabel(r"$F_{q}(ns)$")
ax[0][0].set_xscale('log')
ax[0][0].set_yscale('log')
for i in range(len(nq)):
    ax[0][0].scatter(scales_exp, Fq_multifrac[i, :], color='darkblue')
    ax[0][0].plot(scales_exp, np.exp(qRegLine_multifrac[i]),  color='darkblue
')

ax[0][1].set_title("Monofractal")
ax[0][1].set_xlabel(r"$ns$")
ax[0][1].set_xscale('log')
ax[0][1].set_yscale('log')
for i in range(len(nq)):
    ax[0][1].scatter(scales_exp, Fq_monofrac[i, :], color='magenta')
    ax[0][1].plot(scales_exp, np.exp(qRegLine_monofrac[i]),  color='magenta')

ax[1][0].set_title("White noise")
ax[1][0].set_xlabel(r"$ns$")
ax[1][0].set_ylabel(r"$F_{q}(ns)$")
ax[1][0].set_xscale('log')
ax[1][0].set_yscale('log')
for i in range(len(nq)):
    ax[1][0].scatter(scales_exp, Fq_white_noise[i, :], color='red')
    ax[1][0].plot(scales_exp, np.exp(qRegLine_white_noise[i]),  color='red')

ax[1][1].set_title(r"Hurst exponent of $q$th order")
ax[1][1].set_xlabel(r"$q$")
ax[1][1].set_ylabel(r"$h(q)$")
ax[1][1].plot(nq, Hq_multifrac, linestyle='-', marker='o', label="Multifract
al", color='darkblue')
ax[1][1].plot(nq, Hq_monofrac,linestyle='-', marker='o', label="Monofractal"
, color='magenta')
ax[1][1].plot(nq, Hq_white_noise, linestyle='-', marker='o', label="White no
ise", color='red')
ax[1][1].legend(loc='center right', fontsize=12)

fig.tight_layout(pad=0.1)
plt.show();
```

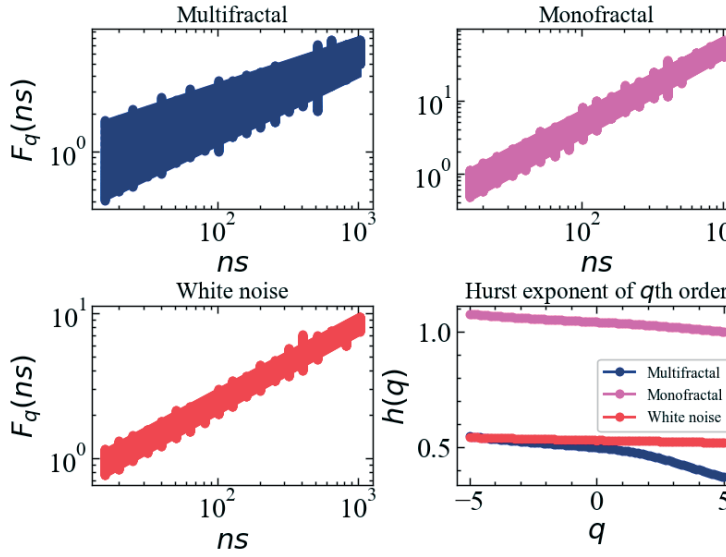


Fig. 4.32: RMS values of F_q for different q-orders and corresponding regression lines calculated by MF-DFA for multifractal, monofractal, and white noise

We can see that the generalized fluctuation function for the multifractal depends not only on the scale, but also on q, as demonstrated by the different slopes of the regression lines $h(q)$. The scaling generalized fluctuation functions F_q for the monofractal and white noise are q-independent, since their regression lines for different scales have the same slope. The q-order Hurst exponent $h(q)$ for the multifractal series (blue line) appears to be independent for $q < 0$ and variable for $q > 0$. This indicates that the source of multifractality of S&P 500 is abnormally large fluctuations, such as the coronavirus pandemic crisis. For the monofractal (pink line) and white noise (red line), $h(q)$ remain constant.

4.9.7 Multifractal spectrum of time series

The q-order Hurst exponent $h(q)$ is just one of several types of scaling measures used to parameterize the multifractal structure of time series. As presented earlier, we can derive a q-order mass index $(\tau(q))$, and then use $\tau(q)$ to

obtain a q-order singularity index $(\alpha(q))$ and a fractal dimension $(f(\alpha))$ of fluctuations (regions) with a degree of singularity $\alpha(q)$. The graph of $\alpha(q)$ versus $f(\alpha)$ represents the multifractal spectrum (see Fig. 4.33). The mass, singularity, and fractality indices can be calculated according to the code below:

```
tau_multifrac = nq * Hq_multifrac - 1
tau_monofrac = nq * Hq_monofrac - 1
tau_white_noise = nq * Hq_white_noise - 1

alpha_multifrac = np.gradient(tau_multifrac, nq)
alpha_monofrac = np.gradient(tau_monofrac, nq)
alpha_white_noise = np.gradient(tau_white_noise, nq)

f_multifrac = nq * alpha_multifrac - tau_multifrac
f_monofrac = nq * alpha_monofrac - tau_monofrac
f_white_noise = nq * alpha_white_noise - tau_white_noise

fig, ax = plt.subplots(1, 3)

ax[0].set_xlabel(r"$q$")
ax[0].set_ylabel(r"$\tau(q)$")
ax[0].plot(nq, tau_multifrac, linestyle='-', marker='o', label="Multifractal
", color='darkblue')
ax[0].plot(nq, tau_monofrac, linestyle='-', marker='o', label="Monofractal",
color='magenta')
ax[0].plot(nq, tau_white_noise, linestyle='-', marker='o', label="White noise
", color='red')
ax[0].legend()

ax[1].set_xlabel(r"$\alpha$")
ax[1].set_ylabel(r"$f(\alpha)$")
ax[1].plot(alpha_multifrac, f_multifrac, linestyle='-', marker='o', label="M
ultifractal", color='darkblue')
ax[1].plot(alpha_monofrac, f_monofrac, linestyle='-', marker='o', label="Mono
fractal", color='magenta')
ax[1].plot(alpha_white_noise, f_white_noise, linestyle='-', marker='o', label
="White noise", color='red')

ax[2].set_xlabel(r"$q$")
ax[2].set_ylabel(r"$f(\alpha)$")
ax[2].plot(nq, f_multifrac, linestyle='-', marker='o', label="Multifractal",
 color='darkblue')
ax[2].plot(nq, f_monofrac, linestyle='-', marker='o', label="Monofractal", co
lor='magenta')
ax[2].plot(nq, f_white_noise, linestyle='-', marker='o', label="White noise",
 color='red')

fig.tight_layout(pad=0.01)
plt.show();
```

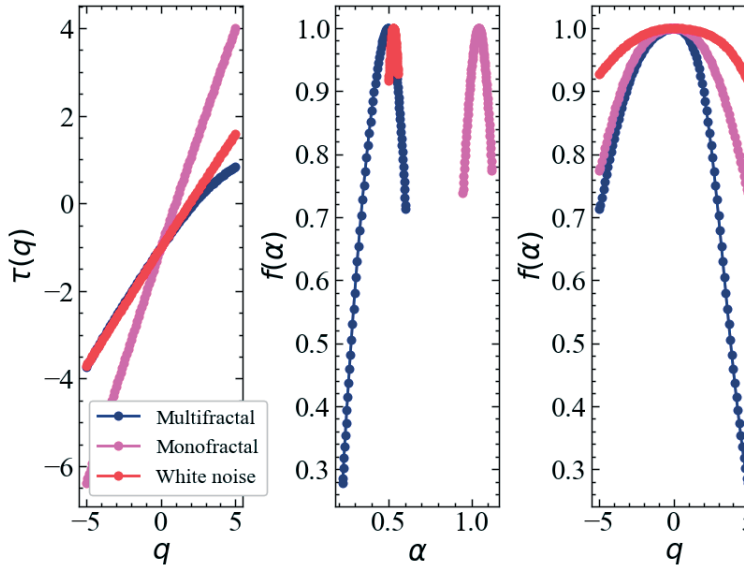


Fig. 4.33: Multiple representation of the multifractal spectrum for multifractal, monofractal, and white noise

The singularity indices α for large highly concentrated fluctuations are small and located in the left tail of the spectrum, while α for small fluctuations are large and located in the right tail of the spectrum.

Thus, the strength of multifractality is described by a large deviation of the local singularity exponent α from the central tendency $\alpha(0)$. A monofractal signal is the case when α remains almost constant, and in some cases, the multifractal spectrum reduces to a single point at a given α.

The range of α indicates the variety of singularity exponents that describe the dynamics of the system, and the value of $f(\alpha)$ indicates the contribution of elements with the corresponding α.

The multifractal spectrum can be characterized by different widths, which indicates the variability of processes occurring within the system. It can also be either symmetrical or asymmetrical. The asymmetry can be both right- and left-handed, indicating different degrees of influence of highly concentrated and low-

concentrated elements (fluctuations). A multifractal spectrum will have a long left tail when the time series has a multifractal structure that is sensitive to local fluctuations with large amplitudes.

On the contrary, a multifractal spectrum will have a long right tail when it is sensitive to local fluctuations with small amplitudes.

Let's illustrate the dependence of the width of the multifractal spectrum on the level of fluctuations in the series. We will demonstrate this dependence on the example of series distributed according to the alpha-stable Levy distribution. To generate random variables from this distribution, we will use the `scipy.stats` module. From it, we import the `levy_stable` class to use the `rvs()` method. The method takes an indicator α, which is responsible for the frequency of events that fall outside the normal distribution. Consider the range of such α values and the spectra of the generated series in Fig. 4.34.

```
from scipy.stats import levy_stable

alphas = np.linspace(1.5, 2.0, 7)
scmin = 16
scmax = 1024
scres = 19

q_min = -5.0
q_max = 5.0
q_step = 0.1
nq_levy = np.arange(q_min, q_max+q_step, q_step)

exponents = np.linspace(np.log(scmin), np.log(scmax), scres)
scales_exp = np.round(np.exp(1)**exponents).astype(int)

color = iter(plt.cm.plasma(np.linspace(0, 0.8, len(alphas))))

fig = plt.figure()
subfigs = fig.subfigures(1, 2)
ax1 = subfigs[0].subplots(len(alphas), 1, sharex=True)
ax2 = subfigs[1].subplots(1, 1)

for i in range(len(alphas)):

# generate an alpha-stable process
    r = levy_stable.rvs(alpha=alphas[i], beta=0, loc=0,
                        scale=1, size=len(sp_ret), random_state=123)

    Hq_levy, qRegLine_levy, Fq_levy = calc_Hq(r, scale=scales_exp, q=nq_levy,
 m=1)
    tau_levy = nq_levy * Hq_levy - 1
```

```
    alpha_levy = np.gradient(tau_levy, nq_levy)
    f_levy = nq_levy * alpha_levy - tau_levy

    c = next(color)
    ax1[i].plot(np.arange(len(r)), r, label=fr'$\alpha$={alphas[i]:.2f}', c=c
)
    ax1[i].margins(x=0)
    ax1[i].legend(loc="upper left", fontsize=12)
    ax2.plot(alpha_levy, f_levy, marker='o', c=c)

ax1[0].set_title("Multifractal time series", fontsize=16)
ax1[-1].set_xlabel("Time (ordinal number)")
ax1[len(alphas)//2].set_ylabel('Oscillation amplitude')

ax2.set_title("Multifractal spectra", fontsize=16)
ax2.set_xlabel(r"$\alpha$")
ax2.set_ylabel(r"$f(\alpha)$")

fig.subplots_adjust(hspace=0.1)

plt.show();
```

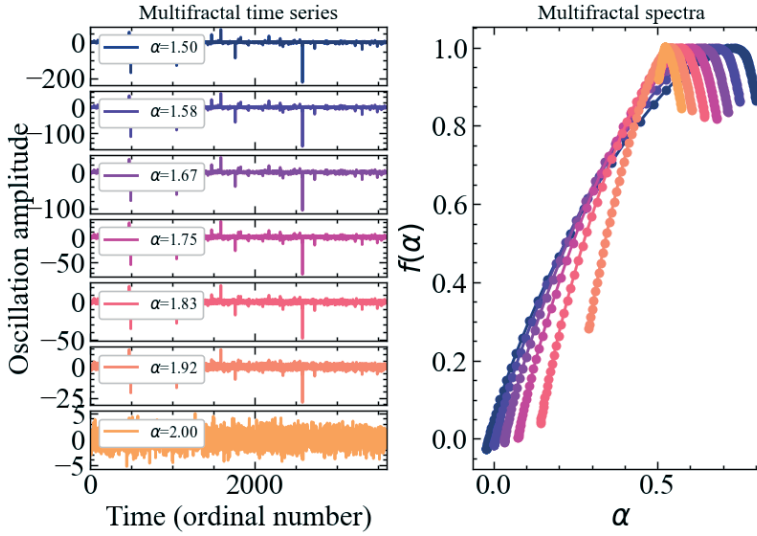


Fig. 4.34: Illustration of a set of multifractal time series (Levy of alpha stable processes) and their multifractal spectra generated with different values of α. Note the growth of structural differences between periods with small and large fluctuations with increasing width of the multifractal spectrum

A system whose complexity is caused by highly concentrated elements will have a clearly defined left-handed spectrum. The complexity of the system caused

by weakly concentrated elements is characterized by the right tail of the multifractal spectrum. If the complexity of the system develops due to elements of two types, then the spectrum will appear symmetrical, where the elements of two types will be equally probable. For the alpha-stable processes generated by Levy above, it can be seen that the lower the value of α, the stronger the dominance of highly concentrated (large) fluctuations. At $\alpha = 2.0$, the spectrum is increasingly narrowed to a singular point.

Further, it will be shown that for the resulting multifractal parabola of the multifractal spectrum, the values of both the entire **spectral width** ($\Delta\alpha$) and its **right** and **left tails** (R and L) can be calculated. It is also possible to calculate the **value of the singularity**, where $f(\alpha)$ takes the maximum value α_0, and even the so-called "**asymmetry**" of this spectrum (Δf). Fig. 4.35 schematically shows the position of the key indicators of multifractal spectrum.

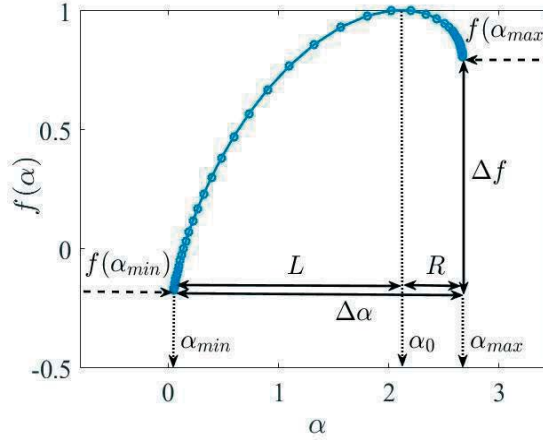


Fig. 4.35: Graph of the multifractal spectrum with the values of the multifractal spectrum width ($\Delta\alpha$), the values of the minimum, central, and maximum singularity ($\alpha_{min}, \alpha_0, \alpha_{max}$), the width of the left and right tails of the spectrum L, R), and the difference between the fractal dimensions at the ends of the parabola (Δf)

It is also worth noting that this diagram does not represent an exhaustive list of system multifractality indicators that we will use in the future, but should provide an intuitive understanding of how most multifractal indicators are derived.

4.9.8 Generalized fractal dimensions

Along with the multifractal spectrum, it will be useful to consider the spectrum of generalized fractal dimensions, or in other words, **Renyi dimensions**, since they also have information-theoretical significance. Let us find out the physical meaning of generalized fractal dimensions for some values of q. When $q = 0$, $Z(0, \varepsilon) = N(\varepsilon)$. On the other hand, we can define that $Z(0, \varepsilon) \approx \varepsilon^{\tau(0)} = \varepsilon^{-D_0}$. Comparing these inequalities, we can come to the ratio $N(\varepsilon) \approx \varepsilon^{-D_0}$. Thus, D_0 is the usual Hausdorff dimension of the set Ω. It also corresponds to the maximum of the multifractal spectrum, $f(\alpha)$, which is always equal to one for a one-dimensional signal. For crisis recognition tasks, this characteristic is the coarsest and does not provide information about the statistical properties of the system.

Now let's find out the meaning of D_1. Since the statistical sum $Z(1, \varepsilon) = 1$ when $q = 1$, $\tau(1) = 0$. Thus, we have uncertainty when $D_1 = \tau(1)/(1 - 1)$. Let's reveal this uncertainty using the following equation:

$$Z(q, \varepsilon) = \sum_{i=1}^{N(\varepsilon)} p_i^q = \sum_{i=1}^{N(\varepsilon)} p_i \exp[(q - 1) \ln p_i].$$

Now, setting $q \to 1$, expanding the exponent, and taking into account the condition for normalizing the probabilities of p_i, we obtain

$$Z(q \to 1, \varepsilon) \approx \sum_{i=1}^{N(\varepsilon)} [p_i + (q - 1) p_i \ln p_i] = 1 + (q - 1) \sum_{i=1}^{N(\varepsilon)} p_i \ln p_i.$$

As a result, we derive the following expression:

$$D_1 = \lim_{\varepsilon \to 0} \sum_{i=1}^{N(\varepsilon)} p_i \ln p_i \Big/ \ln \varepsilon.$$

The numerator in this formula is the **information entropy** of the fractal set $S(\varepsilon)$:

$$S(\varepsilon) = -\sum_{i=1}^{N(\varepsilon)} p_i \ln p_i.$$

Thus, the resulting value of the generalized fractal dimension D_1 is related to the entropy $S(\varepsilon)$ by the following relation:

$$D_1 = -\lim_{\varepsilon \to 0} S(\varepsilon)/\ln \varepsilon.$$

Returning to the problem of distributing points on a fractal set Ω, we can say that since $S(\varepsilon) \approx \varepsilon^{-D_1}$, the value of D_1 characterizes the information needed to describe the position of a point in a certain cell.

Additional information on the information dimension

Information dimension can be used to describe the spatial heterogeneity of a system. The more homogeneous the attractor is, the higher this indicator should be. That is, the more configurations the elements of a given system can take, the more information we need to account for each element. With spatial homogeneity, the information entropy also increases, which links the information dimension to the concept of entropy. Since D_1 is the tangent of the slope of the regression line plotted against the entropy and the radius of the circles in which the frequency of hits of individual attractor elements is measured, we can say that the information dimension reflects the rate of change of the information entropy. The higher D_1 is, the faster the entropy grows – a measure of our current ignorance about the system. The lower D_1, the lower the entropy itself. In other words, the greater the spatial asymmetry, the more ordered the complexity, the higher our knowledge of the current state of the system, and the less information we need to describe the configurations that the system can take

For the generalized fractal dimension at $q = 2$, the following expression is valid:

$$D_2 = \lim_{\varepsilon \to 0} \ln \sum_{i=1}^{N(\varepsilon)} p_i^2 \Big/ \ln \varepsilon.$$

The value p_i represents the probability of a point falling into a cell of size ε. Then the value p_i^2 is the probability of two points hitting this cell. Finding the sum of p_i^2 over all occupied cells, we get the probability that two randomly selected points from the set Ω are inside the same cell of size ε. Thus, the distance between these two points will be less than or of the order of ε. The probability of finding two trajectories within a neighborhood of radius ε can be found using the correlation integral.

In this case, we conclude that the generalized dimension determines the dependence of the correlation integral $C(\varepsilon)$ on ε. For this reason, D_2 is referred to in the literature as the **correlation dimension**.

Now let's analyze the behavior of $f(\alpha)$. The value of the function at the maximum can be easily determined by using the expression (4.14), where $\tau(q) = q\alpha(q) - f\big(\alpha(q)\big)$ or $(q-1)D_q = q\alpha(q) - f\big(\alpha(q)\big)$. When $q = 0$, we obtain that $f(\alpha_0) = D_0$, i.e., the maximum value of the spectrum is equal to the Hausdorff dimension.

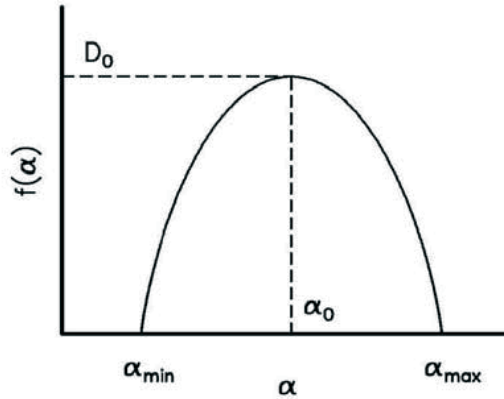


Fig. 4.36: The maximum of the function $f(\alpha)$ is equal to the fractal dimension D_0

Consider the case when $q = 1$. Since $\tau(1) = 0$, it follows from the equation above that $\alpha(1) = f\big(\alpha(1)\big)$. On the other hand, we know that since $q = df(\alpha)/d\alpha$, the derivative of $f(\alpha)$ at this point is 1. Differentiate the relation $\tau(q) = (q-1)D_q$ with respect to q,

$$\frac{d\tau}{dq} = D_q + (q-1)D_q' = \alpha(q),$$

and assuming that $q = 1$, we get that $\alpha(1) = D_1$. Thus, we have $D_1 = \alpha(1) = f\big(\alpha(1)\big)$. Thus, the information dimension D_1 lies on the curve $f(\alpha)$ at the point where $\alpha = f(\alpha)$ and $f'(\alpha) = 1$.

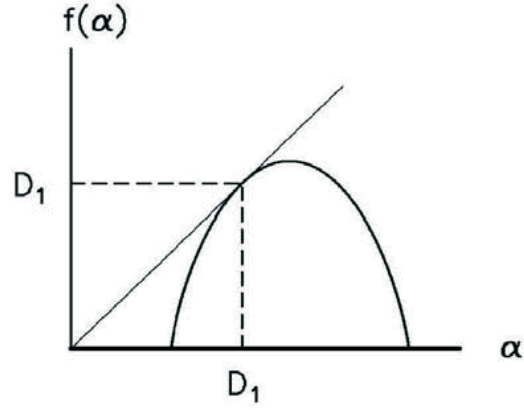


Fig. 4.37: Position of the information dimension D_1: $D_1 = \alpha = f(\alpha)$

Now consider the case when $q = 2$. Using the previous formula, we obtain that $D_2 = 2\alpha(2) - f\big(\alpha(2)\big)$ or $f\big(\alpha(2)\big) = 2\alpha(2) - D_2$.

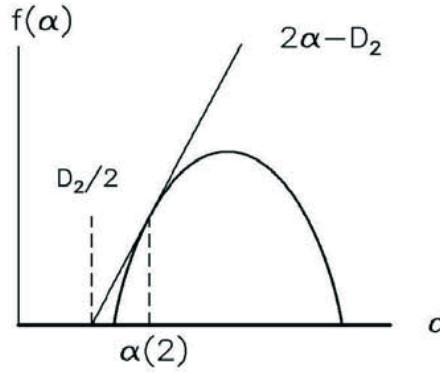


Fig. 4.38: Geometric definition of the correlation dimension D_2

Next, let us consider the dependence of the generalized fractal dimension D_q on different values of q for a multifractal series, monofractal, and white noise (see Fig. 4.39).

```
difference_zero = np.absolute(nq-0)
idx_zero = difference_zero.argmin()

difference_one = np.absolute(nq-1)
idx_one = difference_one.argmin()

difference_two = np.absolute(nq-2)
idx_two = difference_two.argmin()

# initialize arrays for the dimensions
Dq_multifrac = np.zeros(len(nq))
Dq_monofrac = np.zeros(len(nq))
Dq_white_noise = np.zeros(len(nq))

# We define generalized fractal dimensions where q!=1
Dq_multifrac[nq!=nq[idx_one]] = tau_multifrac[nq!=nq[idx_one]] / (nq[nq!=nq[i
dx_one]]-1)
Dq_monofrac[nq!=nq[idx_one]] = tau_monofrac[nq!=nq[idx_one]] / (nq[nq!=nq[idx
_one]]-1)
Dq_white_noise[nq!=nq[idx_one]] = tau_white_noise[nq!=nq[idx_one]] / (nq[nq!=
nq[idx_one]]-1)

# We define separately the generalized fractal dimensions at q=1
Dq_multifrac[nq==nq[idx_one]] = -tau_multifrac[nq==nq[idx_one]]
Dq_monofrac[nq==nq[idx_one]] = -tau_monofrac[nq==nq[idx_one]]
Dq_white_noise[nq==nq[idx_one]] = -tau_white_noise[nq==nq[idx_one]]
```

```
fig, ax = plt.subplots(1, 1)

ax.plot(nq, Dq_multifrac, linestyle='-', marker='o', label="Multifractal", co
lor='darkblue')
ax.plot(nq, Dq_monofrac, linestyle='-', marker='o', label="Monofractal", colo
r='magenta')
ax.plot(nq, Dq_white_noise, linestyle='-', marker='o', label="White noise", c
olor='red')
ax.set_xlabel(r"$q$")
ax.set_ylabel(r"$D_{q}$")
ax.legend(loc="upper right")

ax.annotate(fr'$D_{0}$={Dq_multifrac[nq==nq[idx_zero]][0]:.2f}',
            xy=(nq[idx_zero], Dq_multifrac[nq==nq[idx_zero]]),
            xytext=(nq[idx_zero]-2, Dq_multifrac[nq==nq[idx_zero]]+2),
            arrowprops=dict(facecolor='black', shrink=0.05), fontsize=16)

ax.annotate(fr'$D_{1}$={Dq_multifrac[nq==nq[idx_one]][0]:.3f}',
            xy=(nq[idx_one], Dq_multifrac[nq==nq[idx_one]]),
            xytext=(nq[idx_one]-3, Dq_multifrac[nq==nq[idx_one]]-1.5),
            arrowprops=dict(facecolor='black', shrink=0.05), fontsize=16)

ax.annotate(fr'$D_{2}$={Dq_multifrac[nq==nq[idx_two]][0]:.2f}',
            xy=(nq[idx_two], Dq_multifrac[nq==nq[idx_two]]),
            xytext=(nq[idx_two], Dq_multifrac[nq==nq[idx_two]]-1.5),
            arrowprops=dict(facecolor='black', shrink=0.05), fontsize=16)

plt.show();
```

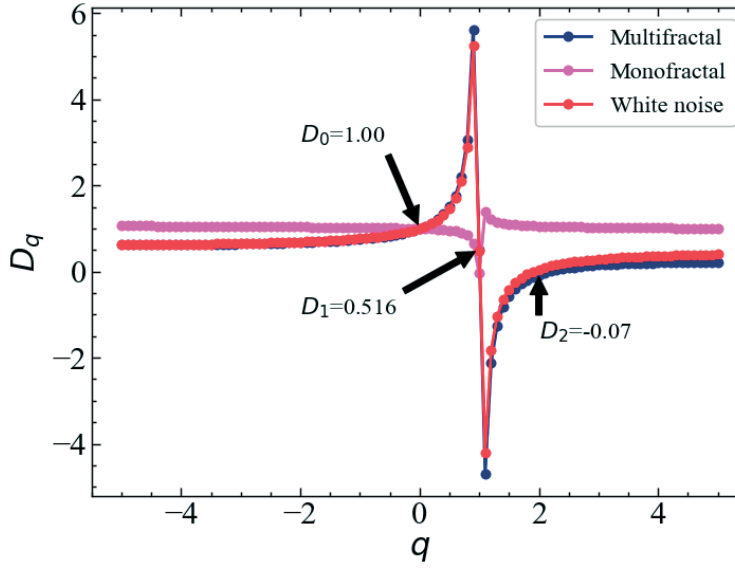


Fig. 4.39: Dependence of generalized fractal dimensions $D(q)$ on q

The figure shows that, first of all, $D_0 = 1$ for all signals, which is consistent with theoretical considerations. The information dimension D_1 for the multifractal and white noise is the same, which may indicate the information content of both signals. For the monofractal, it is close to zero. The correlation dimension D_2 shows that, in general, both S&P 500 and white noise are quite similar: their values appear to be mostly independent of each other. This is in contrast to the conclusions drawn in our previous work, where the approach of D_2 to zero indicated an increase in the degree of correlation of the system. For the monofractal, D_2 is at the level of 1, which indicates a higher degree of correlation in this signal compared to the mono- and multifractals.

4.9.9 Analogies of multifractals with thermodynamics

Using MF-DFA concepts, we can take a fresh look at the time signal as a thermodynamic system. Within the framework of MF-DFA, the mass index $\tau(q)$ can be considered as an analog of free energy, the singularity index α as an analog of internal energy U, and the multifractal spectrum $f(\alpha)$ as entropy. Indeed, the shape of the multifractal spectrum resembles the dependence of the entropy of a thermodynamic system on the energy U. The parameters α_{min} and α_{max} can be characterized as upper and lower limits of the internal energy of the system. The function $Z(q)$ is a formal analog of the partition function $Z(\beta)$ in thermodynamics, where $\beta = 1/T = q$.

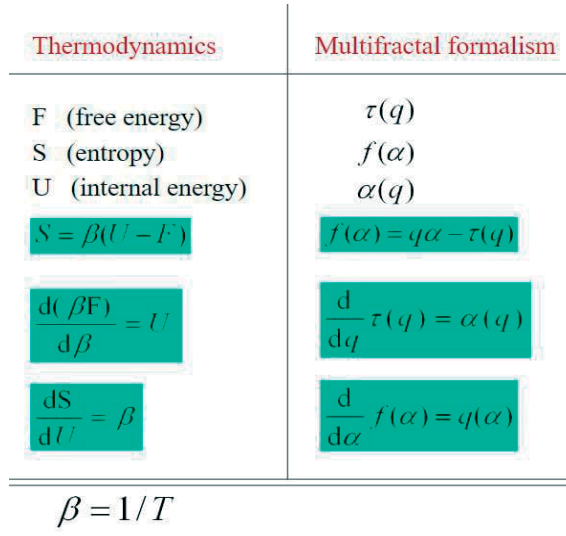


Fig. 4.40: Schematic representation of the analogy of multifractals with the concepts of thermodynamics

In addition, we can calculate the "temperature" of the signal under study. More specifically, the **multifractal heat capacity** $C(q)$ [57] can be defined as

$$C(q) \equiv -(\partial^2 \tau(q)/\partial q^2) = -(\partial \alpha/\partial q) \approx \tau(q+1) - 2\tau(q) + \tau(q-1).$$

The specific heat capacity, as a measure of the rate of energy change, is an indicator of phase transition phenomena. In a thermodynamic system, a phase is characterized by homogeneous physical properties, and a phase transition is a sudden change in certain properties under a critical external condition. The study of phase transitions in the multifractal spectrum has been limited to simple systems, such as the Cantor set and the logistic map. However, our analysis shows the presence of phase transitions [8] in the multifractal spectrum of financial systems. In particular, the "energy" α exhibits significant fluctuations in the neighborhood of q_c, which are reflected in the peak of the specific heat capacity $C(q_c)$ (see Fig. 4.41).

```
C_q_multifrac = -np.gradient(alpha_multifrac, nq, edge_order=2)
C_q_monofrac = -np.gradient(alpha_monofrac, nq, edge_order=2)
C_q_white_noise = -np.gradient(alpha_white_noise, nq, edge_order=2)

fig, ax = plt.subplots(1, 2)

ax[0].plot(nq, C_q_multifrac, linestyle='-', marker='o', label="Multifractal"
, color='darkblue')
ax[0].plot(nq, C_q_monofrac, linestyle='-', marker='o', label="Monofractal",
color='magenta')
ax[0].plot(nq, C_q_white_noise, linestyle='-', marker='o', label="White noise
", color='red')
ax[0].set_xlabel(r"$q$")
ax[0].set_ylabel(r"$C(q)$")
ax[0].legend(loc='center left')

ax[1].plot(alpha_multifrac, C_q_multifrac, linestyle='-', marker='o', label="
Multifractal", color='darkblue')
ax[1].plot(alpha_monofrac, C_q_monofrac, linestyle='-', marker='o', label="Mo
nofractal", color='magenta')
ax[1].plot(alpha_white_noise, C_q_white_noise, linestyle='-', marker='o', lab
el="White noise", color='red')
ax[1].set_xlabel(r"$\alpha$")

fig.tight_layout(pad=0.3)
plt.show();
```

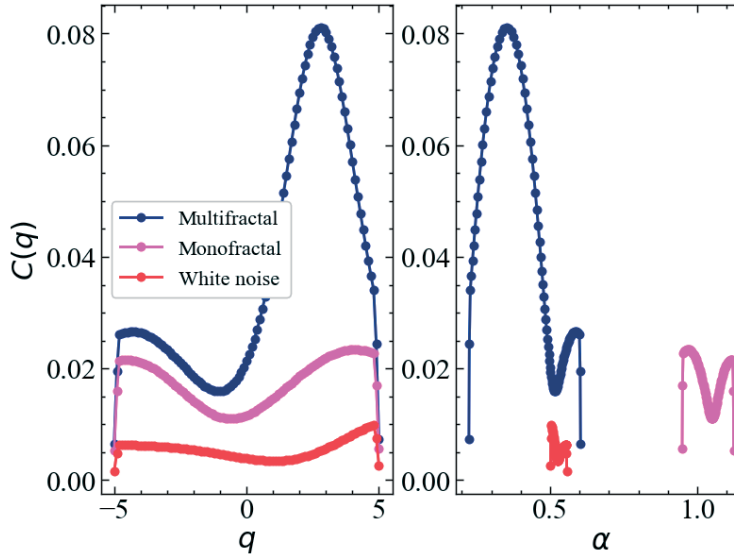


Fig. 4.41: Dependence of the multifractal heat capacity $C(q)$ on q and α

Fig. 4.41 shows that $C(q)$ reaches local and global maxima at positive and negative q values, which indicates that S&P 500 becomes extremely irregular due to the dynamics of both large and small fluctuations during crisis periods, which serve as a quasi-phase transitions of S&P 500.

4.10 MF-DFA empirical results

Of course, a fractal analysis of the entire series is important, but this approach ignores the assumption that both monofractal and multifractal areas exist in the time sequence. That is, it ignores the assumption that the degree of complexity changes over time. Quantitative measures of multifractality calculated within the sliding window approach are the most objective and practical in system analysis. In addition, quantitative measures can be used as indicators or predictors of abnormal phenomena, or as a basis for building another predictive model.

Some charts will present a pair plot of only the time series and the multifractal indicator. Let's use the `plot_pair()` function that we defined in the previously:

```
def plot_pair(x_values,
              y1_values,
              y2_values,
              y1_label,
              y2_label,
              x_label,
              file_name,
              clr="magenta"):

    fig, ax = plt.subplots()

    ax2 = ax.twinx()

    ax2.spines.right.set_position(("axes", 1.03))

    p1, = ax.plot(x_values,
                  y1_values,
"b-", label=fr"{y1_label}")
    p2, = ax2.plot(x_values,
                   y2_values,
                   color=clr,
                   label=y2_label)

    ax.set_xlabel(x_label)
    ax.set_ylabel(f"{y1_label}")
```

```
    ax.yaxis.label.set_color(p1.get_color())
    ax2.yaxis.label.set_color(p2.get_color())

    tkw = dict(size=2, width=1.5)

    ax.tick_params(axis='x', rotation=45, **tkw)
    ax.tick_params(axis='y', colors=p1.get_color(), **tkw)
    ax2.tick_params(axis='y', colors=p2.get_color(), **tkw)


    ax2.legend(handles=[p1, p2])

    plt.savefig(file_name+".jpg")

    plt.show();
```

For further calculations, we will again use the `fathon` library, which we used earlier to perform the classical DFA. The advantages of this particular library are the ability to use the procedure for calculating the division of the series into segments starting from the end of the series, since the length of the series does not always allow us to divide it into local segments as a whole. That is, theoretically, we are left with a segment of the series that cannot be divided into local segments. Therefore, repeating the procedure of dividing into local segments starting from the end of the series allows us to get around this problem. To simplify the presentation of the theoretical material, we did not implement this procedure, but it is available in the `fathon` library. In addition, the library provides the ability to calculate the cross-correlation DFA of and its multifractal analog.

```
import fathon
from fathon import fathonUtils as fu
```

Let's start initializing the parameters.

```
window = 500 # sliding window width
tstep = 5    # sliding window time step
ret_type = 4 # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

win_beg = 10       # Initial segment width
win_end = window-1 # Final segment width
```

```
scales_exp_wind = fu.linRangeByStep(win_beg, win_end) # generate an array
# of linearly separated elements

rev = True # whether to repeat the calculation of the fluctuation function fr
om the end

length = len(time_ser.values)

q_min = -5  # minimum q value
q_max = 5   # maximum q value
q_step = 1  # incremental step of q

nq = np.arange(q_min,
               q_max+q_step,
               q_step)

order = 3 # order of the polynomial trend

delta_alph = []
delta_spec = []
max_alph = []
min_alph = []
mean_alph = []
alpha_zero = []
delta_alph_right = []
delta_alph_left = []
assym = []
delta_s = []
D_0 = []
D_1 = []
D_2 = []
D_left = []
D_right = []
C_q = []
h_q = []
tau_q = []
D_q = []
mfSpect = []
alpha = []
hFI = []
alphaCF = []
C_q_area_wind = []
```

Let's start the rolling window procedure, which will combine the previous stages of calculations:

```
for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()

    fragm = transformation(fragm, ret_type)

#  finding a cumulative series
    cumulative = fu.toAggregated(fragm)
```

```
# initialization of the MF-DFA procedure
    pymfdfa = fathon.MFDFA(cumulative)

# calculation of the fluctuation function and obtaining the generalized Hurst
 exponent
    n, F = pymfdfa.computeFlucVec(scales_exp_wind, nq, revSeg=rev, polOrd=ord
er)
    Hq_fragm, _ = pymfdfa.fitFlucVec()

# obtaining the tau indicator
    tau_wind = nq * Hq_fragm - 1

# obtaining the singularity index
    alpha_wind = np.gradient(tau_wind, nq, edge_order=2)

# obtaining a multifractal spectrum
    f_wind = nq * alpha_wind - tau_wind

# obtaining multifractal heat capacity
    C_q_wind = -np.gradient(alpha_wind, nq, edge_order=2)

# integral indicator C(q)
    C_q_area = cumulative_trapezoid(np.abs(C_q_wind), nq, initial=0)[-1]

# width of the multifractal spectrum
    delta_alpha_wind = alpha_wind.max() - alpha_wind.min()

# distance between the ends of the multifractality spectrum
    delta_phi = f_wind[-1] - f_wind[0]

# maximum alpha value
    maximal_alpha = alpha_wind.max()

# minimum value of alpha
    minimal_alpha = alpha_wind.min()

# average alpha value
    mean_alpha = np.mean(alpha_wind)

# is the value of the singularity at which the spectrum takes the maximum val
ue (α0)
    alpha_0 = alpha_wind[np.nanargmax(f_wind)]

# width of the right tail of the spectrum
    delt_alpha_right = maximal_alpha - alpha_0

# width of the left tail of the spectrum
    delt_alpha_left = alpha_0 - minimal_alpha

# difference between the width of the left and right tails
    delt_s = delt_alpha_right - delt_alpha_left

# asymmetry index
    A = (delt_alpha_left - delt_alpha_right)/(delt_alpha_left + delt_alpha_ri
```

```
ght)

# define the index at q=0
    difference_zero = np.absolute(nq-0)
    idx_zero = difference_zero.argmin()

# define the index at q=1
    difference_one = np.absolute(nq-1)
    idx_one = difference_one.argmin()

# define the index at q=2
    difference_two = np.absolute(nq-2)
    idx_two = difference_two.argmin()

# initialize arrays for the dimensions
    Dq_wind = np.zeros(len(nq))

# define generalized fractal dimensions where q!=1
    Dq_wind[nq!=nq[idx_one]] = tau_wind[nq!=nq[idx_one]] / (nq[nq!=nq[idx_one
]]-1)

# define separately the generalized fractal dimensions at q=1
    Dq_wind[nq==nq[idx_one]] = -tau_wind[nq==nq[idx_one]]

# generalized fractal dimensions obtained from the multifractal spectrum
    D_zero = f_wind[nq==nq[idx_zero]]
    D_one = f_wind[nq==nq[idx_one]]
    D_two = 2*alpha_wind[nq==nq[idx_two]] - f_wind[np.where(alpha_wind[nq==nq
[idx_two]])]

# distance from the center of the distribution of generalized dimensions to t
he left end
    delta_D_Q_left = Dq_wind[nq==q_min] - Dq_wind[nq==nq[idx_zero]]

# distance from the center of the distribution of generalized dimensions to t
he right end
    delta_D_Q_right = Dq_wind[nq==nq[idx_zero]] - Dq_wind[nq==q_max]

# h-fluctuation index (hFI)
    fluct = np.sum(np.gradient(np.gradient(Hq_fragm, nq, edge_order=2), nq, e
dge_order=2)**2)/(2*np.max(np.abs(nq))+2)

# cumulative index of increments of generalized Hurst exponents (αCF)
    incr = np.sum(np.gradient(Hq_fragm, edge_order=2)**2/ np.gradient(nq, edg
e_order=2))

    delta_alph.append(delta_alpha_wind)
    delta_spec.append(delta_phi)
    max_alph.append(maximal_alpha)
    min_alph.append(minimal_alpha)
    mean_alph.append(mean_alpha)
    alpha_zero.append(alpha_0)
    delta_alph_right.append(delt_alpha_right)
    delta_alph_left.append(delt_alpha_left)
    delta_s.append(delt_s)
```

```
    assym.append(A)
    D_0.append(D_zero)
    D_1.append(D_one)
    D_2.append(D_two)
    D_left.append(delta_D_Q_left)
    D_right.append(delta_D_Q_right)
    C_q.append(C_q_wind)
    mfSpect.append(f_wind)
    alpha.append(alpha_wind)
    hFI.append(fluct)
    alphaCF.append(incr)
    C_q_area_wind.append(C_q_area)
    h_q.append(Hq_fragm)
    tau_q.append(tau_wind)
    D_q.append(Dq_wind)
```

Save absolute values of indicators to text files.

```
# list of names of each indicator to save to txt
subtitle_of_txts = ['delta_alpha', 'delta_f', 'max_alpha', 'min_alpha', 'mean
_alpha',
'zero_alpha', 'delta_alpha_right', 'delta_alpha_left', 'assymetry',
'delta_s', 'D_0', 'D_1', 'D_2', 'hFI', 'alphaCF', 'C_q_area',
'delta_d_left', 'delta_d_right']

# list of output values of indicators for saving to txt
mfdfa_indicators = [delta_alph, delta_spec, max_alph, min_alph, mean_alph, al
pha_zero, delta_alph_right, delta_alph_left, assym, delta_s, D_0, D_1, D_2, h
FI, alphaCF, C_q_area_wind, D_left, D_right]

for i in range(len(subtitle_of_txts)):
    np.savetxt(f"mfdfa_{subtitle_of_txts[i]}_name={symbol}_ret={ret_type}_ \
               order={order}_qmin={q_min}_qmax={q_max}_qinc={q_step}_ \
              wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}.tx
t", mfdfa_indicators[i])
```

Let’s analyze the dynamics of the obtained indicators.

4.10.1 The width of the multifractal spectrum $\Delta\alpha$

The first and one of the most practical indicators of system complexity is the multifractal width, $\Delta\alpha$, which can be represented as the difference between the maximum degree of singularity and the minimum:

$$\Delta\alpha = \alpha_{max} - \alpha_{min}.$$

If we draw an analogy with thermodynamic indicators, then the width of the multifractality spectrum will be the difference between the highest and lowest values of the system’s internal energy. Let’s consider the dynamics of this indicator for the stock market indices (see Fig. 4.42):

```
measure_label = r'$\Delta\alpha$'
file_name = f"mfdfa_delta_alpha_name={symbol}_ret={ret_type}_order={order}_qm
in={q_min}_qmax={q_max}_qinc={q_step}_ \
          wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          delta_alph,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='red')
```

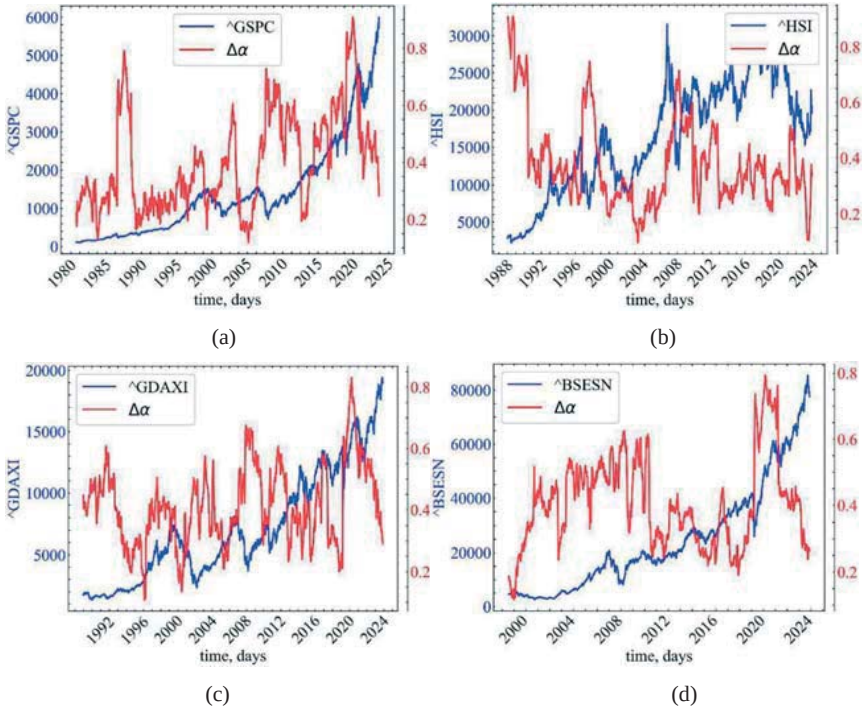


Fig. 4.42: Multifractal spectrum width indicator $\Delta\alpha$ for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

Fig. 4.42 shows that the width of the multifractality spectrum increases during crisis events, indicating an increase in the overall degree of complexity and periodization. That is, this indicator serves as another confirmation that traders in the stock market, for example, behave in a synchronized manner during a crisis.

The growth of the overall degree of multifractality is an indicator of the growth of correlations in the system, which was confirmed by the previous indicators of complexity.

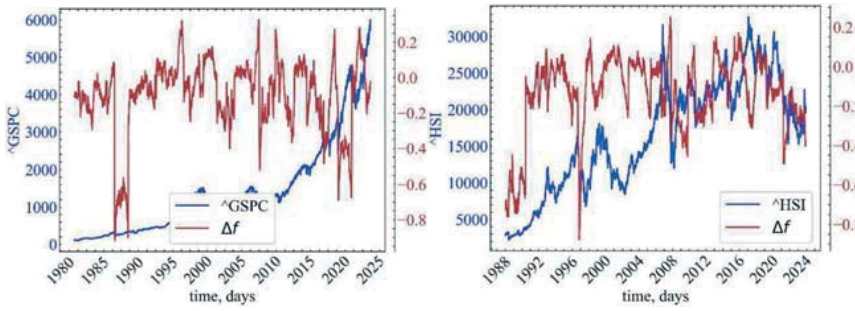
4.10.2 The difference between the ends of the multifractal spectrum Δf

However, the simple width of the multifractality spectrum does not show, for example, what type of fluctuations are most likely, what type of density elements play the greatest role in increasing or decreasing the complexity of the system. Later, we proposed such an indicator of multifractality as Δf, which can be represented as follows [105, 106, 174]:

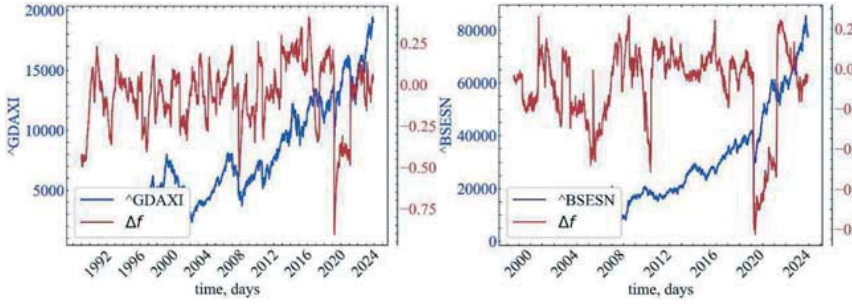
$$\Delta f = f(\alpha_{min}) - f(\alpha_{max}).$$

```
measure_label = r'$\Delta f$'
file_name = f"mfdfa_delta_f_name={symbol}_ret={ret_type}_order={order}_qmin={
q_min}_qmax={q_max}_qinc={q_step}_ \
           wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          delta_spec,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='brown')
```

Fig. 4.43 illustrates the comparative dynamics of the distance between the ends of the multifractality spectrum Δf for the time series of S&P 500, Hang Seng index, DAX, and BSE Sensex.



(a) (b)



(c) (d)

Fig. 4.43: Distance between the ends of the multifractality spectrum Δf for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

The meaning of this indicator is that it allows us to determine the degree of probability of occurrence of elements with high densities and low densities. If this indicator is less than zero, then fluctuations reflecting elements with the highest concentration (highest fractions) have the highest probability. If this indicator is higher than zero, then fluctuations reflecting low-concentration elements (small fluctuations) determine the dynamics of the system. If this indicator is zero, then both highly singular and low-singular elements contribute equally to the system dynamics.

Turning to thermodynamics, we can recall that $f(\alpha)$ is the entropy of a system. Then it becomes clear that the variability of the multifractal spectrum allows us to determine the degree of contribution of highly concentrated and low-concentrated elements to minimizing the entropy of the system. The left-handed asymmetry of the multifractal spectrum ($\Delta f > 0$) tells us that highly concentrated elements of the phase space make the greatest contribution to the minimum thermodynamic entropy. In other words, these elements are the engine of increasing orderliness of the system's dynamics. In turn, the right-hand side asymmetry of the multifractal spectrum ($\Delta f < 0$) indicates the minimization of entropy due to low-concentrated elements. The symmetry of the ends of the spectrum indicates an equal contribution of high-density and sparse regions to

entropy minimization. As already mentioned, there are cases when the multifractal spectrum practically converges to a singularity (one point). In this case, we are dealing with a simple monofractal system, which in our case was characterized by independent and normally distributed random variables. For such a spectrum, both $\Delta\alpha$ and Δf will tend to zero. For such a time series, there is no longer a set of fractal dimensions, but only one fractal index, $f[\alpha(q=0)]=1$. This is exactly the region where the system reaches its thermodynamic equilibrium – the maximum entropy. In turn, Δf can be characterized as the difference of entropies at the maximum and minimum internal energy of the system.

4.10.3 Width of the left $\Delta\alpha_L$ and right $\Delta\alpha_R$ tails of the multifractal spectrum

In addition, we can examine the degree of complexity of the dynamics of high-density regions (with large fluctuations) and low-density regions (with small fluctuations) separately. To do this, we measure the **width of the left** and **right tails** separately. The width of the left tail is defined as

$$\Delta\alpha_L = \alpha_0 - \alpha_{min},$$

and the width of the right tail as

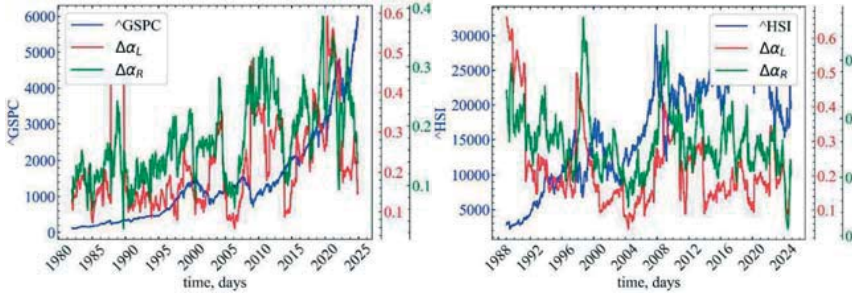
$$\Delta\alpha_R = \alpha_{max} - \alpha_0.$$

In turn, the width of the left tail measures the degree of complexity of fluctuations with a large amplitude, and the width of the right tail measures the degree of complexity of small fluctuations. An increase in the width of each of the tails will reflect an increase in the degree of correlation between the elements.

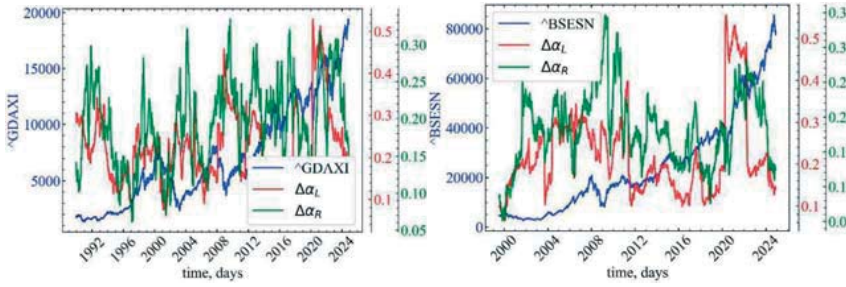
```
fig, ax = plt.subplots(1, 1)

ax2 = ax.twinx()
ax3 = ax.twinx()

ax2.spines.right.set_position(("axes", 1.03))
ax3.spines.right.set_position(("axes", 1.12))

p1, = ax.plot(time_ser.index[window:length:tstep], time_ser.values[window:len
gth:tstep],
"b-", label=fr"{ylabel}")
p2, = ax2.plot(time_ser.index[window:length:tstep], delta_alph_left, color="r
```

```
", label=r"$\Delta\alpha_{L}$")
p3, = ax3.plot(time_ser.index[window:length:tstep], delta_alph_right, color="
g", label=r"$\Delta\alpha_{R}$")

ax.set_xlabel(xlabel)
ax.set_ylabel(f"{ylabel}")

ax.yaxis.label.set_color(p1.get_color())
ax2.yaxis.label.set_color(p2.get_color())
ax3.yaxis.label.set_color(p3.get_color())

tkw = dict(size=4, width=1.5)
ax.tick_params(axis='y', colors=p1.get_color(), **tkw)
ax2.tick_params(axis='y', colors=p2.get_color(), **tkw)
ax3.tick_params(axis='y', colors=p3.get_color(), **tkw)
ax.tick_params(axis='x', rotation=45, **tkw)

ax3.legend(handles=[p1, p2, p3])

plt.savefig(f"mfdfa_delta_alpha_left_right_name={symbol}_ret={ret_type}_order
={order}_qmin={q_min}_qmax={q_max}_qinc={q_step}_wind={window}_step={tstep}_w
indbeg={win_beg}_winden={win_end}.jpg")
plt.show();
```

Fig. 4.44 demonstrates the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their width of the left and right tails of the multifractal spectrum.



(a) (b)



(c) (d)

Fig. 4.44: Width of the left and right tails of the multifractal spectrum for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

Fig. 4.44 shows that the studied indicators react in a characteristic way to crisis events. The width of the left side of the multifractality spectrum increases during 1992, 1996-2000, 2008, 2016, and the coronavirus pandemic. This indicates the dominance of highly concentrated fluctuations (with a large amplitude of fluctuations). In addition, the increase in the width of the left tail indicates that fluctuations with a large amplitude of fluctuations are characterized by an increase in the degree of correlation during crisis events, which in turn can serve as an indicator of the growth of self-organization processes.

Although smaller, the dynamics of the width of the right tail of the multifractal spectrum is no less remarkable. This indicator works almost similarly to the width of the left tail, but characterizes the dynamics of low-concentrated values – fluctuations with a small amplitude of oscillations. Almost synchronous dynamics of both indicators indicates an increase in the influence of fluctuations of both large-amplitude fluctuations and small-amplitude fluctuations. In other words, these two types of fluctuations are the source of the growth of nonlinear correlations during crisis events.

4.10.4 Singularity exponent α and its variants

Possible indicators of system complexity include α_{min}, α_{max}, α_{mean}, and α_0, which respectively characterize the **minimum singularity strength**,

maximum, **average**, and **singularity under the condition of equilibrium consideration** of both large fluctuations and small ones.

```
fig, ax = plt.subplots(1, 1)

ax2 = ax.twinx()
ax3 = ax.twinx()
ax4 = ax.twinx()
ax5 = ax.twinx()

ax3.spines.right.set_position(("axes", 1.08))
ax4.spines.right.set_position(("axes", 1.18))
ax5.spines.right.set_position(("axes", 1.27))

p1, = ax.plot(time_ser.index[window:length:tstep], time_ser[window:length:tst
ep], "b-", label=fr"{ylabel}")
p2, = ax2.plot(time_ser.index[window:length:tstep], max_alph, "r-", label=r"$
\alpha_{max}$")
p3, = ax3.plot(time_ser.index[window:length:tstep], min_alph, "g-", label=r"$
\alpha_{min}$")
p4, = ax4.plot(time_ser.index[window:length:tstep], mean_alph, "c-", label=r"
$\alpha_{mean}$")
p5, = ax5.plot(time_ser.index[window:length:tstep], alpha_zero, "m-", label=r
"$\alpha_{0}$")

ax.set_xlabel(xlabel)
ax.set_ylabel(fr"{ylabel}")

ax.yaxis.label.set_color(p1.get_color())
ax2.yaxis.label.set_color(p2.get_color())
ax3.yaxis.label.set_color(p3.get_color())
ax4.yaxis.label.set_color(p4.get_color())
ax5.yaxis.label.set_color(p5.get_color())

tkw = dict(size=4, width=1.5)
ax.tick_params(axis='y', colors=p1.get_color(), **tkw)
ax2.tick_params(axis='y', colors=p2.get_color(), **tkw)
ax3.tick_params(axis='y', colors=p3.get_color(), **tkw)
ax4.tick_params(axis='y', colors=p4.get_color(), **tkw)
ax5.tick_params(axis='y', colors=p5.get_color(), **tkw)
ax.tick_params(axis='x', **tkw, pad=10, rotation=45)

ax5.legend(handles=[p1, p2, p3, p4, p5])

plt.savefig(f"mfdfa_alpha_min_max_mean_zero_name={symbol}_ret={ret_type}_orde
r={order}_qmin={q_min}_qmax={q_max}_qinc={q_step}_ \
            wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}.jpg
", bbox_inches="tight")
```

Fig. 4.45 demonstrates the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their singularity indicators.

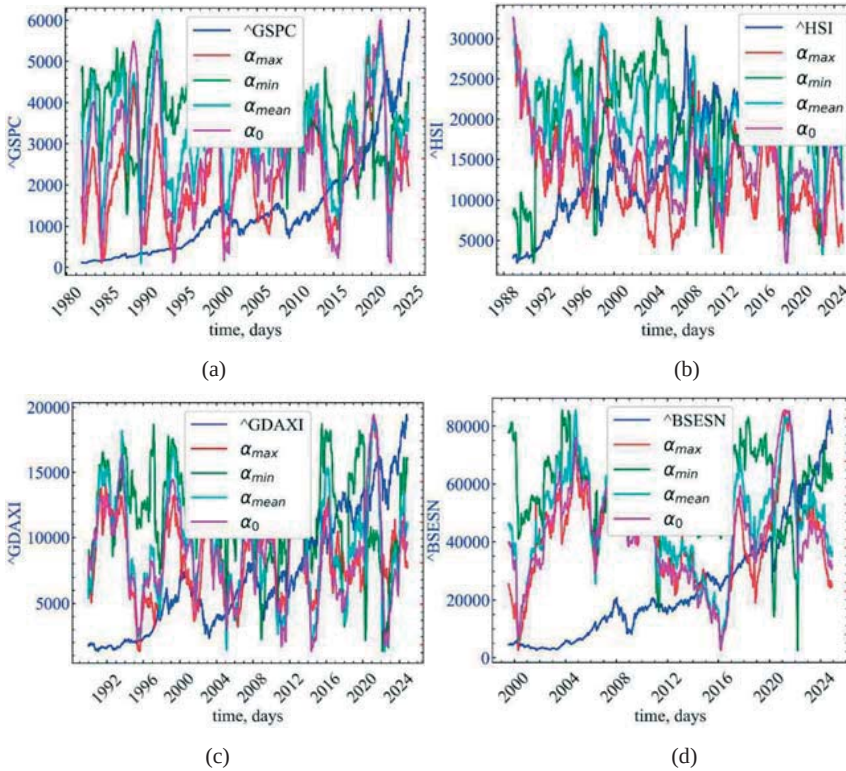


Fig. 4.45: Singularity indicators for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

As can be seen from Fig. 4.45, all singularity indicators increase in the financial phase transition from a state of stability to a state of crisis. This indicates an increase in the complexity of the system: a sharp increase in the number of agents involved in the self-organized development of the system under study. From the point of view of thermodynamics, it could be said that the internal energy of the system increases during financial collapse events.

4.10.5 Type of the long tail of the multifractal spectrum ΔS

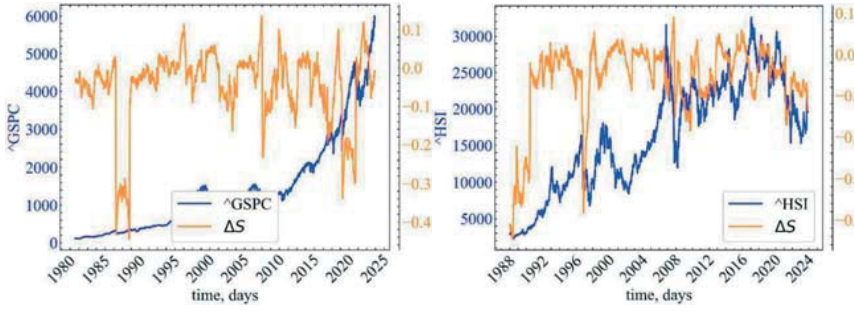
In addition to such a measure as Δf, other measures of multifractal spectrum asymmetry can be presented. For example, we can determine the **type of long tail of the multifractal spectrum** using the ΔS measure [56]:

$$\Delta S = \Delta\alpha_R - \Delta\alpha_L.$$

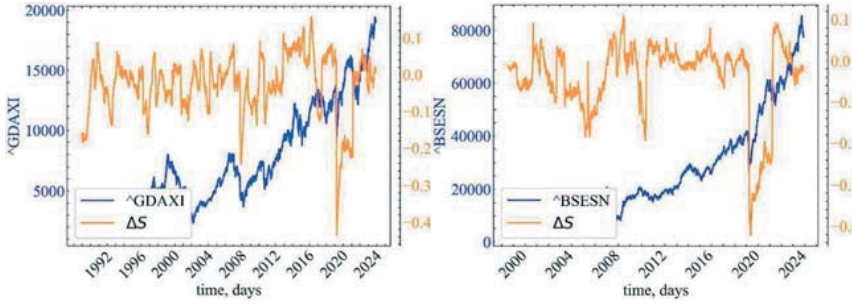
If $\Delta S < 0$, the multifractal spectrum has a long left tail, which indicates the sensitivity of the time series to local fluctuations with a large amplitude. If $\Delta S > 0$, the multifractal spectrum has a long right tail, which indicates the sensitivity of the signal structure to local fluctuations with a small amplitude. In cases where the high- and low-frequency components of the signal are comparable, the singularity spectrum will be approximately symmetrical and $\Delta\alpha_R = \Delta\alpha_L$.

```
measure_label = r'$\Delta S$'
file_name = f"mfdfa_delta_s_name={symbol}_ret={ret_type}_order={order}_qmin={
q_min}_qmax={q_max}_qinc={q_step}_ \
           wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          delta_s,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='darkorange')
```

Fig. 4.46 demonstrates the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their ΔS indicator.



(a) (b)



(c) (d)

Fig. 4.46: Multifractal spectrum tail type indicator ΔS for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

Fig. 4.46 shows that $\Delta S < 0$, which indicates that the most crashing parts of the stock indices are caused by fluctuations with a large amplitude of fluctuations.

4.10.6 Asymmetry index A

Next, we can define the following asymmetry index [128, 144, 145]:

$$A = (\Delta\alpha_L - \Delta\alpha_R)/(\Delta\alpha_R + \Delta\alpha_L) = -\Delta S/\Delta\alpha.$$

The asymmetry parameter is associated with the predominant type of oscillation in the system under study. If $A = 0$ ($\Delta\alpha_L = \Delta\alpha_R$), the system dynamics is represented by a symmetrical spectrum. If $A < 0$ ($\Delta\alpha_L < \Delta\alpha_R$), the multifractal spectrum has a right-handed asymmetry, which emphasizes the stronger influence of small fluctuations on multifractality. Conversely, when $A > 0$ ($\Delta\alpha_L > \Delta\alpha_R$), then we are dealing with a left-handed spectrum, which denotes greater heterogeneity for large fluctuations and indicates that the time series is dominated by the multifractal nature of high-density heterogeneities. Since the asymmetry is detected by the sign of A, which is equivalent to the sign of ΔS, then, based on the sign of ΔS, we can draw conclusions about both the type of long tail and the sign of the multifractal spectrum indicator A, i.e., the insensitivity and type of dominant fluctuations of the multifractal nature of the time series.

```
measure_label = r'$A$'
file_name = f"mfdfa_A_name={symbol}_ret={ret_type}_order={order}_qmin={q_min}
```

```
_qmax={q_max}_qinc={q_step}_ \
          wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          assym,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='darkviolet')
```

Fig. 4.47 illustrates the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their asymmetry index.

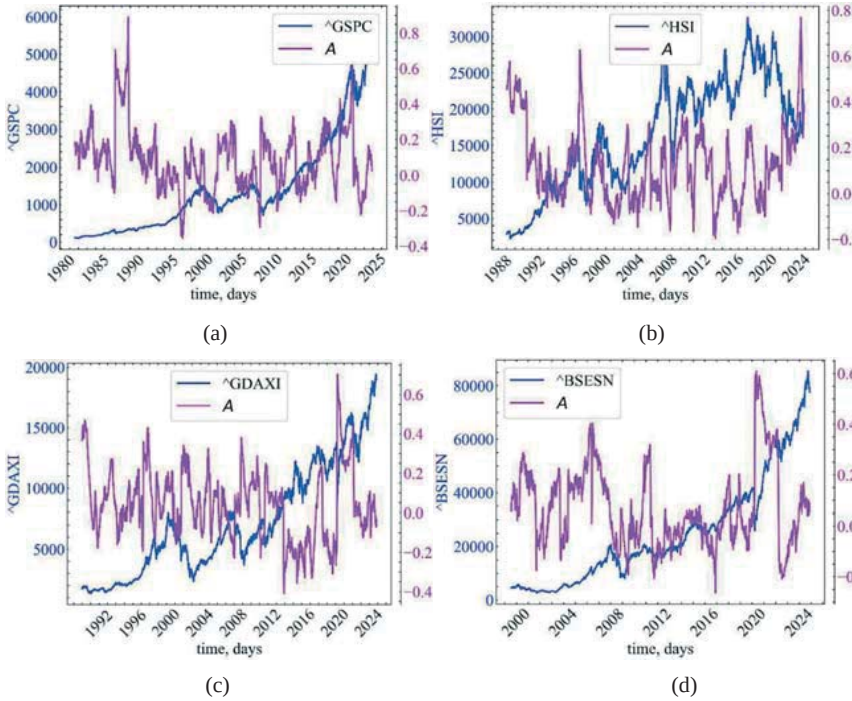


Fig. 4.47: Multifractal spectrum asymmetry index *A* for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

Fig. 4.47 shows that, as a rule, the asymmetry index increases during crashes and indicates the dominance of the left-hand side spectrum (highly concentrated fluctuations with large amplitudes). It is difficult to associate small and large

fluctuations with specific market sentiment or behavioral patterns. At this point, we can only note that these events represented the richest variation in both short-term and long-term correlations.

4.10.7 *h*-fluctuation index *hFI*

The fluctuation can be analyzed using the second derivative of the generalized Hurst exponent. Note that the amplitude of the second derivative in the case of multifractal signals is greater than for monofractal signals. To obtain the necessary information from $h(q)$, the ***h*-fluctuation index** (hFI) was proposed [19], which is defined as the power of the second derivative of $h(q)$:

$$hFI = \frac{1}{2|q_{max}| + 2} \sum_{q=q_{min}-2}^{q_{max}} [h(q) - 2h(q-1) + h(q-2)]^2.$$

The higher the value of this indicator, the higher the self-organization of the system.

```
measure_label = r'$hFI$'
file_name = f"mfdfa_hFI_name={symbol}_ret={ret_type}_order={order}_qmin={q_mi
n}_qmax={q_max}_qinc={q_step}_ \
           wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          hFI,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='green')
```

Fig. 4.48 illustrates the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their h-fluctuation index.

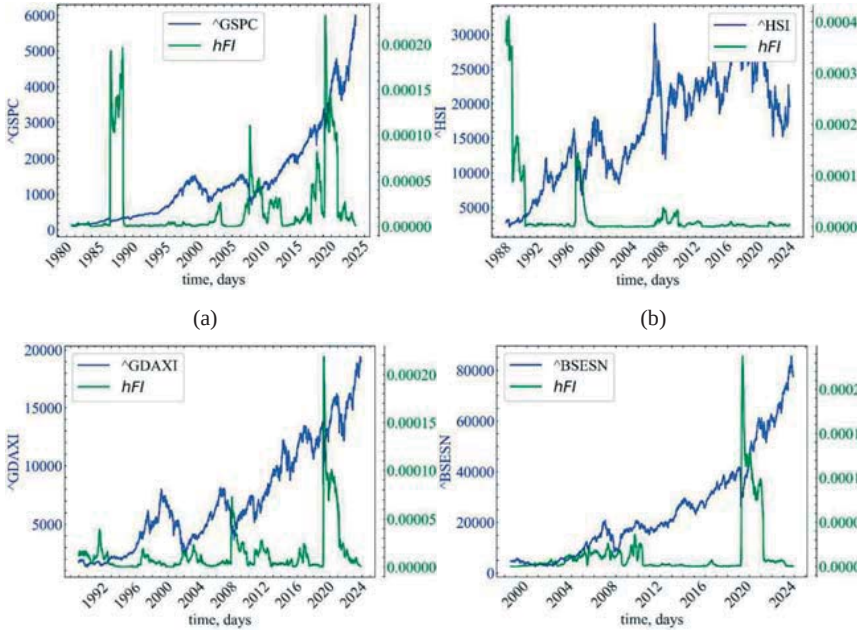


Fig. 4.48: The h-fluctuation index for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

It can be seen that according to hFI, the highest degree of multifractality is manifested precisely for the crises of 1987, 1997, 2008, and 2020. These collapse events include the largest number of different factors that influenced the dynamics of the system under study. This is especially noticeable for the coronavirus pandemic.

4.10.8 Cumulative index of increments of generalized Hurst exponents αCF

The cumulative square function of increments (αCF) [6] of generalized Hurst exponents between successive moment orders is a more reliable measure of the distribution of generalized Hurst exponents.

```
measure_label = r'$\alpha CF$'
file_name = f"mfdfa_alphaCF_name={symbol}_ret={ret_type}_order={order}_qmin={
```

```
q_min}_qmax={q_max}_qinc={q_step}_ \
          wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          alphaCF,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='crimson')
```

In Fig. 4.49 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their αCF index.

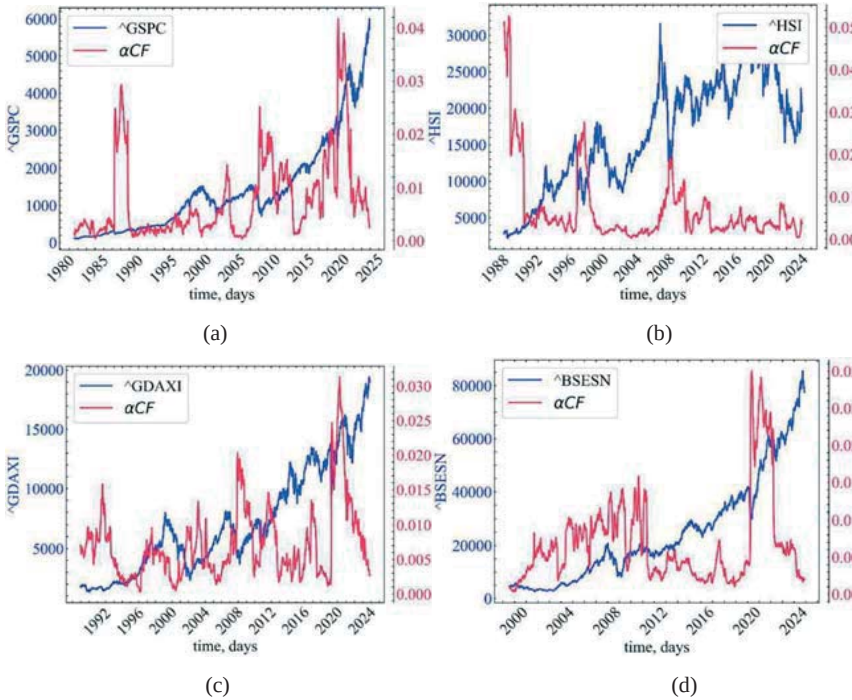


Fig. 4.49: Cumulative index of increments of generalized Hurst exponents αCF for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

The cumulative index presented here is slightly different from the hFI, but logically it is approximately the same: events with the highest degree of multifractality are characterized by a higher amplitude of αCF. The presented

index highlights the same crises as the previous one, but the dynamics of this index is more pronounced, which makes it more reliable for identifying periods of system self-organization.

4.10.9 Integral multifractal heat capacity $C(q)$

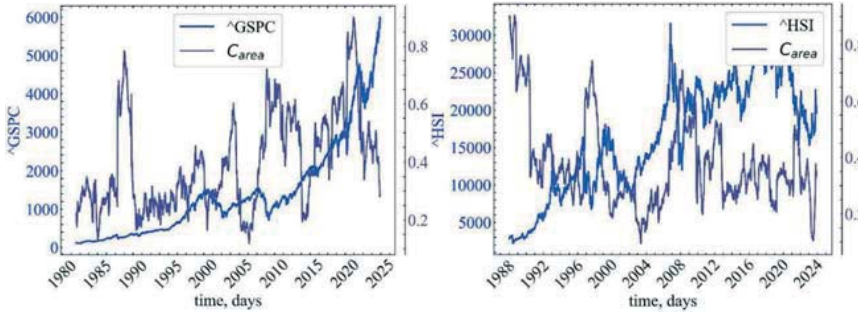
The total degree of multifractality, the **integral multifractal specific heat capacity** (C_{area}), can be expressed in the following form:

$$C_{area} = \int C(q)dq.$$

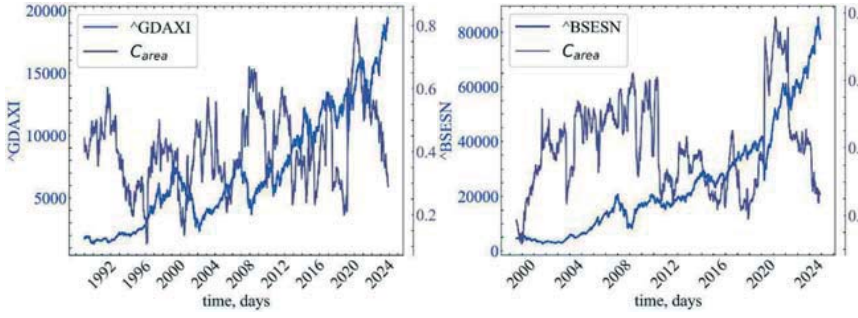
Let's calculate and analyze the dynamics of this indicator.

```
measure_label = r'$C_{area}$'
file_name = f"mfdfa_C_q_area_name={symbol}_ret={ret_type}_order={order}_qmin=
{q_min}_qmax={q_max}_qinc={q_step}_ \
          wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          C_q_area_wind,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='darkslateblue')
```

In Fig. 4.50 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their integral multifractal heat capacity C_{area} index.



(a) (b)



(c) (d)

Fig. 4.50: Integral multifractal heat capacity C_{area} for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

The figure shows that the dynamics of the integral heat capacity is very similar to the width of the multifractality spectrum. In other words, C_{area} is a complexity indicator that indicates the degree of self-organization of the financial phase transition. It can be seen that financial crashes represent a fairly trend-stable dynamic, which is the result of purposeful and collective actions of traders in the market.

4.10.10 Hausdorff dimension D_0

As already noted, D_0 represents the upper limit of the dimensional changes of the fractal subsets of the attractor system. It does not contain information about the statistical properties of the system and is not of particular value.

```
measure_label = r'$D_{0}$'
file_name = f"mfdfa_D_0_area_name={symbol}_ret={ret_type}_order={order}_qmin=
{q_min}_qmax={q_max}_qinc={q_step}_ \
          wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          D_0,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='darkred')
```

Fig. 4.51 shows the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their Hausdorff dimension.

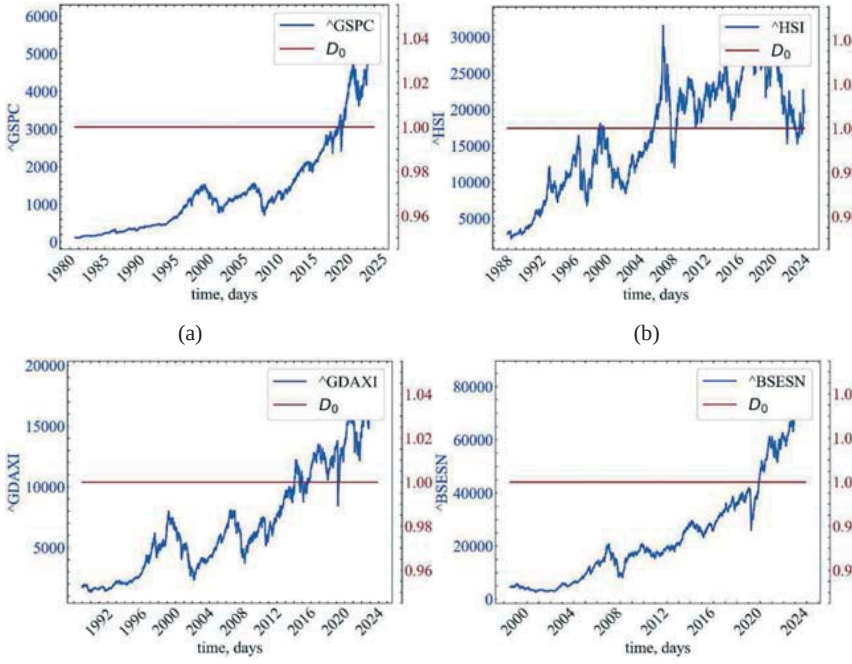


Fig. 4.51: Hausdorff dimension for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

4.10.11 Information dimension D_1

The **information dimension** is closely related to the Shannon information entropy. The higher the value of D_1, the faster the entropy increases, which is an indicator of how little we know about the current state of the system. As D_1 decreases, the entropy decreases, which in turn indicates an increase in asymmetry in space, a decrease in complexity, and an increase in our understanding of the current state of the system. It also means that we need less information to describe possible system configurations.

```
measure_label = r'$D_{1}$'
file_name = f"mfdfa_D_1_area_name={symbol}_ret={ret_type}_order={order}_qmin=
```

```
{q_min}_qmax={q_max}_qinc={q_step}_ \
          wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          D_1,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='darkred')
```

Fig. 4.52 shows the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their information dimension.

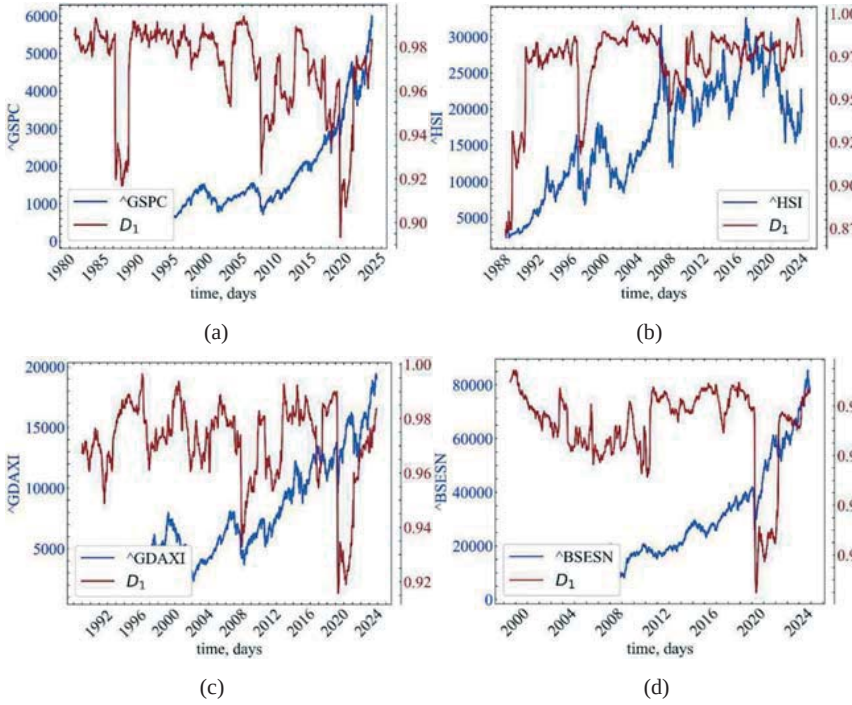


Fig. 4.52: Information dimension for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

Fig. 4.52 shows that the information dimension is characterized by a decline during crash events. This indicates an increase in the degree of orderliness of the

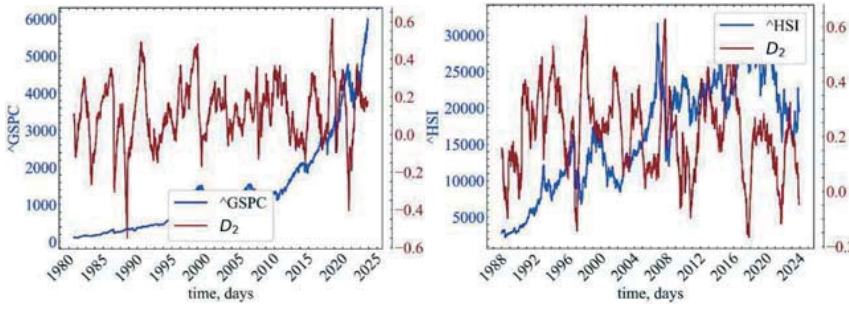
system and the collective attraction of market agents to a specific area of the phase space of the system under study.

4.10.12 Correlation dimension D_2

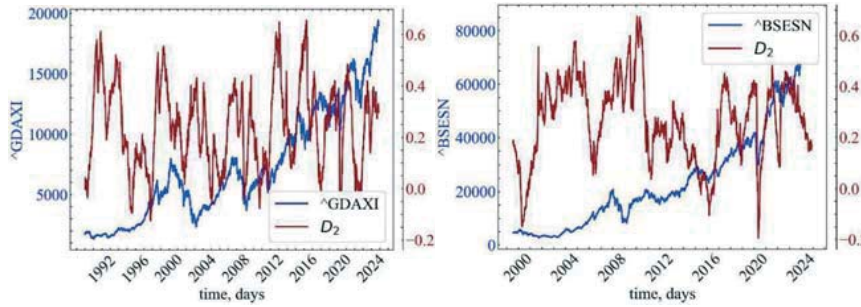
The **correlation dimension**, analogous to the information dimension, can be represented as the tangent of the slope angle of the regression line plotted on a logarithmic scale with respect to the dependence of the correlation integral $C(\varepsilon)$ on ε. Similar to D_1, the correlation dimension also determines how quickly the value of the correlation integral changes.

```
measure_label = r'$D_{2}$'
file_name = f"mfdfa_D_2_area_name={symbol}_ret={ret_type}_order={order}_qmin=
{q_min}_qmax={q_max}_qinc={q_step}_\
          wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          D_2,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='darkred')
```

Fig. 4.53 shows the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their correlation dimension.



(a) (b)



(c) (d)

Fig. 4.53: Correlation dimension for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

The correlation dimension in Fig. 4.53 is characterized by an increase in the pre-crisis period and a decrease during the crisis. This suggests that most market agents are beginning to focus on one particular vector of system development.

4.10.13 Curvature of the left ΔD_L and right ΔD_R tails of the distribution of generalized fractal dimensions

The degree of this complexity can be characterized by the **curvature of the right and left tails** of the generalized fractal dimensions. The right side (ΔD_R) can be defined as

$$\Delta D_R = D_0 - D_{q_{max}}.$$

And the higher the value of this measure, the stronger will be the degree of influence of the elements with the highest concentration (density, amplitude of fluctuations) on the overall complexity of the system.

Curvature of the left tail of the curve of generalized fractal dimensions (ΔD_L):

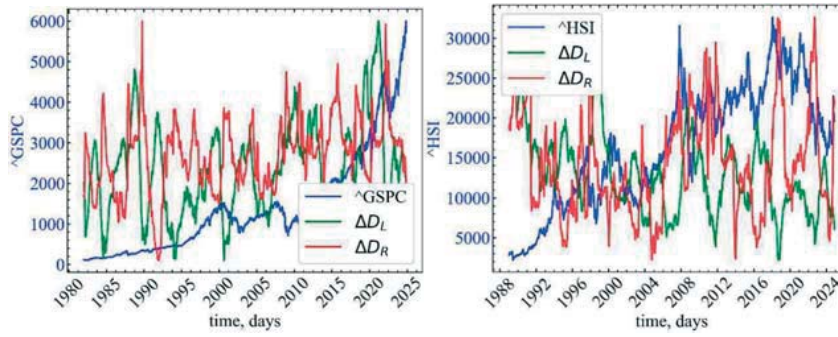
$$\Delta D_L = D_{q_{min}} - D_0.$$

This indicator will tell us how strong the influence of the least concentrated elements is on the complexity of the system.

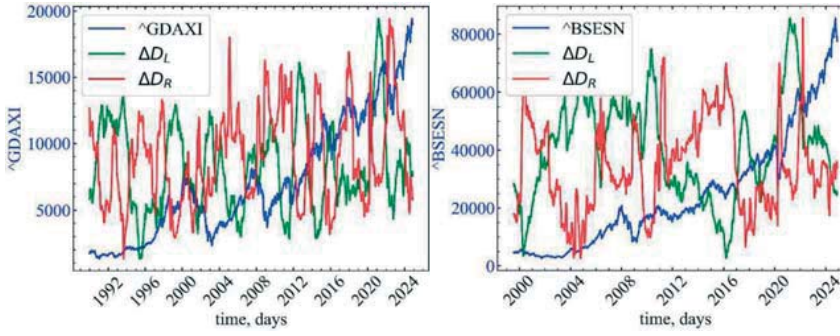
```
fig, ax = plt.subplots(1, 1)
```

```
ax2 = ax.twinx()
ax3 = ax.twinx()

ax2.spines.right.set_position(("axes", 1.03))
ax3.spines.right.set_position(("axes", 1.12))

p1, = ax.plot(time_ser.index[window:length:tstep], time_ser.values[window:len
gth:tstep], "b-", label=fr"{ylabel}")
p2, = ax2.plot(time_ser.index[window:length:tstep], D_left, color="g", label=
r"$\Delta D_{L}$")
p3, = ax3.plot(time_ser.index[window:length:tstep], D_right, color="r", label
=r"$\Delta D_{R}$")

ax.set_xlabel(xlabel)
ax.set_ylabel(f"{ylabel}")

ax.yaxis.label.set_color(p1.get_color())
ax2.yaxis.label.set_color(p2.get_color())
ax3.yaxis.label.set_color(p3.get_color())

tkw = dict(size=4, width=1.5)
ax.tick_params(axis='y', colors=p1.get_color(), **tkw)
ax2.tick_params(axis='y', colors=p2.get_color(), **tkw)
ax3.tick_params(axis='y', colors=p3.get_color(), **tkw)
ax.tick_params(axis='x', rotation=45, **tkw)

ax3.legend(handles=[p1, p2, p3])

plt.savefig(f"mfdfa_delta_D_left_right_name={symbol}_ret={ret_type}_order={or
der}_qmin={q_min}_qmax={q_max}_qinc={q_step}_wind={window}_step={tstep}_windb
eg={win_beg}_winden={win_end}.jpg")
plt.show();
```

Fig. 4.54 demonstrates calculated curvature of the left and right tails of the generalized fractal dimensions spectrum for the time series of S&P 500, Hang Seng index, DAX, and BSE Sensex.



(a) (b)



(c) (d)

Fig. 4.54: Curvature of the left and right tails of the generalized fractal dimensions spectrum for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

4.10.14 Two- and three-dimensional visualization of multifractality indicators

Previously, we analyzed the dependencies $h(q)$, $\tau(q)$, $D(q)$, $C(q)$, and $f(\alpha)$ for the entire time series. Now, using the sliding window procedure, we can look at how they change over time. Let's use the time series of the studied stock indices.

First of all, let's define the functions for constructing two-dimensional graphs:

```
def plot_2d(X, Y, Z, subtitle_jpg, subtitle_fig, ylabel, barlabel, cmap, lims
):

    fig, ax = plt.subplots(1, 1, figsize=(10, 5))

    cp = ax.contourf(X, Y, Z, alpha=0.8, cmap=cmap)
    plt.colorbar(cp, ax=ax, extend='both', label=barlabel)

    ax.set_xlim((time_ser.index[window:length:tstep][0],
                 time_ser.index[window:length:tstep][-1]))
    ax.set_ylim((np.min(lims), np.max(lims)))

    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)

    ax.set_title(subtitle_fig, pad=10)

    ax.tick_params(axis='both', which='major', pad=10)

    fig.tight_layout()

    plt.savefig(f"mfdfa_{subtitle_jpg}_name={symbol}_ret={ret_type}_order={or
```

```
der}_ \
                qmin={q_min}_qmax={q_max}_qinc={q_step}_windbeg={win_beg}_win
den={win_end}.jpg",
                bbox_inches="tight")
    plt.show();
```

and three-dimensional:

```
def plot_3d(X, Y, Z, subtitle_jpg, ylabel, zlabel, cmap):

    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})

    surf = ax.plot_surface(X, Y, Z, cmap=cmap, rstride=2, cstride=2, linewidt
h=0)

    ax.set_xlabel(xlabel, labelpad=15)
    ax.set_ylabel(ylabel, labelpad=15)
    ax.set_zlabel(zlabel, labelpad=15)
    ax.tick_params(axis='both', which='major', pad=5)

    fig.colorbar(surf, shrink=0.5, aspect=10, location='right', pad=0.1)

    fig.tight_layout()

    plt.savefig(f"mfdfa_{subtitle_jpg}_name={symbol}_ret={ret_type}_order={or
der}_ \
                qmin={q_min}_qmax={q_max}_qinc={q_step}_windbeg={win_beg}_ \
                winden={win_end}.jpg", bbox_inches="tight")

    plt.show();
```

After declaring the required functions, you can start visualizing.

```
X, Y = np.meshgrid(time_ser.index[window:length:tstep], nq)
Z = np.array(h_q).T

plot_2d(X, Y, Z,
        subtitle_jpg='contour_h(q)',
        subtitle_fig=fr"Heat chart $h(q)$",
        ylabel=r"$q$",
        barlabel=r"$h(q)$",
        cmap='jet',
        lims=nq)
```

Figs. 4.55 and 4.56 will show the dynamics of the generalized Hurst exponent $h(q)$ changing over time for the time series of S&P 500, Hang Seng index, DAX, and BSE Sensex within two- and three-dimensional representations.

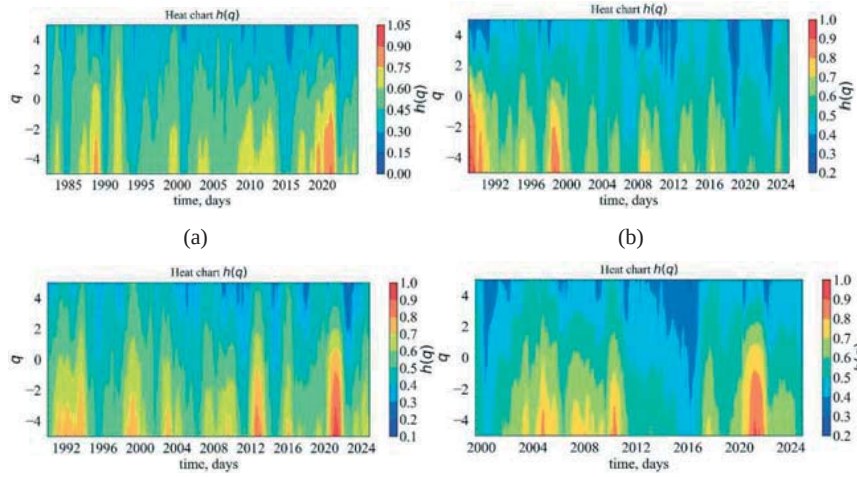
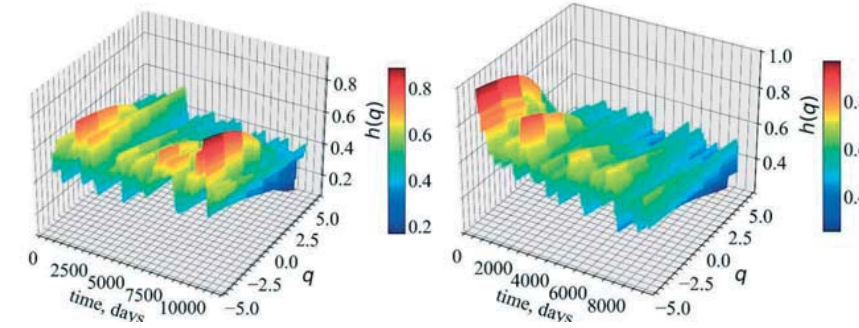
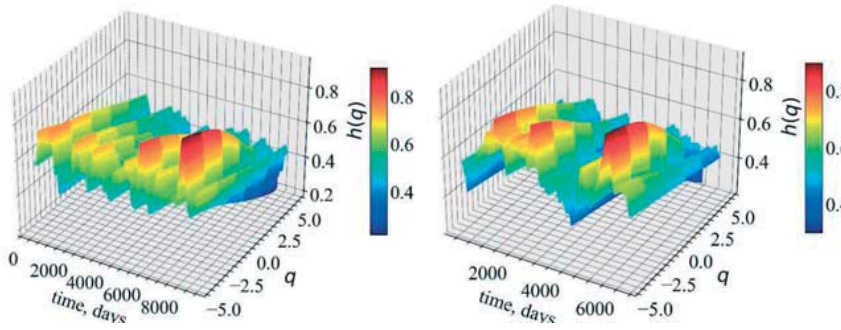


Fig. 4.55: Two-dimensional contour diagram of the dynamics of the generalized Hurst exponent $h(q)$ changing over time for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

```
X, Y = np.meshgrid(np.arange(window, length, tstep), nq)
Z = np.array(h_q).T

plot_3d(X, Y, Z,
        subtitle_jpg='3d_h(q)',
        ylabel=r"$q$",
        zlabel=r"$h(q)$",
        cmap='jet')
```





(c) (d)

Fig. 4.56: Three-dimensional diagram of the dynamics of the generalized Hurst exponent $h(q)$ changing over time for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

Fig. 4.55 and Fig. 4.56 show that the generalized Hurst exponent is characterized by a significant increase during crises. The $h(q)$ is especially high for $q < 0$, which indicates a significant persistence of small fluctuations during periods of turbulence. In this case, the highest degree of nonlinearity is represented by the crises of 1987, 1997, 2008, and 2020-2021, which is confirmed by the previous indicators.

4.10.15 Dynamics of $\tau(q)$ over time in two- and three-dimensional spaces

```
X, Y = np.meshgrid(time_ser.index[window:length:tstep], nq)
Z = np.array(tau_q).T

plot_2d(X, Y, Z,
        subtitle_jpg='contour_tau(q)',
        subtitle_fig=fr"Теплова діаграма $\tau(q)$",
        ylabel=r"$q$",
        barlabel=r"$\tau(q)$",
        cmap='viridis',
        lims=nq)
```

Figs. 4.57 and 4.58 will demonstrate the dynamics of the indicator $\tau(q)$ changing over time for the time series of S&P 500, Hang Seng index, DAX, and BSE Sensex within two- and three-dimensional representations.

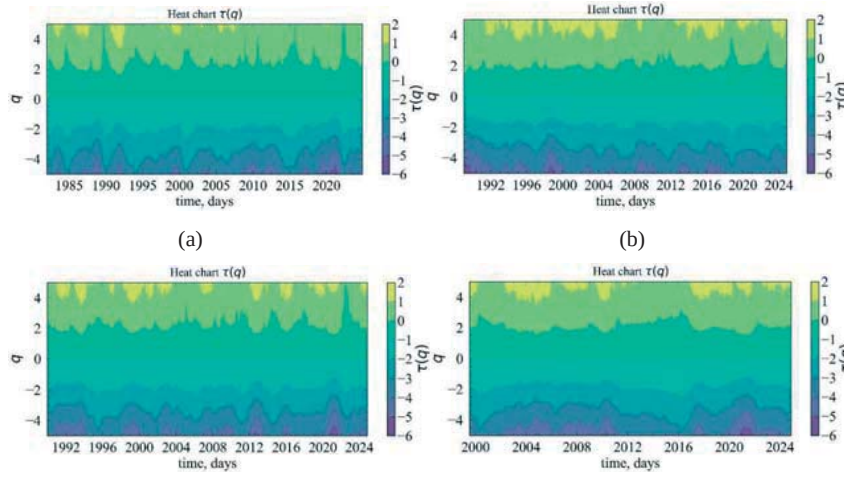
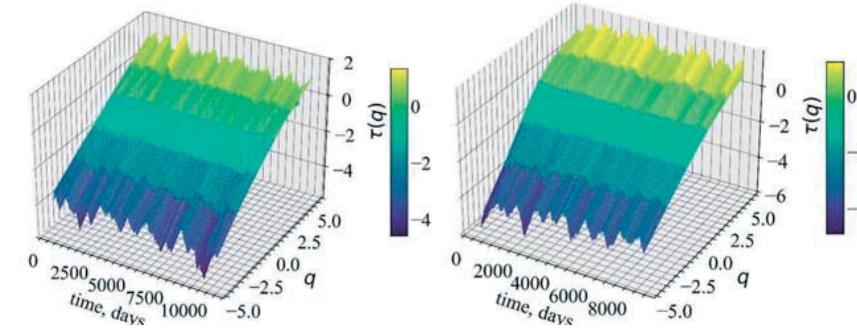
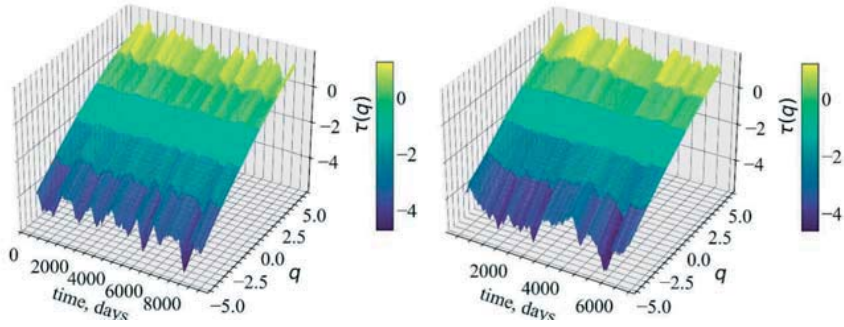


Fig. 4.57: Two-dimensional contour diagram of the dynamics of the indicator $\tau(q)$ changing over time for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

```
X, Y = np.meshgrid(np.arange(window, length, tstep), nq)
Z = np.array(tau_q).T

plot_3d(X, Y, Z,
        subtitle_jpg='3d_tau(q)',
        ylabel=r"$q$",
        zlabel=r"$\tau(q)$",
        cmap='viridis')
```





(c) (d)

Fig. 4.58: Three-dimensional diagram of the dynamics of the indicator $\tau(q)$ changing over time for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

As can be seen from the figures (Fig. 4.57 and Fig. 4.58), $\tau(q)$ becomes more nonlinear for all values of q. Significant troughs can be seen at the ends of the tails of this indicator, which can serve as indicators of crash events, but compared to the same Hurst exponent, this indicator is less expressive.

4.10.16 Dynamics of $D(q)$ over time in two- and three-dimensional spaces

```
X, Y = np.meshgrid(time_ser.index[window:length:tstep], nq)
Z = np.array(D_q).T

plot_2d(X, Y, Z,
        subtitle_jpg='contour_D(q)',
        subtitle_fig=fr"Heat chart $D(q)$",
        ylabel=r"$q$",
        barlabel=r"$D(q)$",
        cmap='magma',
        lims=nq)
```

Figs. 4.59 and 4.60 will illustrate the dynamics of the generalized fractal dimension $D(q)$ changing over time for the time series of S&P 500, Hang Seng index, DAX, and BSE Sensex within two- and three-dimensional representations.

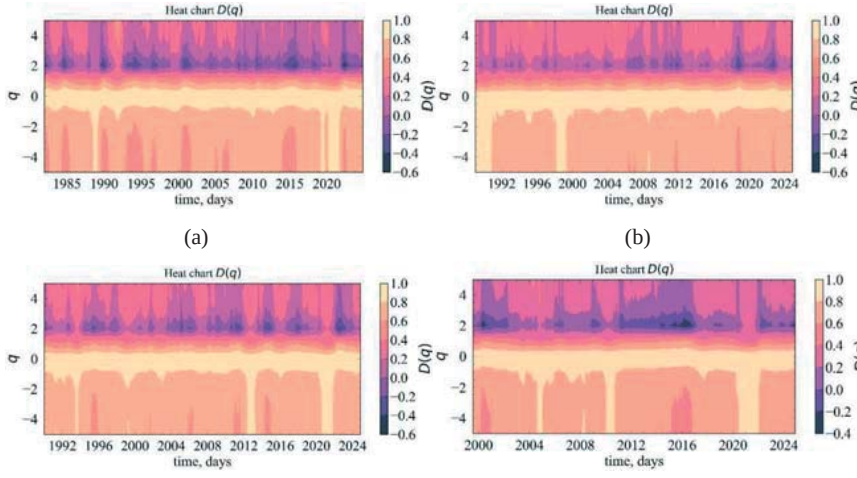
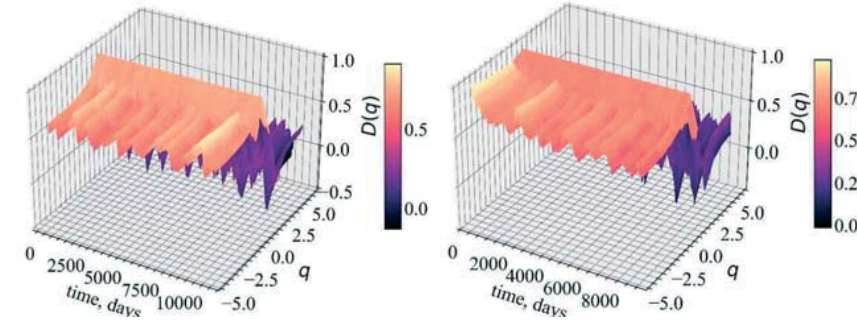
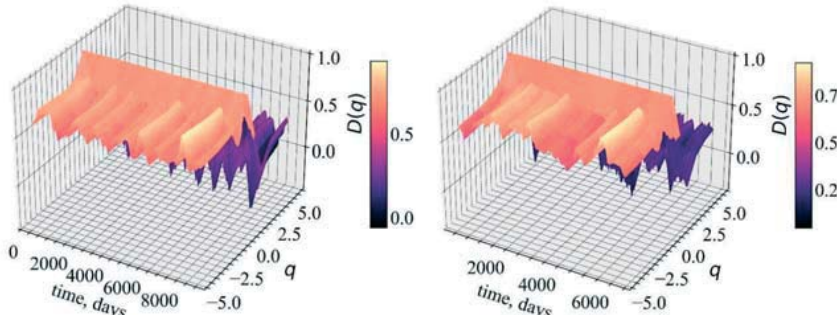


Fig. 4.59: Two-dimensional contour diagram of the dynamics of the generalized fractal dimension $D(q)$ changing over time for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

```
X, Y = np.meshgrid(np.arange(window, length, tstep), nq)
Z = np.array(D_q).T

plot_3d(X, Y, Z,
        subtitle_jpg='3d_D(q)',
        ylabel=r"$q$",
        zlabel=r"$D(q)$",
        cmap='magma')
```





(c) (d)

Fig. 4.60: Three-dimensional diagram of the dynamics of the generalized fractal dimension $D(q)$ changing over time for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

The two- and three-dimensional representations of the generalized fractal dimension show that $D(q)$ increases during crisis events. The generalized fractal dimension also presents the most indicative dynamics for negative values of q, although there are also slight fluctuations for positive q.

4.10.17 Dynamics of $C(q)$ in two- and three-dimensional spaces

```
X, Y = np.meshgrid(time_ser.index[window:length:tstep], nq)
Z = np.array(C_q).T

plot_2d(X, Y, Z,
        subtitle_jpg='contour_C(q)',
        subtitle_fig=fr"Heat chart $C(q)$",
        ylabel=r"$q$",
        barlabel=r"$C(q)$",
        cmap='hot',
        lims=nq)
```

Figs. 4.61 and 4.62 will demonstrate the dynamics of the multifractal heat capacity $C(q)$ changing over time for the time series of S&P 500, Hang Seng index, DAX, and BSE Sensex within two- and three-dimensional representations.

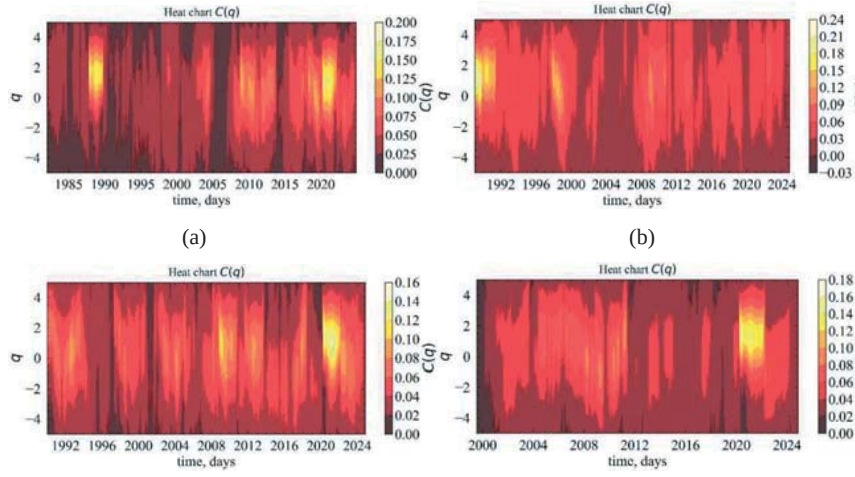
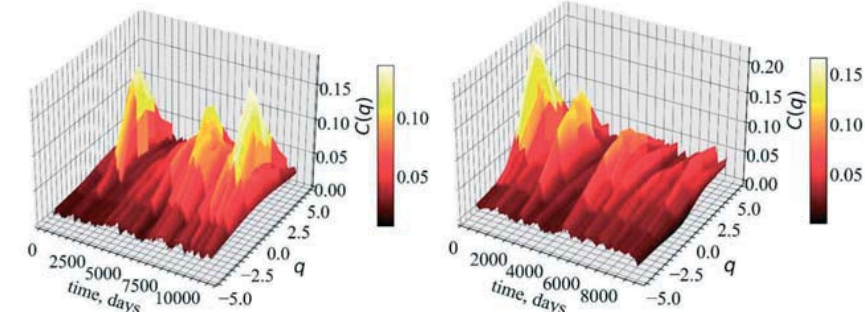
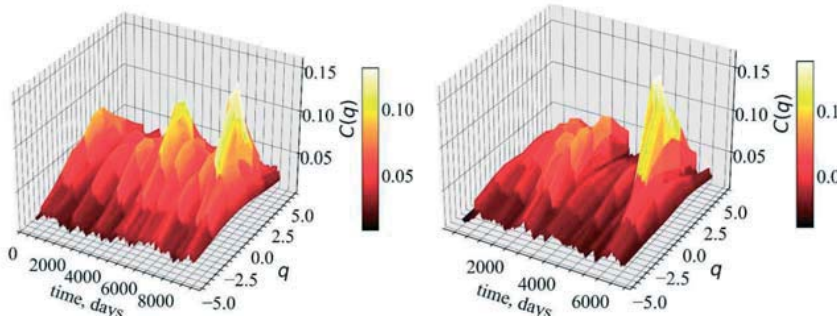


Fig. 4.61: Two-dimensional contour diagram of the dynamics of the multifractal heat capacity $C(q)$ changing with time for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

```
X, Y = np.meshgrid(np.arange(window, length, tstep), nq)
Z = np.array(C_q).T

plot_3d(X, Y, Z,
        subtitle_jpg='3d_C(q)',
        ylabel=r"$q$",
        zlabel=r"$C(q)$",
        cmap='hot')
```





(c) (d)

Fig. 4.62: Three-dimensional contour diagram of the dynamics of the multifractal heat capacity $C(q)$ changing with time for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

In these figures (Fig. 4.61 and Fig. 4.62), jumps in multifractal heat capacity are observed during crisis events, which indicates the analogy between physical phase transitions and crisis events. It can be seen that under different market regimes, $C(q)$ can be symmetrical, demonstrating an equal impact on market dynamics of both highly concentrated and low-concentrated elements. Also, $C(q)$ can shift to the left as well as to the right, which indicates the variability of the market and the influence of different initial conditions on its structuring.

4.10.18 Dynamics of $f(\alpha)$ over time in two- and three-dimensional spaces

```
X = time_ser.index[window:length:tstep].values
X = np.expand_dims(X, axis=1)
X = np.repeat(a=X, repeats=nq.shape[0], axis=1)

Y = np.array(alpha)
Z = np.array(mfSpect)

plot_2d(X, Y, Z,
        subtitle_jpg='contour_f(alpha)',
        subtitle_fig=fr"Heat chart $f(\alpha)$",
        ylabel=r"$\alpha$",
        barlabel=r"$f(\alpha)$",
        cmap='hsv',
        lims=alpha)
```

Figs. 4.63 and 4.64 will show the dynamics of the multifractal spectrum $f(\alpha)$ changing over time for the time series of S&P 500, Hang Seng index, DAX, and BSE Sensex within two- and three-dimensional representations.

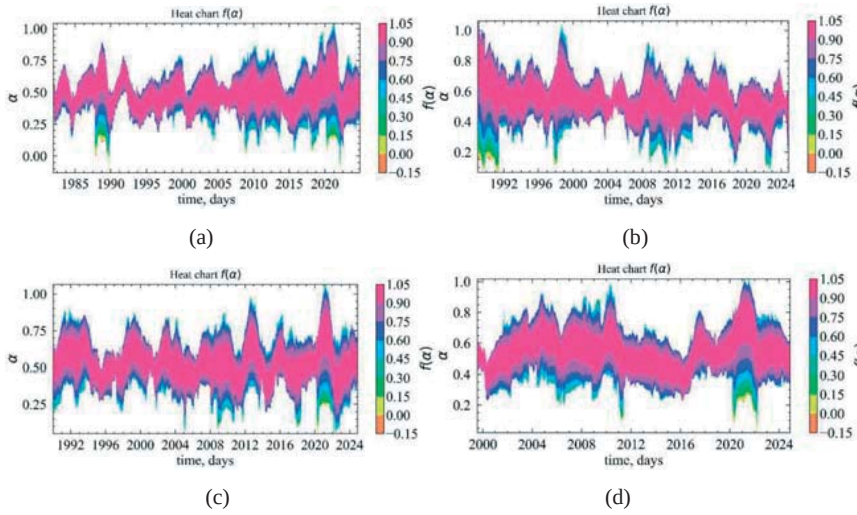


Fig. 4.63: Two-dimensional contour diagram of the dynamics of the multifractal spectrum $f(\alpha)$ changing with time for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

```
X = np.arange(window, length, tstep)
X = np.expand_dims(X, axis=1)
X = np.repeat(a=X, repeats=nq.shape[0], axis=1)

Y = np.array(alpha)
Z = np.array(mfSpect)

plot_3d(X, Y, Z, subtitle_jpg='3d_f(alpha)',
        ylabel=r"$\alpha$",
        zlabel=r"$f(\alpha)$",
        cmap='hsv')
```

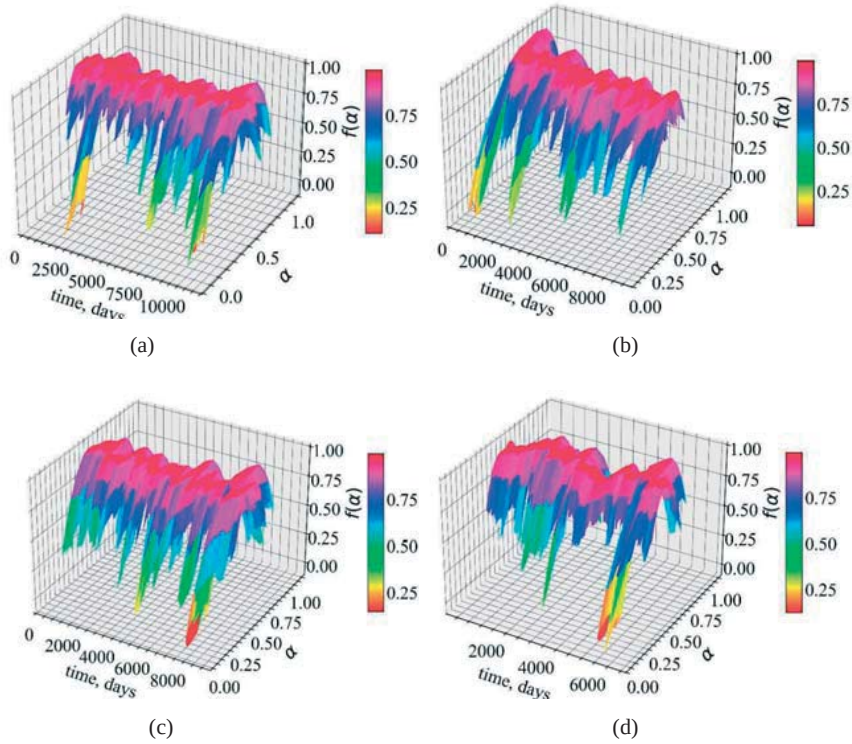


Fig. 4.64: Three-dimensional diagram of the dynamics of the multifractal spectrum $f(\alpha)$ changing over time for the time series of S&P 500 (a), Hang Seng index (b), DAX (c), and BSE Sensex (d)

As can be seen from the last figures (Fig. 4.63 and Fig. 4.64), the width of the multifractality spectrum changes in shape over time, and becomes wider during crisis events, as evidenced by such an indicator as $\Delta\alpha$. It can be seen that in the pre-crisis periods, the left-handed asymmetry increases, which characterizes fluctuations of a significant amplitude. The crises themselves represent a shift of $f(\alpha)$ to the right, indicating the dominance of fluctuations with small amplitudes. In any case, an increase in the width of the spectrum is an indicator of an increase in the degree of self-organization of the elements involved in the system under study. In other words, both $f(\alpha)$ and the previous indicators can be recommended

as indicators or precursors of crisis events. Further, it will be interesting to consider varieties of MF-DFA that, for example, take into account multifractal cross-correlations [5].

4.11 Conclusions on multifractal analysis

In this chapter, we analyzed price fluctuations of the stock indices using a spectrum of monofractal indicators. It has been shown that these methods are quite resistant to non-stationarity of the signal being analyzed. The stock market is characterized by rises and falls in fractal dimension, which indicates a variation in the efficiency of its development at different points in time. As already mentioned, a decline in the fractal dimension in a crisis or pre-crisis state of the market may indicate an increase in the degree of periodization (orderliness) of the system. An increase in the fractal dimension may be an indicator of increasing disorder.

Moreover, a spectrum of multifractal indicators was presented as indicators (precursor indicators) of crash events. It has been shown that the relevant indicators behave in a characteristic way (increase or decrease) in crisis and pre-crisis periods in the stock market. It can be seen that the stock indices are characterized by variability in the degree of multifractality, which indicates a change in the correlations of both small fluctuations and large ones on different spatial and temporal scales. Further research could be aimed at exploring the possibility of determining thresholds for the degree of multifractality that could be used to determine the degree of development of financial markets. Some emerging markets may be more developed than others because they are dynamic and growing, and therefore their range of multifractality may be the widest. Thus, in this case, it is possible to identify different stages of market development and model variables in the dynamics of complex systems as a function of the degree of multifractality.

5 Chaos-dynamic measures of complexity

Seemingly random fluctuations in complex systems often exhibit varying levels of complexity and chaos. Given limited data, it becomes difficult to determine the limits of their predictability. The analysis of such systems, the processes that determine their dynamics, and the theory of chaos have been considered in various fields, such as economics, finance, physics, etc. When it comes to analyzing, for example, DAX dynamics, knowledge of its completely random and, at the same time, deterministic processes can potentially explain time series fluctuations of various nature. Over the years, chaos theory has provided approaches to studying some interesting properties of time series. The most common ones are: correlation dimension, BDS test, Kolmogorov entropy, Lyapunov exponents, etc.

We will demonstrate how Lyapunov exponents make it possible to study the modes of chaotic and deterministic behavior.

5.1 Lyapunov exponents and sensitivity to initial conditions

The evolution of the system demonstrates sensitivity to initial conditions. This means that initially close trajectories that develop can quickly deviate from each other and have completely different outcomes. Accordingly, with small uncertainties that intensify extremely rapidly, long-term forecasts are impossible. On the other hand, in a system with points of attraction or stable points, the distance between them asymptotically decreases with time or with the number of points that tend to converge [38].

To represent the idea more accurately, consider two successive trajectories, $x(t)$ and the nearest neighbor of this trajectory with a slight displacement, $x(t) + \delta(t)$, where $\delta(t)$ represents a small deviation in time t, as shown in Fig. 5.1.

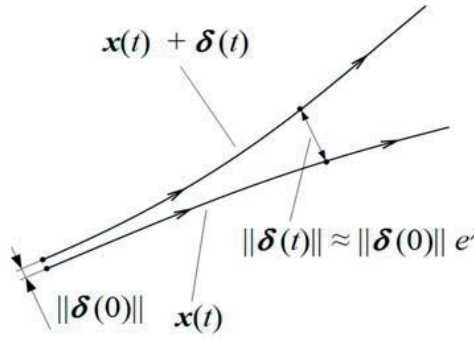


Fig. 5.1: Divergence of two initially close trajectories [21]

When the dynamics of two initially close trajectories are disrupted by a particular event, the distance between them can increase exponentially [166]:

$$\|\delta(t)\| \approx \|\delta(0)\| \exp(\lambda t), \tag{5.1}$$

where λ denotes the **Lyapunov exponent** (LE); $\delta(t)$ is the distance between the point under consideration and its nearest neighbor after t iterations; $\delta(0)$ is the initial distance between the point under consideration and its nearest neighbor at the initial time ($t = 0$).

The LE is a measure of the rate of exponential divergence of trajectories close to each other in the phase space of a dynamic system. In other words, the LE shows how quickly trajectories that start close to each other converge or diverge, measuring the degree of chaos in the system.

In cases where our system is n-dimensional, we have the same number of LEs. To determine them, let's consider the evolution of an infinitesimal sphere subjected to perturbations along different axes. By defining the magnitude of the perturbation along the i-axis as $\delta_i(t)$, we obtain n **Lyapunov exponents**:

$$\|\delta_i(t)\| \approx \|\delta_i(0)\| \exp(\lambda_i t), \text{ for } i = 1, \dots, n. \tag{5.2}$$

To determine whether the motion is periodic or chaotic, especially for large t, it is recommended to consider the contribution of the system to the **largest Lyapunov exponent** (LLE), since the diameter of the n-dimensional ellipsoid

begins to depend on it [21]. It is the LLE that is used to quantify the predictability of systems, since exponential divergence means that in a system where the initial perturbation was infinitesimal, the loss of predictability begins. However, it should be noted that other indicators also contain important information about the stability of the system, including the direction of convergence and divergence of trajectories [53].

The existence of at least one positive LE is usually considered a strong indicator of chaos. A positive LE means that initially close trajectories in phase space are sensitive to initial conditions and diverge exponentially fast. Negative LE corresponds to cases where trajectories remain close to each other, but this does not necessarily mean stability, and we should investigate our system in more detail. Zero or very close to zero values indicate that disturbances have little or no effect on the evolution of the trajectories of a dynamic system.

Due to the great interest in LE, more and more calculation tools are being developed. Unfortunately, there is still no generally accepted and universal method for estimating the entire spectrum of Lyapunov exponents from time series values. Some of the most common and popular algorithms were applied by Wolf et al. [24], Sano and Sawada [114], and later improved by Eckmann [94], Rosenstein [115], Parlitz [160], Balcerak, and others [102].

5.2 Methodology for calculating Lyapunov exponents using the Ekman method

First, according to the approach of Eckmann et al. [94], we have to reconstruct the attractor dynamics from the time series $\{x(i) \mid i = 1, \dots N\}$ with the embedding dimension d_E, and then construct a d_E-dimensional orbit representing the time evolution

$$\vec{X}(i) = \left[x(i), x\big(i+1\big), \dots, x\left(i + \big(d_E - 1\big)\right)\right], \text{ for } i = 1, \dots, N - d_E + 1.$$

Next, we need to determine the trajectories closest to $\vec{X}(i)$:

$$\left\|\vec{X}(i) - \vec{X}(j)\right\| = \max_{0 \le \alpha \le d_E - 1} |x(i+\alpha) - x(j+\alpha)|. \quad (5.3)$$

We sort $x(i)$ so that $x\big(\Pi(1)\big) \le x\big(\Pi(2)\big) \le \ldots \le x\big(\Pi(N)\big)$ and store the permutation Π and its inverse Π^{-1}. Next, we try to find the neighbors of $x(i)$ by looking at $k = \Pi^{-1}(i)$ and scan $x\big(\Pi(s)\big)$ at $s = k+1, k+2, \ldots$ and $k-1, k-2, \ldots$ until the condition $\big(\Pi(s)\big) - x(i) > r$ is satisfied. For the chosen embedding dimension $d_E > 1$, we choose the value of s under the condition

$$|x(\Pi(s)+\alpha) - x(j+\alpha)| \le r, \text{ для } \alpha = 0,1,\ldots,d_E - 1.$$

After the systems are reconstructed to dimension d_E, it is necessary to determine the matrix M_i of dimension $d_E \times d_E$, which describes the temporal evolution of vectors from the environment of the trajectory $\vec{X}(i)$ and how they map to the state $\vec{X}(i+1)$. The matrix M_i is obtained by searching for neighbors

$$M_i\left(\vec{X}(i) - \vec{X}(j)\right) \approx \vec{X}(i+1) - \vec{X}(j+1). \quad (5.4)$$

The vectors $\vec{X}(i) - \vec{X}(j)$ may not cover $\mathbb{R}^{d_E}$. In this case, such uncertainty can lead to false indicators that can lead to spurious analysis. To overcome such obstacles, the projection of trajectories is defined on a subspace of dimension $d_M \le d_E$. Thus, the space on which the dynamics takes place corresponds to the local dimension d_M, and d_E should be slightly larger than d_M to avoid the presence of false neighbors [73, 77]. It follows that the trajectory $\vec{X}(i)$ is associated with a d_M-dimensional vector

$$\vec{X}(i) = [x(i), x(i+\tau), \ldots, x(i+(d_M-1)\tau)] = [x(i), x(i+\tau), \ldots, x(i+d_E-1)], \quad (5.5)$$

where $\tau = (d_E - 1)/(d_M - 1)$. When $\tau > 1$, condition (5.4) is replaced by the following expression:

$$M_i\left(\vec{X}(i) - \vec{X}(j)\right) \approx \vec{X}(i+\tau) - \vec{X}(j+\tau). \quad (5.6)$$

The matrix M_i is determined by the least squares method. The last step is the QR decomposition to find the orthogonal matrices Q_i and the upper triangular matrices R_i for which

$$M_{1+i\tau}Q_i = Q_{i+1}R_{i+1},\ for\ i = 0,1,2,\dots\ .$$

As proposed by Eckmann [92, 94], knowing the K number of points on the attractor, the diagonal eigenvalues of the matrix R_i, and the sampling step Δt, the following equation can be used to find the k-th LE:

$$\lambda_k = \frac{1}{\Delta t}\frac{1}{\tau}\frac{1}{K}\sum_{i=0}^{K-1}\ln(R_i)_{kk}.$$

5.3 Application of the Rosenstein method to calculate the Lyapunov exponent

Rosenstein's algorithm [115] uses a time-delayed embedding reconstruction method that transfers the most important features of a multidimensional attractor into a single one-dimensional time series of some finite size N. For the time series, each vector $\vec{X}(i)$ will be represented similarly to the vector (5.5) with the dimension of the embedding d_E and the time delay τ. Then, on the recovered trajectory, we initialize the search in the state space for the nearest neighbor $\vec{X}(j)$ for the trajectory $\vec{X}(i)$:

$$\delta_i(0) = \min_{\vec{X}(i)}\left\|\vec{X}(i) - \vec{X}(j)\right\|,\ \text{for}\ |i-j| > \text{mean period},$$

where $\|\cdot\|$ is the Euclidean norm, $\vec{X}(j)$ is the nearest neighboring trajectory, and $\vec{X}(i)$ is the trajectory under consideration.

From (5.1), we already know that the distance between states $\vec{X}(i)$ and $\vec{X}(j)$ grows with time according to a power law, where λ is a good approximation of the LLE. For further estimates, let us consider the logarithm of the distance on the trajectory $\ln \delta_i(k) \approx \lambda(k \cdot \Delta t) + \ln c_i$, where $\delta_i(k)$ is the distance between the i-th pair of nearest neighbors defined by Eq. (5.6) after k time steps, c_i is the initial distance between them, and Δt is the time interval between measurements (time series sampling period).

The subsequent result of this algorithm is a function of time:

$$y(k,\Delta t)=\frac{1}{M\Delta t}\sum_{i=1}^{M}\ln\delta_i(k),$$

where $M=N-(d_E-1)\tau$ is the size of the reconstructed time series, and $\delta_i(k)$ is the i-th line whose slope is approximately equal to the LLE. Then it is proposed to calculate the LLE as the slope of the most linear section. Finding such a section turns out to be a non-trivial task. Despite this problem, Rosenstein's method is simple to implement and compute.

5.4 Practical calculations of LLE and LEs

Let's see how these approaches can be used to calculate the corresponding chaos-dynamic indicators. First, let's import the necessary libraries.

```
import matplotlib.pyplot as plt
import numpy as np
import neurokit2 as nk
import yfinance as yf
import pandas as pd
import scienceplots
from tqdm import tqdm

%matplotlib inline
```

Next, let's configure the output format of the figures.

```
plt.style.use(['science', 'notebook', 'grid'])

size = 22
params = {
'figure.figsize': (8, 6),
'font.size': size,
'lines.linewidth': 2,
'axes.titlesize': 'small',
'axes.labelsize': size,
'legend.fontsize': size,
'xtick.labelsize': size,
'ytick.labelsize': size,
"font.family": "Serif",
"font.serif": ["Times New Roman"],
'savefig.dpi': 300,
'axes.grid': False
}

plt.rcParams.update(params)
```

Let's define the `transformation()` function to perform the transformation of a series to yields or standardized values:

```
def transformation(signal, ret_type):

    for_rec = signal.copy()

    if ret_type == 1:
       pass
    elif ret_type == 2:
        for_rec = for_rec.diff()
    elif ret_type == 3:
        for_rec = for_rec.pct_change()
    elif ret_type == 4:
        for_rec = for_rec.pct_change()
        for_rec -= for_rec.mean()
        for_rec /= for_rec.std()
    elif ret_type == 5:
        for_rec = for_rec.pct_change()
        for_rec -= for_rec.mean()
        for_rec /= for_rec.std()
        for_rec = for_rec.abs()
    elif ret_type == 6:
        for_rec -= for_rec.mean()
        for_rec /= for_rec.std()

    for_rec = for_rec.dropna().values

    return for_rec
```

Let's define a function for plotting pairwise graphs:

```
def plot_pair(x_values,
              y1_values,
              y2_values,
              y1_label,
              y2_label,
              x_label,
              file_name, clr="magenta"):

    fig, ax = plt.subplots()

    ax2 = ax.twinx()

    ax2.spines.right.set_position(("axes", 1.03))

    p1, = ax.plot(x_values,
                  y1_values,
"b-", label=fr"{y1_label}")
    p2, = ax2.plot(x_values,
                   y2_values,
                   color=clr,
                   label=y2_label)

    ax.set_xlabel(x_label)
```

```
    ax.set_ylabel(f"{y1_label}")

    ax.yaxis.label.set_color(p1.get_color())
    ax2.yaxis.label.set_color(p2.get_color())

    tkw = dict(size=4, width=1.5)
    ax.tick_params(rotation=45, axis='x', **tkw)
    ax.tick_params(axis='y', colors=p1.get_color(), **tkw)
    ax2.tick_params(axis='y', colors=p2.get_color(), **tkw)


    ax2.legend(handles=[p1, p2])

    plt.savefig(file_name +".jpg")

    plt.show();
```

5.4.1 Calculations of Lyapunov exponents using the sliding window procedure

For further calculations, we will use the `neurokit2` library. The key function for obtaining the relevant indicators is `complexity_lyapunov()`. It provides access to calculations according to the following algorithms:

- Rosenstein et al.'s (1993).
- **Makowski** is a custom modification of Rosenstein's algorithm, using KDTree for more efficient nearest neighbors computation. Additionally, the LLE is computed as the slope up to the changepoint of divergence rate (the point where it flattens out), making it more robust to the length trajectory parameter.
- Eckmann et al. (1986).

Let's take a closer look at its syntax:

```
complexity_lyapunov(signal, delay=1, dimension=2, method='rosenstein1993', separation='auto', **kwargs)
```

Parameters:

- **signal** (*Union[list, np.array, pd.Series]*) – the signal (i.e., a time series) in the form of a vector of values;
- **delay** (*int*) – time delay (often denoted *Tau* τ, sometimes referred to as *lag*) in samples;

- **dimension** (*int*) – embedding dimension (d_E, sometimes referred to as *d* or *order*). if method is "eckmann1986", larger values for dimension are recommended;
- **method** (*str*) – the method that defines the algorithm for computing LE. Can be one of "rosenstein1993", "makowski", or "eckmann1986";
- **len_trajectory** (*int*) – applies when method is "rosenstein1993". The number of data points in which neighboring trajectories are followed;
- **matrix_dim** (*int*) – applies when method is "eckmann1986". Corresponds to the number of LEs to return;
- **min_neighbors** (*int*, *str*) – applies when method is "eckmann1986". Minimum number of neighbors. If "default", min(2 * matrix_dim, matrix_dim + 4) is used;
- ****kwargs** (*optional*) – other arguments to be passed to signal_psd() for calculating the minimum temporal separation of two neighbors.

Returns:

- **lle** (*float*) – an estimate of the largest Lyapunov exponent (LLE) if method is "rosenstein1993", and an array of LEs if "eckmann1986";
- **info** (*dict*) – a dictionary containing additional information regarding the parameters used to compute LLE.

Before calculations, let's update the `neurokit2` library:

```
!pip install --upgrade neurokit2
```

5.4.1.1 Calculation of the LLE based on the Rosenstein method

First, let's perform the calculations for our stock indices:

```
signal = time_ser.copy()
ret_type = 1 # type of a series:
             # 1 - initial
             # 2 - detrending (difference between present and previous values
)
             # 3 – initial returns
             # 4 – standardized returns
```

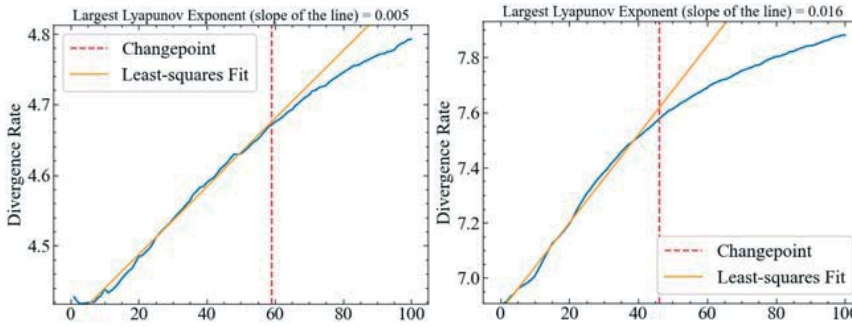
```
            # 5 – absolute values (volatility)
            # 6 – standardized series

time_ser_ret = transformation(signal, ret_type)
```

Next, let's define the following parameters:

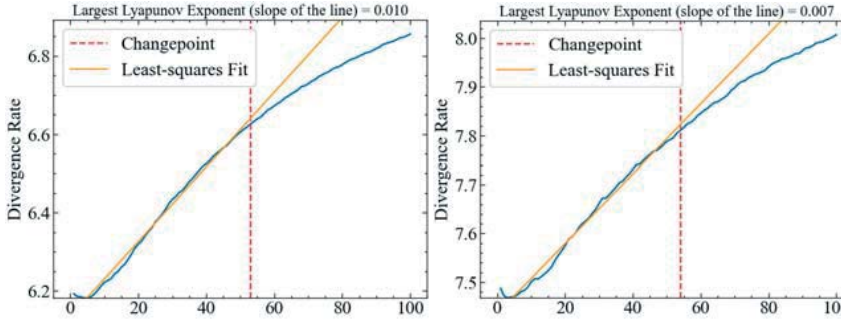
```
d_E = 3                    # embedding dimension
tau = 10                   # time delay
approach_lyap = "makowski" # method for LLE
max_len = "auto"           # set the maximum trajectory length to 10 times th
e delay
sep = "auto"               # estimation of the average period as the inverse
of the average frequency of the power spectrum
```

and visualize the trajectory divergence of the reconstructed phase space of S&P 500, Hang Seng index, DAX, BSE Sensex, representing the calculated LLEs (see Fig. 5.3):

```
lle, _ = nk.complexity_lyapunov(signal=time_ser_ret,
                                method=approach_lyap,
                                dimension=d_E,
                                delay=tau,
                                max_length=max_len,
                                separation=sep,
                                show=True)
```



(a) (b)



(c) (d)

Fig. 5.3: Diagram of trajectory divergence of the reconstructed phase space of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), representing the calculated LLEs

Fig. 5.3 shows a typical plot (solid curve) of the average trajectory divergence versus time Δt; the orange line has a slope equal to the theoretical value of λ_{max}. The short blue section before the red dashed line is used to extract the largest Lyapunov exponent. As we can see, the curve changes at longer time periods because the system is limited in phase space and the average divergence cannot exceed the "length" of the attractor. The resulting LE indicates that the stock indices are on the borderline between chaos and stability, i.e., the divergence index of the series dynamics is balanced by convergence.

As we have already seen, complex systems are volatile, and the system can show either convergence or divergence or complete immutability over time.

Next, let's look at the dynamics of the system under study over time using the sliding window procedure. Let us define the following parameters:

```
window = 500
tstep = 1
length = len(time_ser)
ret_type = 1 # type of a series:
             # 1 - initial,
             # 2 - detrending (difference between present and previous values
)
             # 3 – initial returns,
             # 4 – standardized returns,
             # 5 – absolute values (volatility)
             # 6 – standardized series

d_E = 3 # embedding dimension
```

```
tau = 1 # time delay
approach_lyap = "makowski" # method for LLE: rosenstein1993, makowski
max_len = "auto" # set the maximum trajectory length to 10 times the delay: a
uto
sep = "auto" # estimation of the average period as the inverse of the average
 frequency of the power spectrum

LLE = []                # array to save LLE
```

Now you can start the sliding window procedure:

```
for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)
    lle, _ = nk.complexity_lyapunov(signal=fragm,
                                    method=approach_lyap,
                                    dimension=d_E,
                                    delay=tau,
                                    max_length=max_len,
                                    separation=sep,
                                    show=False)

    LLE.append(lle)
```

Save the results to a text file:

```
name = f"LLE_name={symbol}_window={window}_step={tstep}_rettype={ret_type}_\
    d_E={d_E}_tau={tau}_approach={approach_lyap}_max_len={max_len}_separation
={sep}.txt"

np.savetxt(name, LLE)
```

Define the parameters for further figures:

```
# notation of the Lyapunov exponent in the figure legend
label_lyap = r'$\lambda_{max}$'

# title of the figure
file_name = f"LLE_name={symbol}_window={window}_step={tstep}_rettype={ret_typ
e}_\
    d_E={d_E}_tau={tau}_approach={approach_lyap}_max_len={max_len}_separation
={sep}"

# the color of an indicator
color ='red'
```

and plot the dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their LLEs according to Rosenstein method (see Fig. 5.4):

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          LLE,
          ylabel,
          label_lyap,
```

```
        xlabel,
        file_name,
        color)
```

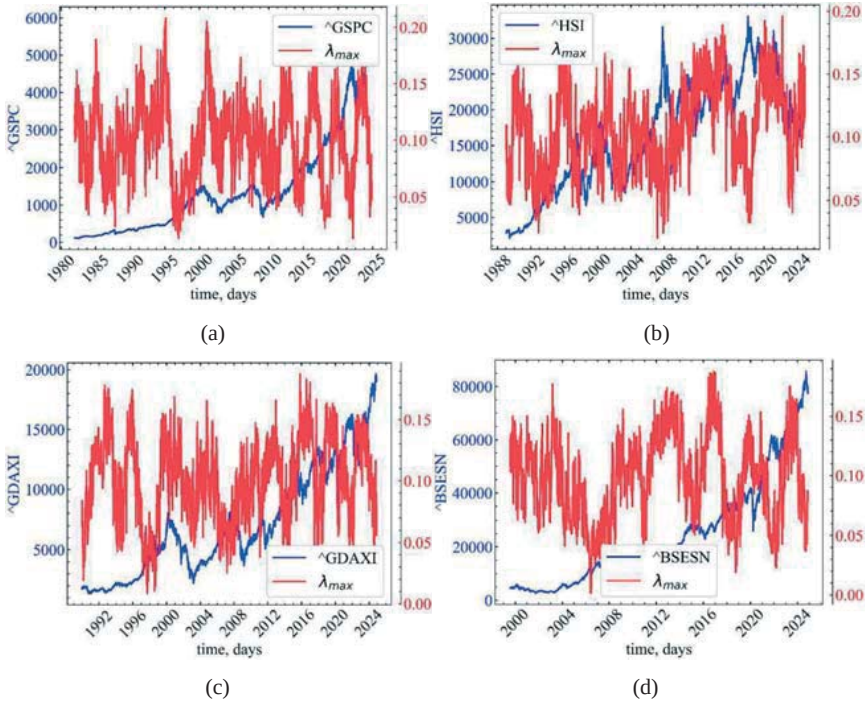


Fig. 5.4: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their LLEs according to Rosenstein method

In Fig. 5.4 we can see that the LLEs begin to decline in crisis and pre-crisis states, indicating an increase in the correlation of the dynamics under study. At the time of the crisis, the LLE begins to rise, indicating an increase in divergence during crisis periods.

5.4.1.2 Calculation of Lyapunov exponents based on the Eckmann method

Let's define the following parameters:

```
window = 500
tstep = 1
length = len(time_ser)
ret_type = 1
d_E = 4 # dimensionality of the original space (number of exponents)
```

```
d_M = 3 # dimension of the subspace embedding

approach_lyap = "eckmann1986" # method for LLE calculations
sep = "auto" # estimation of the average period as the inverse of the average
 frequency of the power spectrum
min_neighb = "default" # min(2 * matrix_dim, matrix_dim + 4)

LE = []                          # array for saving LE
```

Now let's move on to the calculations:

```
for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)
    le, _ = nk.complexity_lyapunov(signal=fragm,
                                   method=approach_lyap,
                                   dimension=d_E,
                                   matrix_dim=d_M,
                                   min_neighbors=min_neighb,
                                   separation=sep,
                                   show=False)

    LE.append(le)
```

Save the results to text files:

```
LE = np.array(LE)

for i in range(d_E):
    np.savetxt(f"LE number={i+1}_name={symbol}_window={window}_step={tstep}_r
ettype={ret_type}_\
    d_E={d_E}_d_M={d_M}_approach={approach_lyap}_min_neighbors={min_neighb}_s
eparation={sep}.txt", LE[i])
```

Let's visualize the dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their spectrum of LEs according to Eckmann method (see Fig. 5.5):

```
fig, ax = plt.subplots(LE.shape[1]+1, 1, sharex=True)

ax[0].plot(time_ser.index[window:length:tstep], time_ser.values[window:length
:tstep], label=symbol)
ax[0].set_ylabel(symbol)
ax[0].legend()

for i in range(1, LE.shape[1]+1):
    ax[i].plot(time_ser.index[window:length:tstep], LE[:, i-1], color='red',
label=fr'$\lambda_{i}$')
    ax[i].set_ylabel(fr"$\lambda_{i}$")
    ax[i].legend()

ax[-1].set_xlabel(xlabel)
fig.subplots_adjust(hspace=0)

plt.savefig(f"LE name={symbol}_window={window}_step={tstep}_rettype={ret_type
```

```
}_\
    #d_E={d_E}_d_M={d_M}_approach={approach_lyap}_min_neighbors={min_neighb}_
separation={sep}.jpg")
plt.show();
```

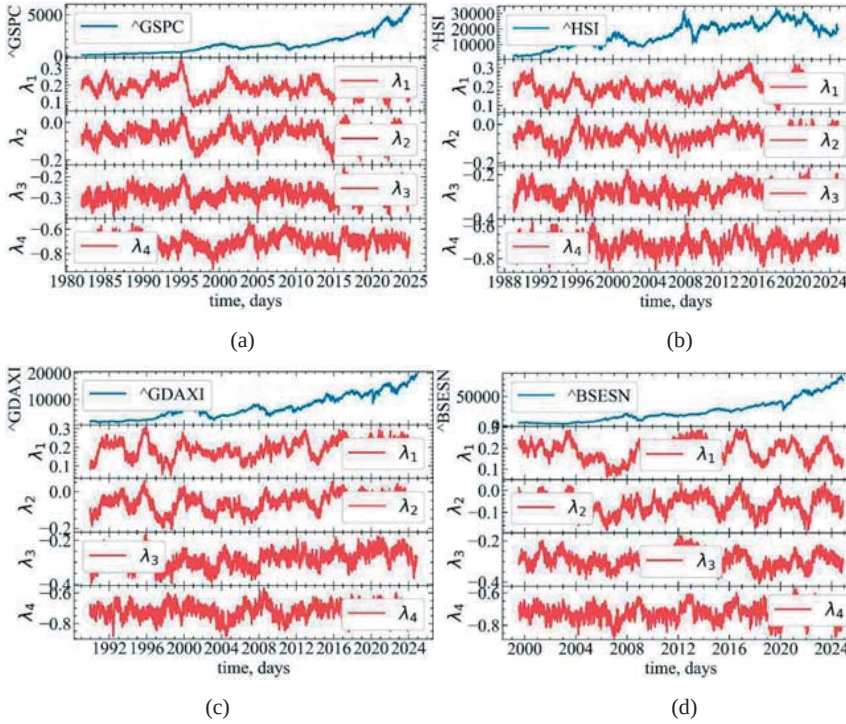


Fig. 5.5: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their spectrum of LEs according to Eckmann method

As shown in Fig. 5.5, the range of LEs reacts in a special way to stock market crises. It can be seen that, first, λ decreases in the pre-crisis periods and increases during the crisis. This dynamics is especially characteristic of the crises of 1997, 2001, 2008, 2011, 2015, and 2020. In the pre-crisis periods, there is a convergence of trajectories in the phase space of the system, which indicates an increase in its orderliness. The crisis and post-crisis periods themselves are characterized by divergence, i.e., divergence of the system's trajectories. Secondly, it is clear that as we go down from the 1st to the 4th LE, we gradually lose

information about the dynamics of the system. That is, the first largest indicators seem to be the most informative in this case. Perhaps, in this case, it makes sense to consider only the largest LE.

Save the indicator in a text file:

```
name = f"LE Eckman name={symbol}_window={window}_step={tstep}_rettype={ret_ty
pe}_\
    #d_E={d_E}_d_M={d_M}_approach={approach_lyap}_min_neighbors={min_neighb}_
separation={sep}.txt"

np.savetxt(name, LE[:, 0])
```

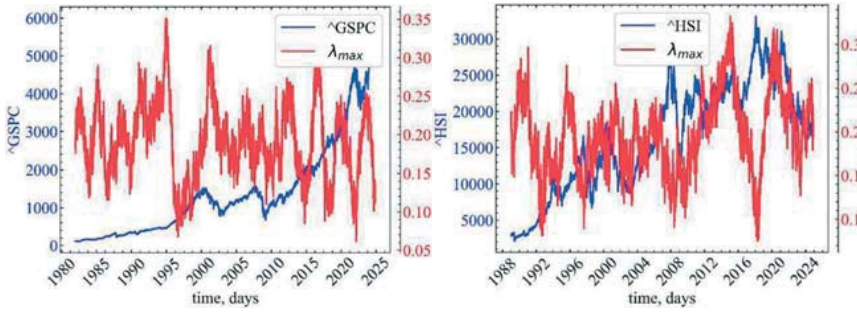
Define the parameters for saving figures:

```
# labeling of the Lyapunov exponent in the figure legend
label_lyap = r'$\lambda_{max}$'

# figure title
file_name = f"LE Eckmann name={symbol}_window={window}_step={tstep}_rettype={
ret_type}_\
    #d_E={d_E}_d_M={d_M}_approach={approach_lyap}_min_neighbors={min_neighb}_
separation={sep}"

# color of an indicator
color ='red'
```

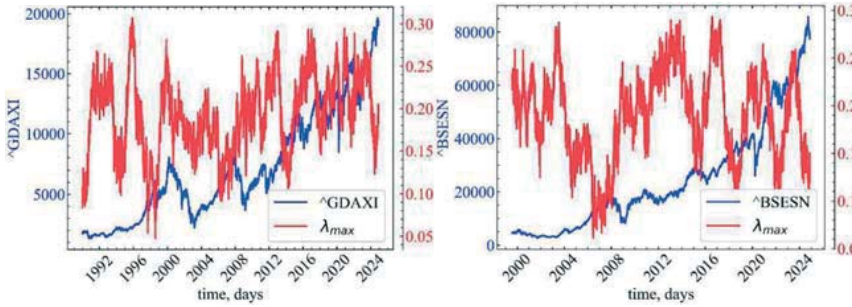
Let's visualize LLE indicator:

```
plot_pair(time_ser.index[window:length:tstep], time_ser.values[window:length:
tstep], LE[:, 0], ylabel, label_lyap, xlabel, file_name, color)
```

Fig. 5.6 shows the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and the LLE based on the Eckmann method.



(a) (b)



(c) (d)

Fig. 5.6: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their LLEs according to the Eckmann method

5.5 Conclusions on Lyapunov exponents

Chaos theory and its tools remain a huge challenge for researchers in various fields of science. In the world of LEs, there is a growing interest in their definition, numerical methods, and application to various complex systems. To summarize, the LLE allows us to establish:

- the region of sensitivity to initial conditions;
- the region of chaos;
- the region of stability.

6 Network analysis of crisis phenomena

In this section, we will demonstrate modern methods for converting time series into a network (graph) with further investigation of the corresponding spectral and topological measures of complexity. We will also show that these measures can be compared with the dynamics of the initial time series (hence graphodynamics) and if they are informative about possible changes in the series itself, then they can be used to build indicators of the characteristic dynamics of complex systems.

Most complex systems inform their structural and dynamic nature by generating a sequence of certain characteristics that can be represented by time series. In recent years, interesting algorithms for converting time series into a network have been developed, which allows expanding the range of known characteristics of time series even to network ones [9, 13, 17]. Recently, several approaches to converting time sequences into complex network-like mappings have been proposed. These methods can be roughly divided into three classes [141]. The first is based on the study of the "visibility" of successive values of the time series and is called the Visibility Graph (VG) [99, 141].

The second analyzes the mutual approximation of different segments of the time sequence and uses the technique of recurrence analysis [141]. A recurrence diagram displays the existing recurrence of phase trajectories in the form of a binary matrix, the elements of which are ones or zeros, depending on whether the selected points of the phase space of a dynamical system are close (recurrent) with a given precision or not. A recurrence diagram is easily transformed into an adjacency matrix, according to which the spectral and topological characteristics of the graph are calculated [15].

Finally, if we base the formation of connections of the elements of the graph on the correlation relations between them, then we get a correlation graph [141]. To construct and analyze the properties of a correlation graph, it is necessary to form an adjacency matrix from the correlation matrix. To do this, it is necessary to

enter a value that for the correlation field will serve as the distance between the correlated agents. Such a distance can be represented as $\sqrt{2(1-C_{ij})}$, where C_{ij} is the correlation coefficient between the two assets. Thus, if the correlation coefficient between two assets is significant, then the distance between them is small, and starting from a certain critical value of d_{cr}, the assets can be considered related on the graph. For the adjacency matrix, this means that they are adjacent on the graph. Otherwise, the assets are not contiguous. In this case, the coherence condition of the graph is mandatory condition.

The main goal of such methods is to accurately reproduce the information stored in time series in an alternative mathematical structure, so that powerful graph theory tools can later be used to characterize time series from a different perspective in order to bridge the gap between nonlinear analysis of time series, dynamical systems, and graph theory.

In this chapter, we will consider only the algorithm of the VG.

6.1 Methods for converting time series into visibility graphs

Visibility graphs (VG) are based on the simple mapping of time series to a network domain, where each observation is a vertex in a complex network. Two vertices i and j are connected by an edge if the following condition applies to them [98]:

$$x_k < x_j + (x_i - x_j)(t_j - t_k)/(t_j - t_i),$$

where x_k presents a certain obstacle that should not be present for the two vertices to be linked by a path.

The **adjacency matrix** (A_{ij}) of the represented non-directional and unweighted VG can be represented as

$$A_{ij}^{VG} = A_{ji}^{VG} = \prod_{k=i+1}^{j-1} H\left[x_k < x_j + (x_i - x_j)(t_j - t_k)/(t_j - t_i)\right],$$

where $H(\cdot)$ is the Heaviside function.

The **Horizontal visibility graph** (HVG) is a simplified version of this algorithm [30]. For the time series under study, the sets of vertices VG and HVG are the same, while the set of edges of the HVG displays the mutual horizontal visibility of the two observations x_i and x_j. That is, it is possible to construct an edge (i, j) if $x_k < \min(x_i, x_j)$ for all k at $t_i < t_k < t_j$ so that

$$A_{ij}^{VG} = A_{ji}^{VG} = \prod_{k=i+1}^{j-1} H\,(x_i - x_k) H(x_j - x_k).$$

VG and HVG capture essentially the same properties of the system under study, since HVG is a subgraph of VG with the same set of vertices, but possesses only a subset of VG edges. Note that VG is invariant with respect to the superposition of linear trends, while HVG is not.

6.1.1 Library ts2vg

To further construct the classic **Visibility Graph (VG)** or its horizontal counterpart, we will use the `ts2vg` library. The `ts2vg` package provides a high-performance implementation of algorithms for constructing visibility graphs from time series data, first introduced by Lucas Lacassa et al. [99].

Visibility graphs and some of their properties (e.g., power distributions) are computed quickly and efficiently, even for time series with millions of observations. An efficient divide-and-conquer algorithm is used to calculate graphs whenever possible [173].

6.1.1.1 ts2vg installation

The latest released version of `ts2vg` is available on PyPI and can be easily installed by running the following command:

```
!pip install ts2vg
```

6.1.1.2 Supported graph types

6.1.1.2.1 Main types

- Natural visibility graph [99] (`ts2vg.NaturalVG`);

- Horizontal visibility graph [30] (`ts2vg.HorizontalVG`).

6.1.1.2.2 Available variations

- Weighted visibility graph (via the parameter `weighted`);
- Directional visibility graph (via the parameter `directed`);
- Parametric visibility graph [83] (via the parameters `min_weight` and `max_weight`);
- Limited Penetrable Visibility Graphs [131, 159] (via the parameter `penetrable_limit`).

Please note that several graph options can be combined and used at the same time. More detailed documentation can be found on the website of the `ts2vg` library.

6.1.1.2.3 Compatibility with other libraries

The resulting graphs can be easily converted to graph objects from other common Python graph libraries such as `igraph`, `NetworkX`, and `SNAP` for further analysis.

For this, the following methods are provided:

- `as_igraph()`;
- `as_network()`;
- `as_snap()`.

6.2 Network measures estimation

First, import the necessary modules for further work:

```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import numpy as np
import neurokit2 as nk
import yfinance as yf
import pandas as pd
import networkx as nx
import scienceplots

from sklearn import preprocessing
from tqdm import tqdm
```

```
from ts2vg import NaturalVG, HorizontalVG
from scipy.spatial import distance

%matplotlib inline
```

And we will configure the figures for the output:

```
plt.style.use(['science', 'notebook', 'grid'])

size = 22
params = {
'figure.figsize': (8, 6),
'font.size': size,
'lines.linewidth': 2,
'axes.titlesize': 'small',
'axes.labelsize': size,
'legend.fontsize': size,
'xtick.labelsize': size,
'ytick.labelsize': size,
"font.family": "Serif",
"font.serif": ["Times New Roman"],
'savefig.dpi': 300,
'axes.grid': False
}

plt.rcParams.update(params)
```

Let us consider the possibility of using graphodynamic indicators as indicators or indicators-harbingers of crisis phenomena.

As before, let's define a function for transforming a series (its standardization or finding profitability). To do this, declare the `transformation()` function, which will take a time signal, a series type, as input and return its transformation. As the authors' previous studies have shown, the original representation of the time series provides the most informative representation for graph construction. Nevertheless, we assume that, for example, the profitability of a physical signal may have a better graph representation, which is why we define this function in this chapter.

```
def transformation(signal, ret_type):

    for_graph = signal.copy()

    if ret_type == 1:
       pass
    elif ret_type == 2:
       for_graph = for_graph.diff()
    elif ret_type == 3:
```

```
        for_graph = for_graph.pct_change()
    elif ret_type == 4:
        for_graph = for_graph.pct_change()
        for_graph -= for_graph.mean()
        for_graph /= for_graph.std()
    elif ret_type == 5:
        for_graph = for_graph.pct_change()
        for_graph -= for_graph.mean()
        for_graph /= for_graph.std()
        for_graph = for_graph.abs()
    elif ret_type == 6:
        for_graph -= for_graph.mean()
        for_graph /= for_graph.std()

    for_graph = for_graph.dropna().values

    return for_graph
```

We return the same output signal. Next, set the parameters of the graph under study. For further calculations, we will use the same values of the time window, step, and series type.

```
signal = time_ser.copy()
ret_type = 1 # type of a series:
# 1 – initial
# 2 - detrending (difference between present and previous values)
# 3 – initial returns
# 4 – standardized returns
# 5 – absolute values (volatility)
# 6 – standardized series

for_graph = transformation(signal, ret_type) # series transformation

window = 500           # sliding window width
tstep = 5              # sliding window time step
graph_type = 'classic' # graph type: classic, horizontal

length = len(time_ser)
```

6.2.1 Graph construction

Since constructing a graph for the entire time series can take quite a long period of time, we will only plot a visibility graph for its fragment. To do this, we will determine the parameters of `index_begin` and `index_end` that will indicate the beginning of the construction and the ending. For the classic VG we have the following visibility connections (see Fig. 6.1):

```
index_begin = 3700
index_end = 5700
```

```
date = date_in_num[index_begin:index_end]

if graph_type == 'classic':
    g = NaturalVG(directed=None).build(for_graph[index_begin:index_end], xs=d
ate)
    pos1 = g.node_positions()
    nxg = g.as_networkx()
if graph_type == 'horizontal':
    g = HorizontalVG(directed=None).build(for_graph[index_begin:index_end], x
s=date)
    pos1 = g.node_positions()
    nxg = g.as_networkx()

graph_plot_options = {
'with_labels': False,
'node_size': 0,
'node_color': [(0, 0, 0, 1)],
'edge_color': [(0, 0, 0, 0.15)],
}

fig, ax = plt.subplots(1, 2, figsize=(15, 8))

nx.draw_networkx(nxg, ax=ax[0], pos=pos1, **graph_plot_options)
ax[0].tick_params(bottom=True, labelbottom=True)
ax[0].plot(time_ser.index[index_begin:index_end], for_graph[index_begin:index
_end], label=fr"{ylabel}")
ax[0].set_title(f'Visibility connections of {ylabel}', pad=10)
ax[0].set_xlabel(xlabel)
ax[0].set_ylabel(f"{ylabel}")
ax[0].legend(loc='upper right')
ax[0].tick_params(axis='x', labelrotation=45)


ax[1].set_title(f'Graph representation of {symbol}', pad=10)

# determine the position of the nodes on the graph
pos2 = nx.spring_layout(nxg, k=0.15, iterations=100)

# calculate degree centrality
degCent = nx.degree_centrality(nxg)

# create a list of vertex sizes based on degree centrality
node_sizes = [v*100 for v in degCent.values()]

# colors of nodes based on their degree of centrality
node_colors = [v for v in degCent.values()]

# build a graph
nx.draw_networkx(nxg, ax=ax[1], pos=pos2,
                 node_size=node_sizes,
                 node_color=node_colors,
                 with_labels=False,
                 cmap=plt.get_cmap('plasma'))
```

```
# assign a minimum and maximum value
# of degree centrality to build the heat scale
vmin = np.asarray(list(degCent.values())).min()
vmax = np.asarray(list(degCent.values())).max()

sm = plt.cm.ScalarMappable(cmap=plt.get_cmap('plasma'),
                           norm=plt.Normalize(vmin=vmin, vmax=vmax))
cb = plt.colorbar(sm, ax=ax[1])
cb.set_label('Degree centrality')

plt.savefig(f"Time_ser_connections_symbol={symbol}_idx_beg={index_begin}_\
            idx_end={index_end}_sertype={ret_type}_network_type={graph_type}.
jpg", bbox_inches="tight", dpi=1000)
```

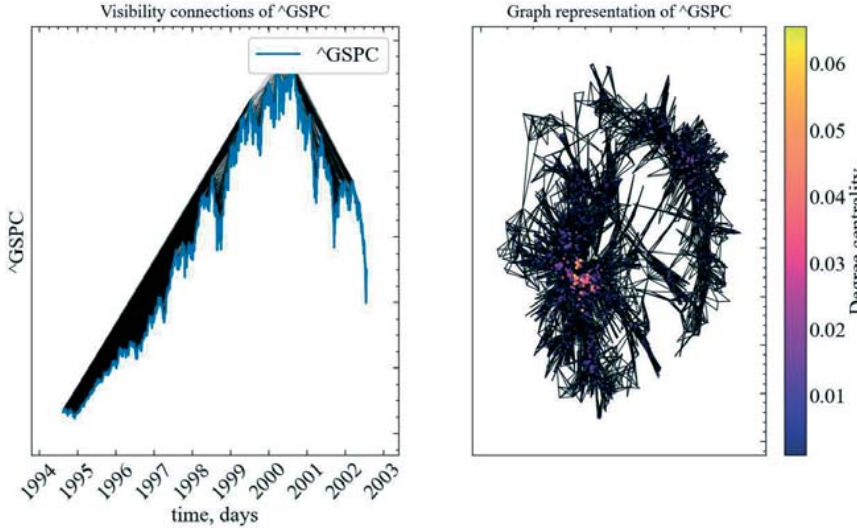


Fig. 6.1: A natural visibility graph before the crash of 2001 in the S&P 500 market and a network representation of this fragment

As we can see from the figure, the pre-crisis period of 2001 is characterized by a significant degree of visibility. The figure on the right shows that a cluster characterized by a high degree of centrality begins to form in the stock market network, which can serve as a harbinger of a crash.

6.2.2 Sliding window procedure for network analysis

Next, we will observe how the properties of the network change over time. To do this, we will use the well-known procedure of a moving window. As part of this procedure, we will investigate the graphodynamics of both spectral and topological indicators.

To construct the pairwise dynamics of a particular indicator and the series under study, we determine the function `plot_pair`:

```
def plot_pair(x_values, y1_values, y2_values, y1_label, y2_label, x_label, fi
le_name, clr="magenta"):

    fig, ax = plt.subplots()

    ax2 = ax.twinx()
    ax2.spines.right.set_position(("axes", 1.03))

    p1, = ax.plot(x_values,
                  y1_values,
"b-", label=fr"{y1_label}")
    p2, = ax2.plot(x_values,
                   y2_values,
                   color=clr,
                   label=y2_label)

    ax.set_xlabel(x_label)
    ax.set_ylabel(f"{y1_label}")
    ax.yaxis.label.set_color(p1.get_color())
    ax2.yaxis.label.set_color(p2.get_color())

    tkw = dict(size=2, width=1.5)

    ax.tick_params(axis='x', rotation=35, **tkw)
    ax.tick_params(axis='y', colors=p1.get_color(), **tkw)
    ax2.tick_params(axis='y', colors=p2.get_color(), **tkw)
    ax2.legend(handles=[p1, p2])

    plt.savefig(file_name +".jpg")
    plt.show();
```

6.2.3 Spectral characteristics

Spectral graph theory is based on the study of the properties of graphs through eigenvalues or eigenvectors of the adjacency matrix A or the **Laplace matrix** (Laplacian matrix) L [65].

Recall that the standard Laplace matrix for the graph G is defined as

$$L = D - A,$$

where D is the diagonal matrix G, where the i-th diagonal element is the degree of vertex i in G [117], and A is the adjacency matrix G. In this chapter, we present the spectral characteristics for the normalized Laplace matrix [143], which is defined as

$$\hat{L} = D^{-1/2}LD^{-1/2}.$$

If λ is the eigenvalue of $\hat{L}$, then $\lambda \in [0,2]$ [65]; that is, by normalizing the Laplace matrix, we normalize the eigenvalues.

```
AlgebraicCon = []
GraphEnergy = []
SpecMoment_3 = []
SpecRadius = []
SpecGap = []
NaturalConnectivity = []

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

    if graph_type == 'classic':
        g = NaturalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()
    if graph_type == 'horizontal':
        g = HorizontalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()

# spectrum of eigenvalues of the adjacency matrix
    adj_spectrum = nx.adjacency_spectrum(nxg).real

# sort eigenvalues in ascending order
    sorted_adj_spectrum = np.sort(adj_spectrum)

# calculate algebraic connectivity
    alg_con = nx.algebraic_connectivity(nxg, normalized=True, method='tracemi
n_lu')

# calculate the energy of the graph
    graph_en = np.sum(np.abs(adj_spectrum))

# calculate the spectral gap
    spec_gap = sorted_adj_spectrum[-1] - sorted_adj_spectrum[-2]

# calculate the spectral radius
    spec_rad = np.max(np.abs(adj_spectrum))
```

```
# calculate the spectral moment
    spec_mom_3 = np.mean(adj_spectrum **3)

# calculate natural connectivity
    nat_con = np.log(np.mean(np.exp(adj_spectrum)))

    AlgebraicCon.append(alg_con)
    GraphEnergy.append(graph_en)
    SpecRadius.append(spec_rad)
    SpecGap.append(spec_gap)
    SpecMoment_3.append(spec_mom_3)
    NaturalConnectivity.append(nat_con)
```

Save initial values to a text document. We also prepare labels for figures and names of saved measures:

```
ind_names = ['algebraic_conn', 'graph_energy', 'spectral_radius',
'spectral_grap', 'spectral_moment_3', 'natural_connectivity']

indicators = [AlgebraicCon, GraphEnergy, SpecRadius,
              SpecGap, SpecMoment_3, NaturalConnectivity]

measure_labels = [r'$\lambda_2$', r'$E$', r'$R$', r'$\delta$', r'$m_3$', r'$N
_c$']

file_names = []

for i in range(len(ind_names)):
    name = f"{ind_names[i]}_symbol={symbol}_wind={window}_step={tstep}_series
type={ret_type}_graph_type={graph_type}"
    np.savetxt(name +".txt", indicators[i])
    file_names.append(name)
```

6.2.3.1 Algebraic connectivity

Regarding the eigenvalues of the Laplace matrix, one of the main characteristics we can obtain is the **algebraic connectivity** λ_2 of the graph, which corresponds to the second smallest eigenvalue of the matrix. This indicator reflects the number of disconnected components. For an unconnected graph, λ_2 will be zero, and for a graph with a higher density of connections, λ_2 will be higher. Using this indicator, it is possible to determine the fault tolerance and synchronization of the system under study.

```
plot_pair(time_ser.index[window:length:tstep], time_ser.values[window:length:
tstep], indicators[0], ylabel, measure_labels[0], xlabel, file_names[0], clr=
"magenta")
```

In Fig. 6.2 is illustrated the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their algebraic connectivity.

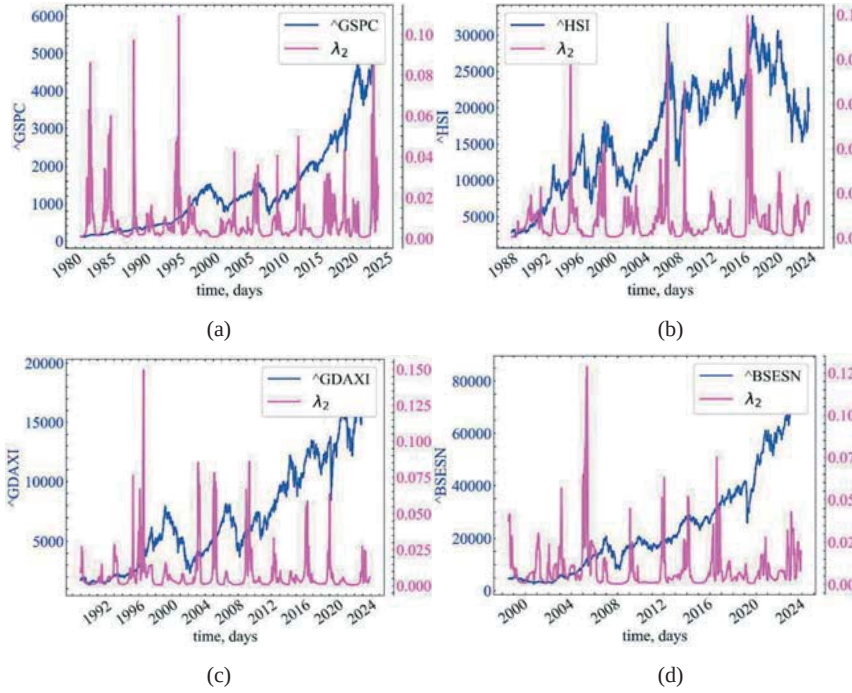


Fig. 6.2: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their algebraic connectivity

Fig. 6.2 shows that λ_2 increases in the pre-crisis periods, which indicates an increase in the degree of synchronization between market traders in these periods. The stock market network is becoming more and more correlated and stable. Such dynamics may indicate an increase in coherence between major market players regarding their further actions on stock market.

6.2.3.2 Graph energy

From the eigenvalues of the adjacency matrix A from G, one can determine a measure such as the **graph energy** $E(G)$ [68, 80]:

$$E = E(G) = \sum_{i=1}^{N} |\lambda_i|.$$

Similar to λ_2, we have a completely disconnected graph when $E(G) = 0$. For each $\lambda_i > 0$, there are many edges e_{ij} that determine the high and effective connectivity of G.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[1],
          ylabel,
          measure_labels[1],
          xlabel,
          file_names[1],
          clr="crimson")
```

In Fig. 6.3 is shown the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their graph energy.

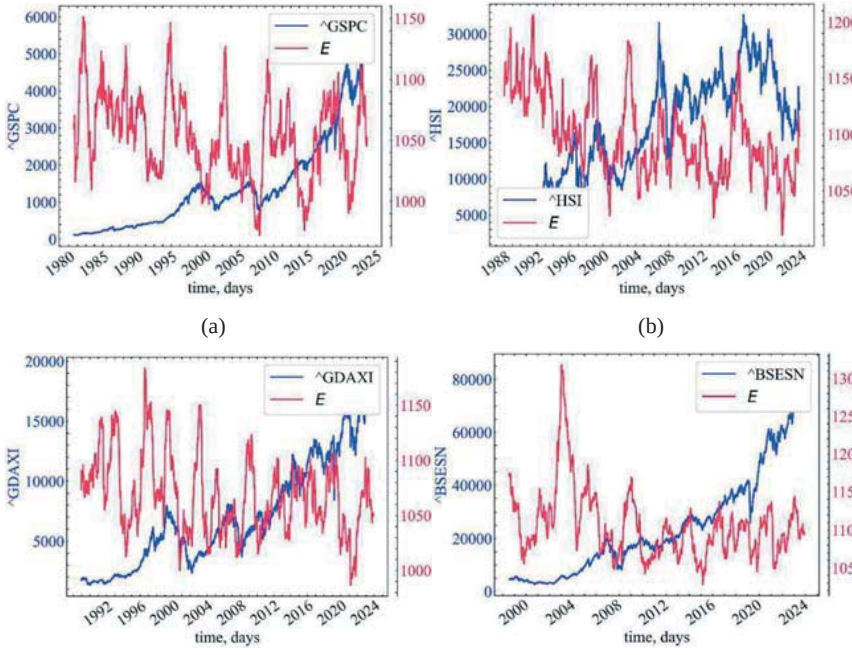


Fig. 6.3: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their graph energy

Fig. 6.3 shows that during periods of relative stability, E remains at a rather low level, indicating that market traders are disconnected during such periods. Both buyers and sellers act rather uncorrelated. In pre-crisis periods, energy begins to increase, indicating an increase in the efficiency of work between market players and their connectivity.

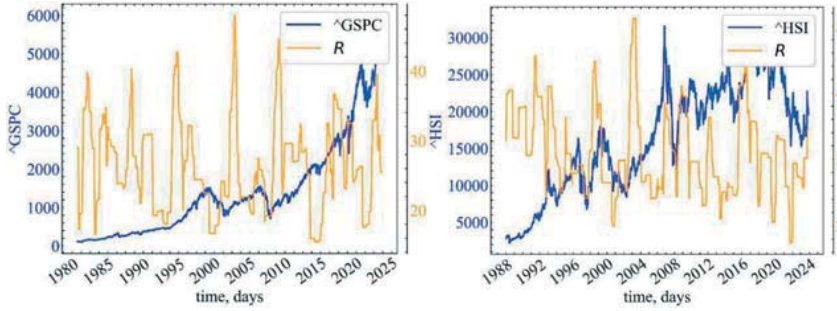
6.2.3.3 Spectral radius

In addition to the above measures, you can define such measures as the **spectral radius**, which is the largest absolute eigenvalue of the matrix A:

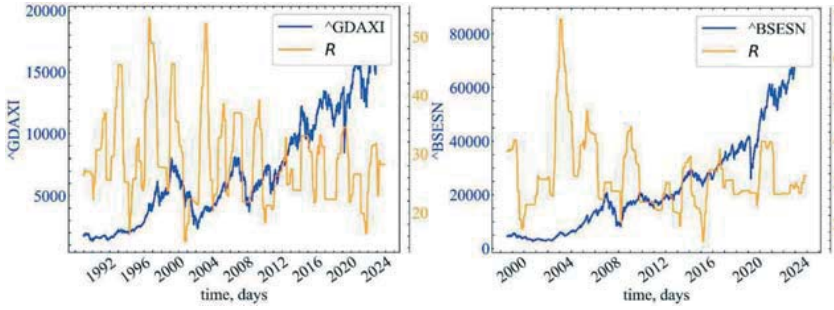
$$R = R(G) = \max_{1 \le i \le N} |\lambda_i|.$$

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[2],
          ylabel,
          measure_labels[2],
          xlabel,
          file_names[2],
          clr="orange")
```

In Fig. 6.4 is shown the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their spectral radius.



(a) (b)



(c) (d)

Fig. 6.4: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their spectral radius

Fig. 6.4 shows that the spectral radius increases during crisis and pre-crisis periods, indicating that the correlation of the stock market graph is growing and that traders are synchronizing their actions.

6.2.3.4 Spectral gap

By ranking the eigenvalues of the adjacency matrix G in non-decreasing order, i.e., $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$, we can define a measure called the **spectral gap**:

$$\delta = \delta(G) = \lambda_n - \lambda_{n-1}$$

for which for which λ_n is the first largest eigenvalue of $\hat{L}$ and λ_{n-1} is the second largest eigenvalue. The spectral gap shows the synchronization rate in the studied network. The larger it is, the more interconnected the nodes are and the more complex the graph is.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[3],
          ylabel,
          measure_labels[3],
          xlabel,
          file_names[3],
          clr="darkgreen")
```

In Fig. 6.5 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their spectral gap.

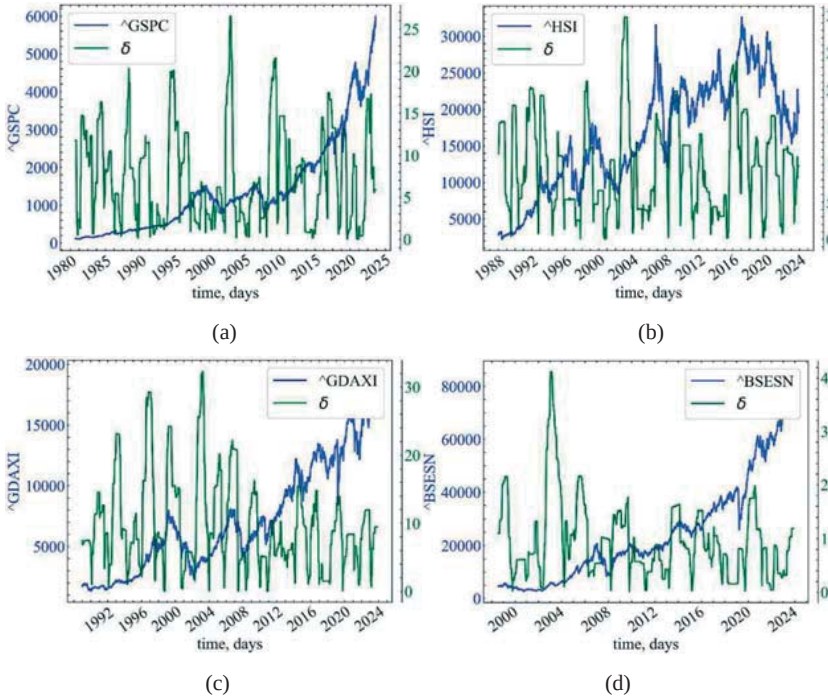


Fig. 6.5: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their spectral gap

Fig. 6.5 demonstrates that the spectral gap is also an indicator of market synchronization in pre-crisis periods. However, the dynamics of this indicator suggests that in times of crisis, the largest eigenvalue of the Laplace matrix begins to carry the most information. It can be assumed that the second and third can also serve as indicators of crash events, but the largest eigenvalue in this case seems to be the best solution.

6.2.3.5 Spectral moment

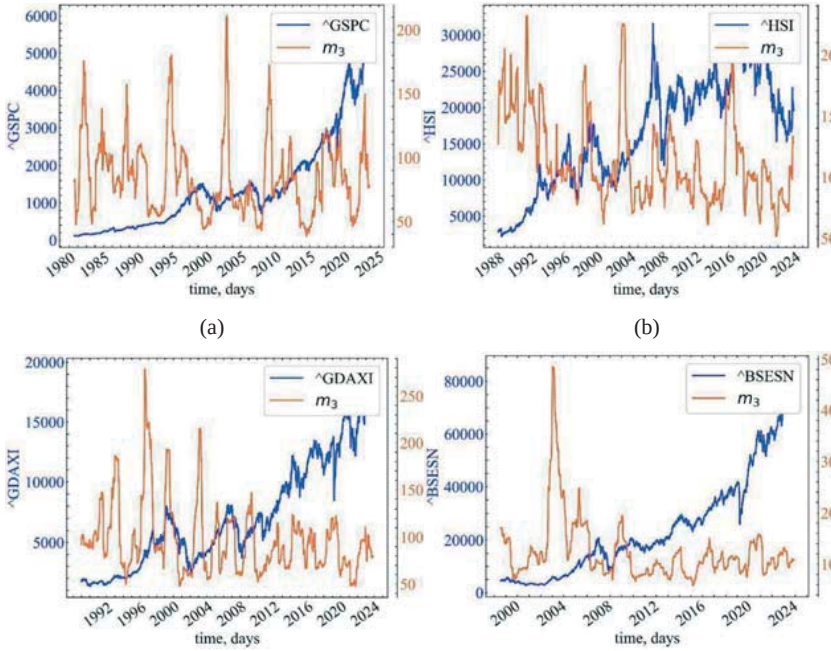
A spectral measure of complexity that we would also like to introduce is the ***k*-th spectral moment**. For a nonnegative integer k, the k -th spectral moment is defined as

$$m_k = m_k(G) = \sum_{i=1}^{N} \lambda_i^k,$$

where m_k is the number of closed loops of length k [52]. The number of closed traversals is an important indicator for measuring the complexity of a system. As shown in the work of Wu et al. [90], using the number of closed traversals of the entire length, we can measure the complexity of the graph and the redundancy of alternative shortest paths. Thus, higher values of m_k correspond to higher network complexity. For further calculations, we chose $k = 3$.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[4],
          ylabel,
          measure_labels[4],
          xlabel,
          file_names[4],
          clr="chocolate")
```

In Fig. 6.6 is illustrated the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their spectral momentum indicator.



(c) (d)

Fig. 6.6: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their spectral momentum indicator

The dynamics of m^3 shows that the most significant degree of synchronization was characterized by the pre-crisis dynamics. During these periods, we had the largest number of fairly high eigenvalues of the Laplace matrix, and thus a fairly high degree of market synchronization in these periods.

6.2.3.6 Spectral natural connectivity

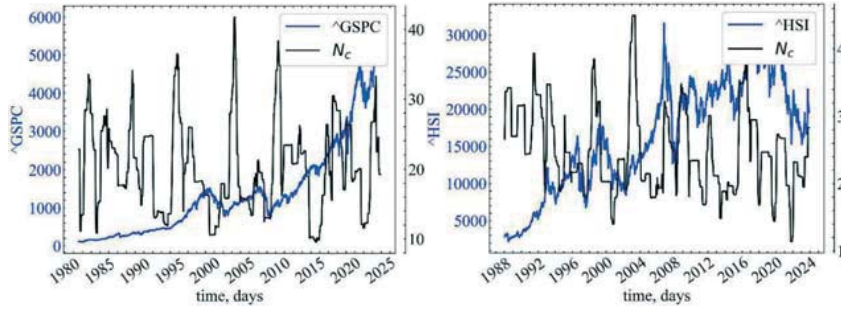
Yun et al. [172] proposed to measure the "average eigenvalue" of the adjacency spectrum of a graph G. It was proposed to call this indicator the **natural connectivity** or **natural eigenvalue**:

$$N_c = N_c(G) = \ln\left(\frac{1}{N}\sum_{i=1}^{N}\exp\lambda_i\right).$$

Estrada [59], Wu et al. [90] have shown that N_c is a sensitive and reliable measure of network resilience.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[5],
          ylabel,
          measure_labels[5],
          xlabel,
          file_names[5],
          clr="black")
```

In Fig. 6.7 is illustrated the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their spectral natural connectivity indicator.



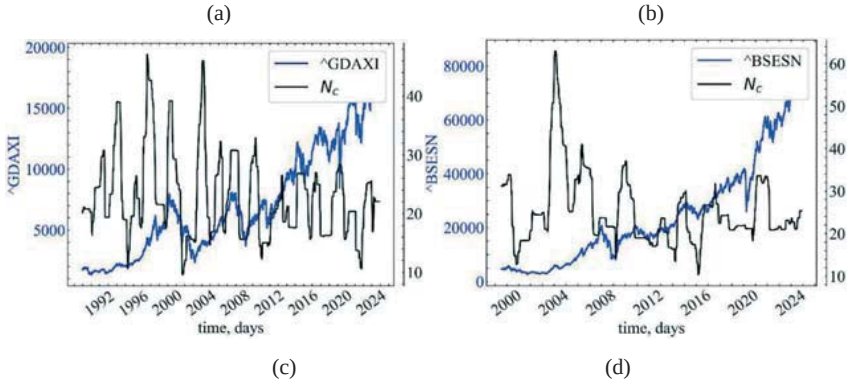


Fig. 6.7: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their spectral natural connectivity

Fig. 6.7 shows that the natural connectivity index increases in pre-crisis periods. That is, this indicator can be used as an indicator or precursor of crash events in the stock market. Particularly characteristic is the increase in the degree of market synchronization on the eve of 1997 or 2021, which may indicate the initial stages of strengthening the stability of the stock market network.

6.2.4 Topological measures of centrality

There are many ways to quantify the importance of a vertex or an edge in terms of a particular network attribute, thus reflecting the **topology** of a complex network.

```
DegreeMax = []
GlobalEigenvectorCentrality = []
GlobalClosenessCentrality = []
GlobalInformationCentrality = []
GlobalBetweennessCentrality = []
GlobalHarmonicCentrality = []

for i in tqdm(range(0,length-window,tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

    if graph_type == 'classic':
        g = NaturalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()
```

```
    if graph_type == 'horizontal':
        g = HorizontalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()

# maximum vertex degree
    deg_max = max(dict(nxg.degree()).values())

# global eigenvector centrality
    glob_eigenvector_centrality = np.mean(list(nx.eigenvector_centrality_nump
y(nxg).values()))

# global closeness centrality
    glob_closeness_centrality = np.mean(list(nx.closeness_centrality(nxg).val
ues()))

# global information centrality
    glob_information_centrality = np.mean(list(nx.information_centrality(nxg)
.values()))

# global betweenness centrality
    glob_betweenness_centrality = np.max(list(nx.betweenness_centrality(nxg).
values()))

# global harmonic centrality
    glob_harm_centrality = np.mean(list(nx.harmonic_centrality(nxg).values())
)

    DegreeMax.append(deg_max)
    GlobalEigenvectorCentrality.append(glob_eigenvector_centrality)
    GlobalClosenessCentrality.append(glob_closeness_centrality)
    GlobalInformationCentrality.append(glob_information_centrality)
    GlobalBetweennessCentrality.append(glob_betweenness_centrality)
    GlobalHarmonicCentrality.append(glob_harm_centrality)
```

Save the initial values to a text document. We also prepare labels for the figures and titles for the saved ones:

```
ind_names = ['DegreeMax', 'GlobalEigenvectorCentrality', 'GlobalClosenessCent
rality',
'GlobalInformationCentrality', 'GlobalBetweennessCentrality', 'GlobalHarmonic
Centrality']

indicators = [DegreeMax, GlobalEigenvectorCentrality, GlobalClosenessCentrali
ty,
              GlobalInformationCentrality, GlobalBetweennessCentrality, Globa
lHarmonicCentrality]

measure_labels = [r'$D_{max}$', r'$X$', r'$C$', r'$I$', r'$B$', r'$GHc$']

file_names = []

for i in range(len(ind_names)):
```

```
    name = f"{ind_names[i]}_symbol={symbol}_wind={window}_step={tstep}_series
type={ret_type}_graph_type={graph_type}"
    np.savetxt(name +".txt", indicators[i])
    file_names.append(name)
```

6.2.4.1 Maximum vertex degree

The **node degree** or **degree centrality** is conceptually the simplest metric for describing the connectivity characteristics of a single node in a complex network. It can be represented as

$$d_i = \sum_{j=1}^{N} A_{ij},$$

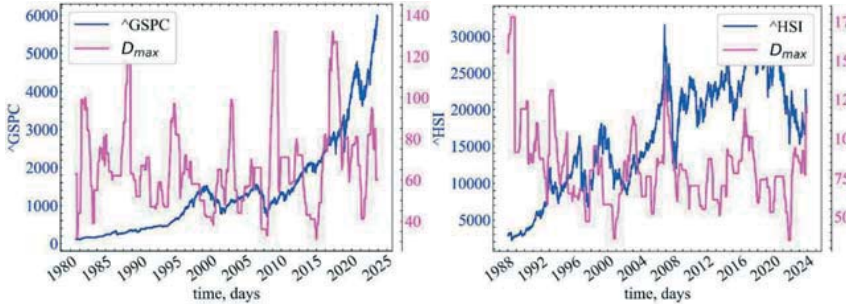
where d_i counts the number of j-th edges incident to vertex i.

In addition to the degree of a particular vertex, we can identify the vertex with the largest number of incident edges. We can denote the number of such vertices as D_{max}:

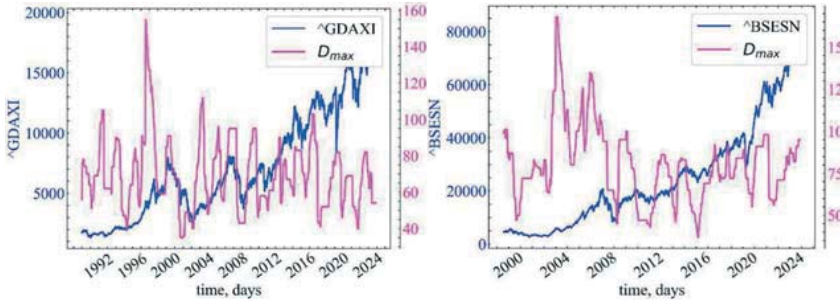
$$D_{max} = \max_{i=1,\dots,N} d_i.$$

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[0],
          ylabel,
          measure_labels[0],
          xlabel,
          file_names[0],
          clr="magenta")
```

Fig. 6.8 represents the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their maximum degree indicator.



(a) (b)



(c) (d)

Fig. 6.8: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their maximum vertex degree

Fig. 6.8 shows that the maximum degree of the peak begins to increase during crisis and pre-crisis periods, indicating that the centrality of one or more nodes is increasing. It can be assumed that one or more market traders begin to concentrate the attention of all other actors involved in the stock market.

6.2.4.2 Mean eigenvector centrality

The **eigenvector centrality** calculates the importance of a node by adding the influences of its neighbors. The centrality for node i is the i-th element of the eigenvector x associated with the eigenvalue λ of the maximum modulus, which is positive. Such an eigenvector x is determined to the nearest multiplicative constant by the equation

$$\lambda x^T = x^T A,$$

where A is the adjacency matrix of graph G. The above equation is equivalent to the following:

$$\lambda x^T = \sum_{j \to i} x_j,$$

That is, adding the eigenvector centralities of the predecessors of node i gives the degree of influence of i multiplied by λ. In the case of undirected graphs, x also solves the familiar equation $Ax = \lambda x$.

According to the Perron-Frobenius theorem [3], if G is strongly connected, then there exists a single eigenvector x, and all its elements are strictly positive.

If G is not highly connected, then there may be several left eigenvectors associated with λ, and some of their elements may be zero.

> **Note**
>
> The degree of influence or eigenvector centrality was introduced by Landau [81] for chess tournaments. Later, it was rediscovered by Wei [156] and then popularized by Kendall [108] in the context of sports rankings. Berge introduced a general definition for graphs based on social ties [34]. Bonacic [120] reintroduced eigenvector centrality and made it popular in linkage analysis.
>
> This function computes the left dominant eigenvector corresponding to the addition of the influence of predecessors: this is a common approach. To add the centrality of successors, first flip the graph with G.reverse().
>
> This implementation uses the SciPy sparse eigenvalue solver (ARPACK) to find the largest eigenvalue/eigenvector pair using Arnoldi iterations.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[1],
          ylabel,
          measure_labels[1],
          xlabel,
          file_names[1],
          clr="crimson")
```

Fig. 6.9 represents the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their global eigenvector centrality.

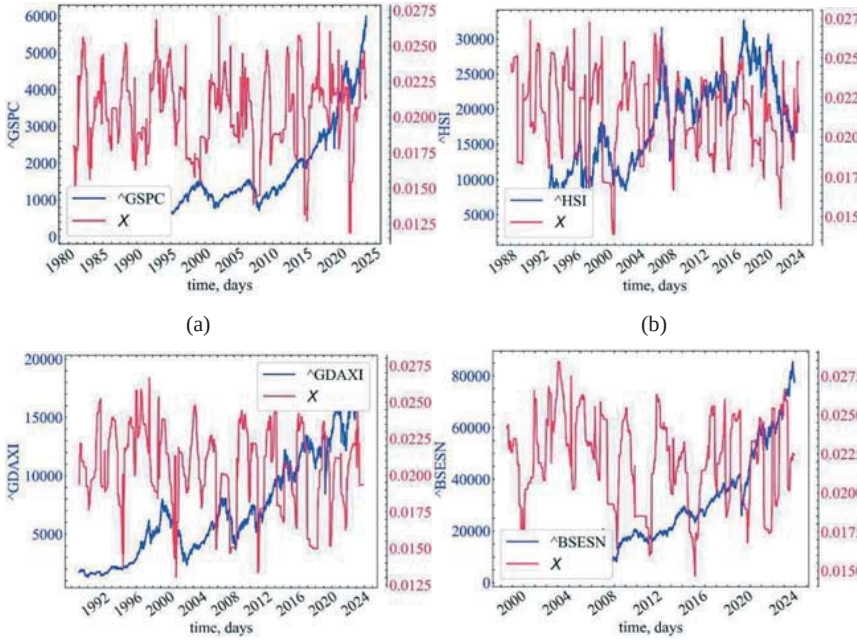


Fig. 6.9: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their global eigenvector centrality

6.2.4.3 Global closeness centrality

In a network, the distance l_{ij} between node i and node j denotes the number of edges that connect the shortest path between these two nodes. Based on the notion of the length of the shortest path between two nodes, we can provide various measures that characterize the connectivity of the entire network. One such measure is the **closeness centrality** between node i and all other nodes

$$c_i = (N-1)\Big/\left(\sum_{j=1}^{N} l_{ij}\right), \tag{6.1}$$

providing the inverse average of all shortest paths from i to all nodes j.

The arithmetic mean of the closeness degree for each i-th node gives us the **global (average) closeness centrality**:

$$C = \frac{1}{N}\sum_{i=1}^{N} c_i.$$

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[2],
          ylabel,
          measure_labels[2],
          xlabel,
          file_names[2],
          clr="orange")
```

Fig. 6.10 represents the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their global closeness centrality.

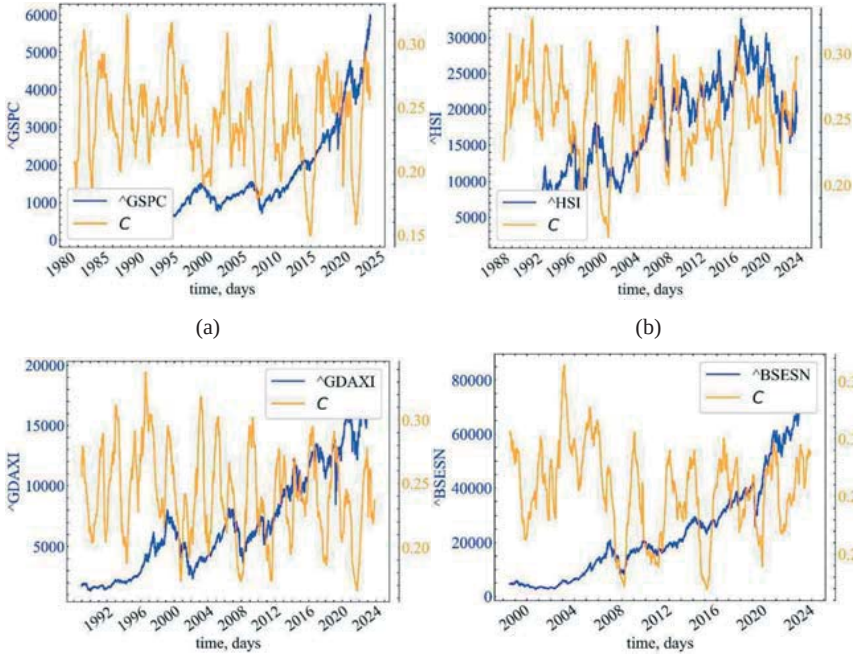


Fig. 6.10: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their global closeness centrality

As can be seen from the figure (Fig. 6.10), the global closeness centrality increases during crisis and pre-crisis periods, indicating a decline in the length of the shortest paths in the stock market market visibility graph. This indicates that

the degree of synchronization between traders increases before and during the crash phenomena.

6.2.4.4 Global information centrality

To determine the centrality of any node i, it is proposed to first determine its information connectivity with other nodes, i.e. $\{I_{ij}|j = 1,\dots,N\}$. The average harmonic value of information about the path from node i to other nodes will be used to determine the **degree of information centrality** of node i. In particular, if I_i is related to the centrality or information content of node i, then

$$\widehat{I_i} = \left(\frac{1}{N}\sum_{j=1}^{N}\frac{1}{I_{ij}}\right)^{-1}. \tag{6.2}$$

According to Stevenson and Zeleny [240], the degree of information can be calculated by inverting a simple matrix. First of all, we define the $N \times N$ matrix $B = \{b_{ij}\}$, where

$$\mathrm{b_{ij}} = \begin{cases} 0, & \text{if } i \text{ and } j \text{ are adjacent,} \\ 1, & \text{in other case,} \end{cases}$$

and $b_{ii} = 1 + d_i$, where d_i is the degree of vertex i.

Next, by defining the matrix $C = \{c_{ij}\} = B^{-1}$, we can calculate I_{ij} according to the following equation:

$$I_{ij} = \left(c_{ii} + c_{jj} - 2c_{ij}\right)^{-1}.$$

The element $\sum_{j=1}^{N} 1/I_{ij}$ in Eq. (6.2) can be rewritten as follows:

$$\sum_{j=1}^{N} c_{ii} + c_{jj} - 2c_{ij} = Nc_{ii} + T - 2R,$$

where $T = \sum_{j=1}^{N} c_{jj}$ and $R = \sum_{j=1}^{N} c_{ij}$.

Therefore, information centrality of node i can be presented as

$$I_i = [(Nc_{ii} + T - 2R)/N]^{-1} = [c_{ii} + (T - 2R)/N]^{-1}.$$

Similarly, to measure the **global information centrality**, we consider the arithmetic mean of the local information centrality:

$$\hat{I} = \frac{1}{N}\sum_{i=1}^{N} I_i.$$

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[3],
          ylabel,
          measure_labels[3],
          xlabel,
          file_names[3],
          clr="darkgreen")
```

Fig. 6.11 shows the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their global information centrality.

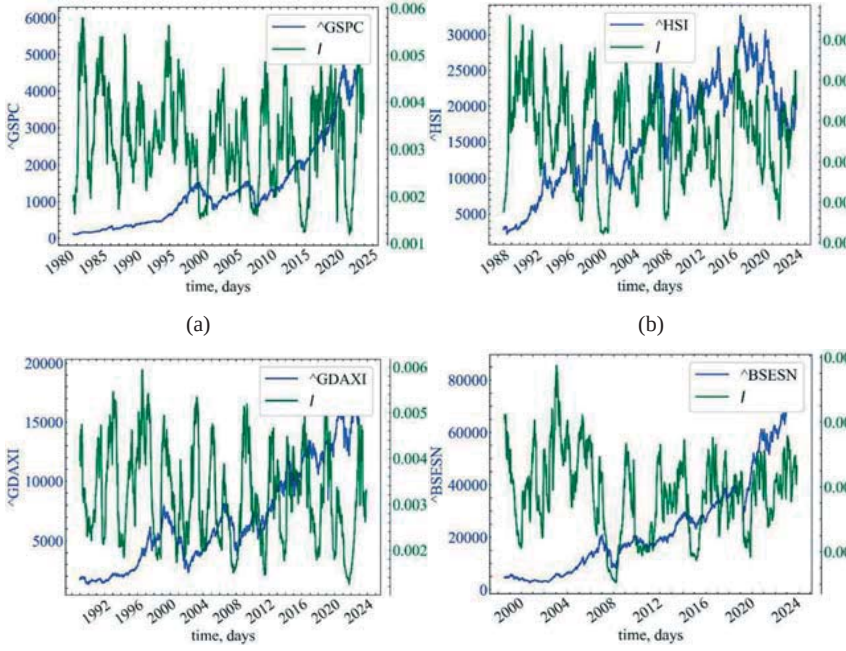


Fig. 6.11: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their global information centrality

Fig. 6.11 shows that the global information centrality is increasing in the pre-crisis periods, which is an indicator of the growing efficiency of information transfer between market traders and the growing determinism of market dynamics.

6.2.4.5 Maximum betweenness centrality

Another commonly studied path-based characteristic of nodes is the **betweenness centrality**, which measures the fraction of all shortest paths in the network that go from i to j through node k. For the total number of shortest paths between nodes i and j, denoted as $\sigma(i,j)$, and the shortest paths passing through a given node $k\big(\sigma(i,j|k)\big)$, the degree of intermediation can be defined as

$$b_k = \sum_{i,j=1;i,j\neq k}^{N} \sigma\,(i,j|k)/\sigma(i,j).$$

To find the largest amount of information passing through a particular k-th node, we measure the **maximum betweenness centrality** by considering each k-th node:

$$B = \max_{i=1,\ldots,N} b_i.$$

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[4],
          ylabel,
          measure_labels[4],
          xlabel,
          file_names[4],
          clr="chocolate")
```

Fig. 6.12 demonstrates the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their maximum betweenness centrality.

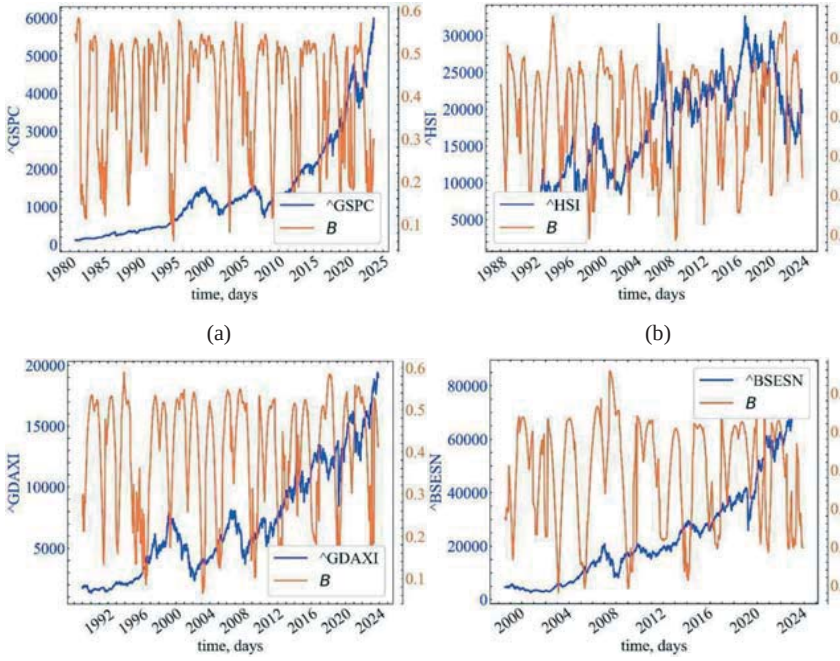


Fig. 6.12: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their maximum betweenness centrality

As you can see, the maximum betweenness centrality decreases in the pre-crisis periods, which indicates a decline in the number of intermediaries through which information about the future dynamics of the studied stock indices can pass. This suggests that there are one or more traders in the market who are the focus of almost all other traders, and all traders can be connected to the most influential ones through one or more shortest paths.

6.2.4.6 Global harmonic centrality

Marchiori and Latora [161] proposed a measure similar to (6.1), called the **harmonic centrality**. For a given node j, it can be defined as

$$Hc_j = \sum_{i=1, i\neq j}^{N} \left(l_{ij}\right)^{-1},$$

where $\left(l_{ij}\right)^{-1} = 0$ if there is no path between nodes i and j. The **global harmonic centrality** is determined by the arithmetic mean of local harmonic centralites.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[5],
          ylabel,
          measure_labels[5],
          xlabel,
          file_names[5],
          clr="black")
```

In Fig. 6.13 is shown the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their global harmonic centrality.

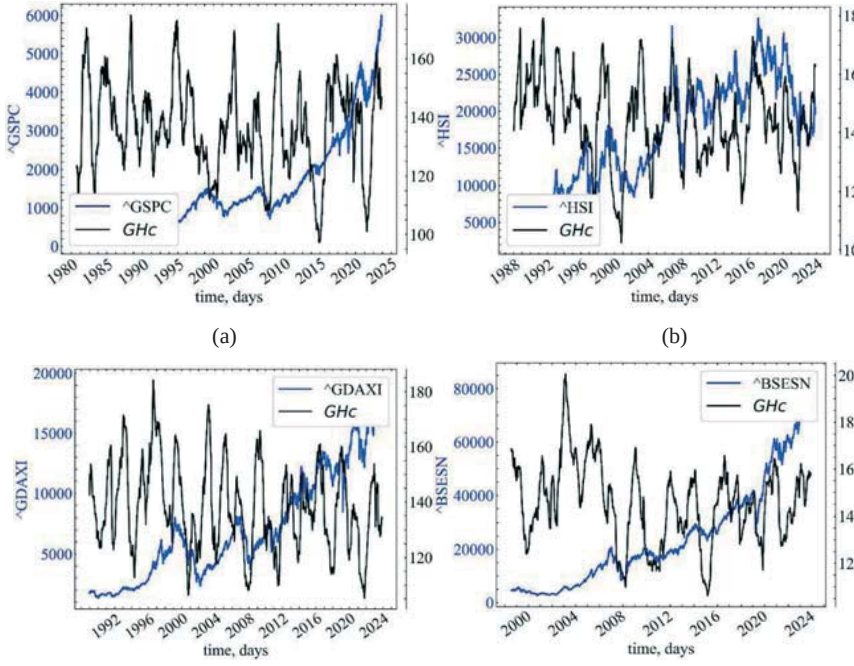


Fig. 6.13: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their global harmonic centrality

6.2.4.7 Assortativity

Assortativity refers to the tendency of a network to connect nodes with similar properties, while disassortativity is manifested in the connection of nodes with dissimilar properties. Real-world networks can exhibit different levels of assortativity. Social networks, such as interactions between scientists or corporate directors, usually have positive assortativity. On the other hand, technological and biological networks, such as power grids, the Internet, protein interactions, neural networks, and food webs, usually exhibit negative assortativity.

In the following, we will present several indicators of assortativity for the early identification of stock market crises.

```
Assortativity = []
AvgDegreeConnectivity = []

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

    if graph_type == 'classic':
        g = NaturalVG(directed='left_to_right').build(fragm)
        pos = g.node_positions()
        nxg_dir = g.as_networkx()
     if graph_type == 'horizontal':
        g = HorizontalVG(directed='left_to_right').build(fragm)
        pos = g.node_positions()
        nxg_dir = g.as_networkx()

# calculation of assortativity
    assort = nx.degree_pearson_correlation_coefficient(nxg_dir)

# average degree connectivity
    avg_deg_con = np.mean(list(nx.average_degree_connectivity(nxg_dir, source
="in", target="in").values()))

    Assortativity.append(assort)
    AvgDegreeConnectivity.append(avg_deg_con)

ind_names = ['Assortativity', 'AvgDegreeConnectivity']

indicators = [Assortativity, AvgDegreeConnectivity]

measure_labels = [r'$r$', r'$\langle d_{nn}^{w} \rangle$']

file_names = []
```

```
for i in range(len(ind_names)):
    name = f"{ind_names[i]}_symbol={symbol}_wind={window}_step={tstep}_series
type={ret_type}_graph_type={graph_type}"
    np.savetxt(name +".txt", indicators[i])
    file_names.append(name)
```

6.2.4.7.1 Average degree connectivity

The **average degree connectivity** $d_{nn}(d)$ for nodes with degree d is another measure used to study the structure of networks [139]. Since it can be expressed as $d_{nn}(d) = \sum_{d'} P(d'|d)$, where $P(d'|d)$ is the conditional probability that a given vertex with degree d is connected to a vertex with degree d'. This value expresses the correlation between the degrees of connected vertices [150]. In the absence of correlations between degrees, $P(d'|d)$ does not depend on d, nor on the average degree of its nearest neighbors, i.e., $d_{nn}(d) = \text{const}$ [139]. In the presence of correlations, the behavior of $d_{nn}(d)$ defines two general classes of networks. If $d_{nn}(d)$ is an increasing function of d, then nodes with a high (low) degree are more likely to be connected to nodes with a higher (lower) degree. This property is called **assortative mixing** in various fields of science [107]. On the contrary, the descending behavior of $d_{nn}(d)$ defines **disassortative mixing**, in the sense that nodes with high (low) degree have most neighbors with low (high) degree.

The measure of such assortativity or disassortativity for the neighbors of a certain vertex i can be defined as the average degree connectivity (weighted average degree of the nearest neighbor):

$$d_i^w = \frac{1}{s_i} \sum_{j=1}^{N} A_{ij}\, w_{ij} d_j,$$

where $s_i = \sum_{j=1}^{N} A_{ij}\, w_{ij}$ is the "strength" of the i-th node; A_{ij} is an element of the adjacency matrix A; w_{ij} is the weight of the edge e_{ij} (in our case, it is equal to 1); d_j represents the vertex degree of the j-th neighbor.

In general, this equation measures the degree of attraction of neighbors with high or low vertex degree to each other relative to the magnitude of actual interactions.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[1],
          ylabel,
          measure_labels[1],
          xlabel,
          file_names[1],
          clr="darkorange")
```

Fig. 6.14 represents the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their average degree connectivity.

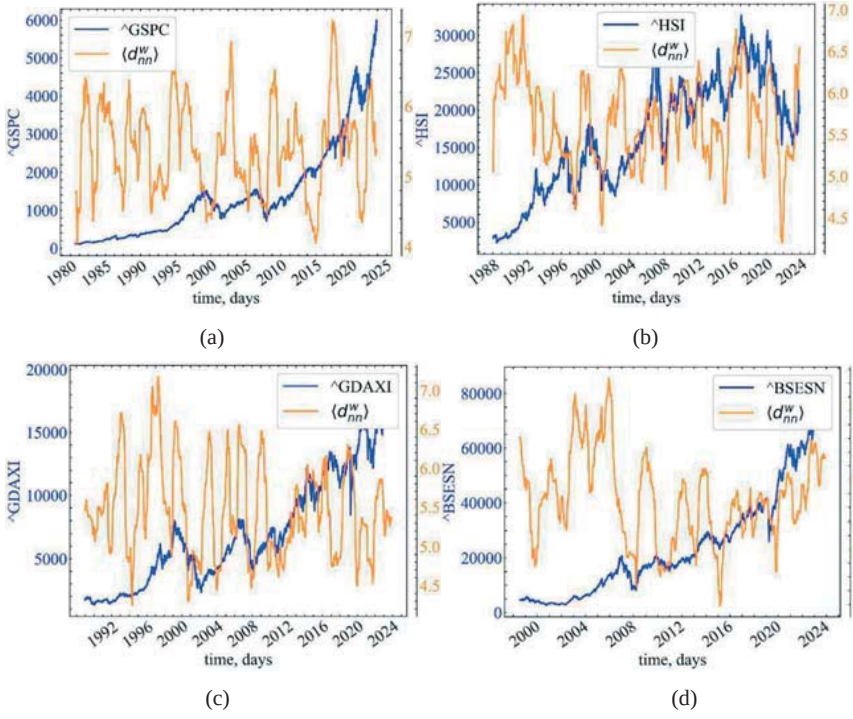


Fig. 6.14: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their average degree connectivity

As can be seen from this figure (Fig. 6.14), the average degree connectivity increases in the pre-crisis periods, indicating a gradual increase in the degree of attraction of nodes with high degree centrality to nodes with even higher centrality.

6.2.4.7.2 Degree of assortativity

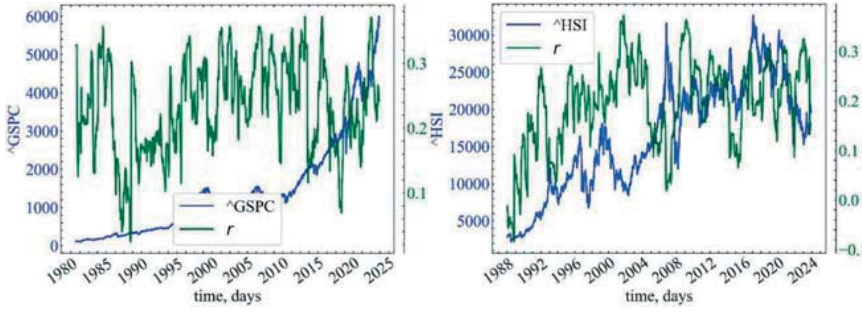
Another form of assortative mixing depends on one or more scalar properties of the network vertices. To calculate it, we define a matrix e_{ij} that satisfies the addition rules: $\sum_{ij} e_{ij} = 1$, $\sum_j e_{ij} = a_i, = a_i$, $\sum_i e_{ij} = b_j$, where a_i and b_j are the shares of edges that start and end at vertices i and j. By calculating Pearson's correlation coefficient, one can determine the degree of assortativity [107]. Thus, this assortativity coefficient is calculated as

$$r = \sum_{xy} xy\left(e_{xy} - a_x b_y\right)/\sigma_a \sigma_b,$$

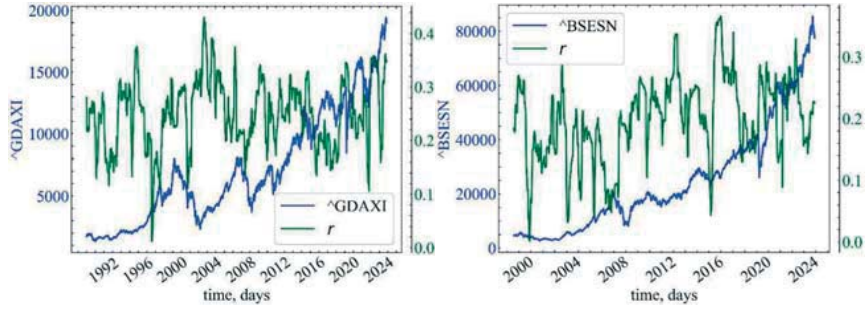
and σ_a and σ_b define the standard deviations of the distributions a_x and b_y; $-1 \le r \le 1$, where $r < 0$ indicates higher disassortativity, $r < 0$ demonstrates higher assortativity, and $r = 0$ indicates no assortativity between nodes.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[0],
          ylabel,
          measure_labels[0],
          xlabel,
          file_names[0],
          clr="darkgreen")
```

Fig. 6.15 represents the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their average degree of assortativity.



(a) (b)



(c) (d)

Fig. 6.15: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their average degree of assortativity

Fig. 6.15 shows that the assortativity coefficient decreases in the pre-crisis periods, indicating disassortative market behavior at these points in time: nodes with a low degree of connectivity and centralization gravitate toward nodes characterized by a high degree of mediation, harmony, information, proximity, etc. As already mentioned, the disassortment inherent in the pre-crisis periods of the stock indices is also characteristic of both real social and complex biological networks.

6.2.4.8 Clustering

In graph theory, the clustering coefficient indicates the degree to which nodes in a graph tend to cluster. Studies show that in most real-world networks, including social media, nodes usually form compact groups with a high number of connections between them.

For further analysis, let's consider the indicators of transitivity, global triadic and quadratic clustering.

```
Transitivity = []
AvgClustering = []
AvgSquareClustering = []

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)
```

```
    if graph_type == 'classic':
        g = NaturalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()
    if graph_type == 'horizontal':
        g = HorizontalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()

# transitivity
    trans = nx.transitivity(nxg)

# global clustering coefficient
    avg_clust = nx.average_clustering(nxg)

# global square clustering coefficient
    avg_sqr_clust = np.mean(list(nx.square_clustering(nxg).values()))

    Transitivity.append(trans)
    AvgClustering.append(avg_clust)
    AvgSquareClustering.append(avg_sqr_clust)

ind_names = ['AvgClustering', 'Transitivity', 'AvgSquareClustering']

indicators = [AvgClustering, Transitivity, AvgSquareClustering]

measure_labels = [r'$\langle C_3 \rangle$', r'$T$', r'$\langle C_4 \rangle$']

file_names = []

for i in range(len(ind_names)):
    name = f"{ind_names[i]}_symbol={symbol}_wind={window}_step={tstep}_series
type={ret_type}_graph_type={graph_type}"
    np.savetxt(name +".txt", indicators[i])
    file_names.append(name)
```

6.2.4.8.1 Global clustering coefficient

In order to characterize the density of connections between the neighbors of vertex i, we can use the **local clustering coefficient**:

$$C_i^3 = \sum_{k,j=1}^{N} A_{ik}A_{kj}A_{ji} \Big/ d_i(d_i - 1),$$

where the numerator denotes the number of closed triangles containing vertex i.

We can consider the **global clustering coefficient** as the arithmetic mean of the local triangle clustering coefficient [51]:

$$\langle C^3 \rangle = \frac{1}{N} \sum_{i=1}^{N} C_i^3, \tag{6.3}$$

which measures the average tendency of the system to form triangular clusters.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[0],
          ylabel,
          measure_labels[0],
          xlabel,
          file_names[0],
          clr="magenta")
```

Fig. 6.16 shows the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their global clustering coefficient.

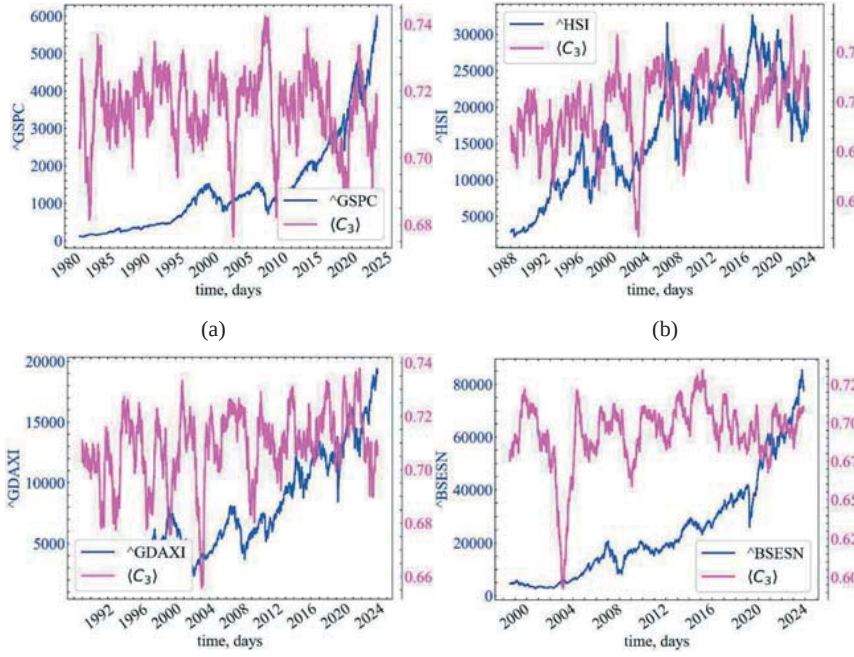


Fig. 6.16: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their global clustering coefficient

Fig. 6.16 shows that in absolute terms, the global triad clustering coefficient remains at a fairly high level, which indicates a fairly high degree of clustering of stock market traders. Locally, in the pre-crisis periods, it can be seen that $\langle C^3 \rangle$ decreases, which indicates the localized destruction of clustered groups of traders and their growing attraction to one or more market players.

6.2.4.8.2 Transitivity

In the case of very heterogeneous degrees, i.e., scale-free networks where only a few nodes have high degrees and the rest have low degrees ($d_i < 2$), the nodes with low degrees will participate mainly in the calculation of the local clustering coefficient, which can lead to underestimation of triangular clusters in the network. Barrat and Weigt [2] proposed an alternative approach to overcome this problem, called **transitivity** [142]:

$$T = \sum_{k,j=1}^{N} A_{ik} A_{kj} A_{ji} \Big/ \sum_{i,k,j=1}^{N} A_{ik} A_{ji}.$$

In real networks, we may encounter cases where connected neighbors in the network can form different cliques (forms of clustering). The classical local clustering coefficient, which measures the probability of finding triangles, usually corresponds to one-way networks. However, it cannot be formed in bipartite networks [126, 127]. The complex structures of one-way, two-way, and multi-way networks in a real-world system can lead to the formation of clusters of a much higher order.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[1],
          ylabel,
          measure_labels[1],
          xlabel,
          file_names[1],
          clr="crimson")
```

Fig. 6.17 illustrates the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their transitivity.

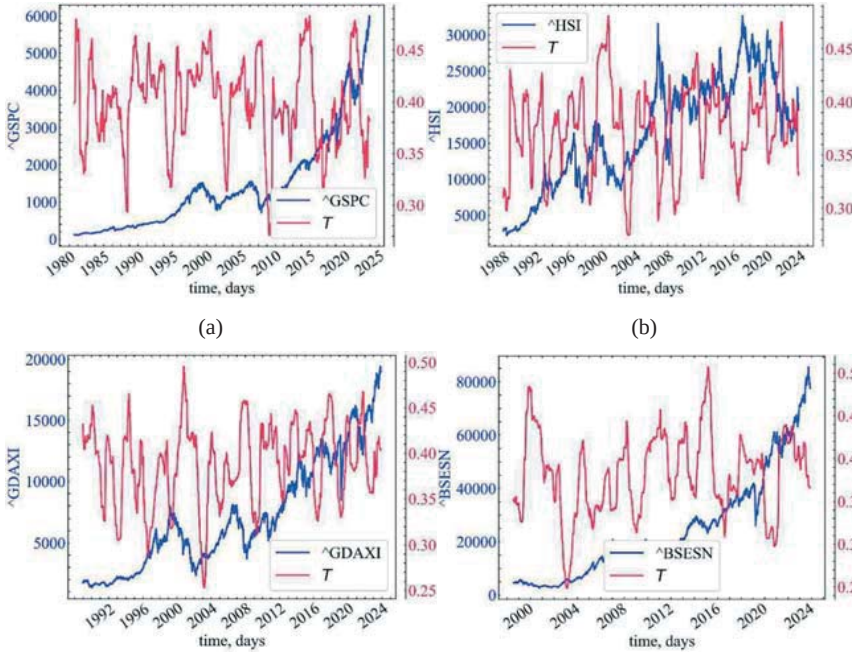


Fig. 6.17: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their transitivity

The transitivity indicator works in a similar way to $\langle C^3 \rangle$. However, unlike $\langle C^3 \rangle$, it provides much more signals of further crash behavior in the stock market. It can be seen that the market retains a fairly high share of triangular clicks that become incomplete in pre-crisis periods, as indicated by the decline in T.

6.2.4.8.3 Square clustering coefficient

Similar to C_i^3, which is the classical local clustering coefficient, it has been proposed to quantify the clustering coefficient C_i^4 [121], which corresponds to the probability of finding a "square" cluster formed by the neighbors of node i. That is, that two neighbors of node i have a common neighbor other than i. For each node i, it can be calculated as

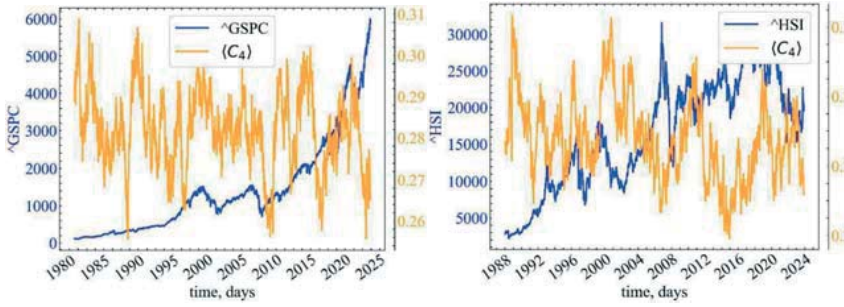
$$C_i^4 = \frac{\sum_{k=1}^{d_i}\sum_{j=k+1}^{d_i} q_i\,(k,j)}{\sum_{k=1}^{d_i}\sum_{j=k+1}^{d_i}[a_i(k,j)+q_i(k,j)]},$$

where $q_i(k,j)$ represents the number of observed quadratic clusters; $a_i(k,j) = \left(d_k - (1+q_i(k,j)+\theta_{ki})\right) + \left(d_j - \left(1+q_i(k,j)+\theta_{kj}\right)\right)$; $\theta_{kj} = 1$ if k and j are connected and 0 otherwise [130]. Similarly to (6.3), we can define the **global square clustering coefficient**:

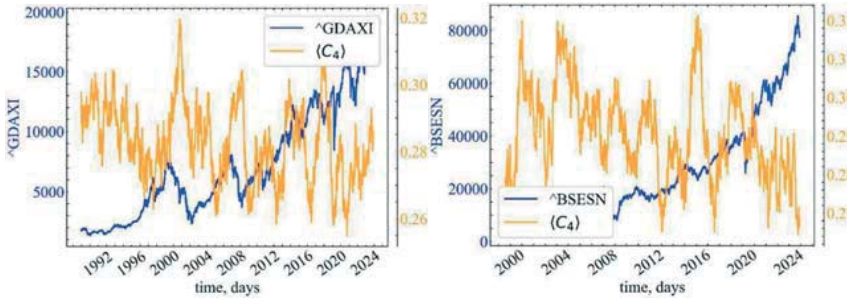
$$\langle C^4 \rangle = \frac{1}{N}\sum_{i=1}^{N} C_i^4.$$

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[2],
          ylabel,
          measure_labels[2],
          xlabel,
          file_names[2],
          clr="orange")
```

Fig. 6.18 represents the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their quadratic clustering coefficient.



(a) (b)



(c) (d)

Fig. 6.18: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their quadratic clustering coefficient

Fig. 6.18 shows that globally, stock indices contain a much smaller share of quadratic clusters compared to triadic clusters. Locally, we observe similar dynamics to the previous indicators: $\langle C_4 \rangle$ decreases in the pre-crisis period and gradually increases during the crisis and post-crisis periods. We can make the same assumption as before: in pre-crisis periods, traders begin to gradually isolate themselves from each other's analytics and focus their attention on the actions of one or more of the most influential groups. Although their clustering decreases, their actions remain coordinated according to the information they receive from the outside.

6.2.4.9 Connectivity

In mathematics, a **connected graph** is a graph whose number of edges approaches the maximum possible number (when each pair of vertices is connected by a single edge). Conversely, a **sparse graph** contains only a small number of edges. The exact definition of which graph is considered connected or sparse is ambiguous. Thus, the definition of graph density may vary depending on the context of the problem.

```
Density = []
for i in tqdm(range(0, length-window, tstep)):
    fragm = time_ser.iloc[i:i+window].copy()
```

```
    fragm = transformation(fragm, ret_type)

    if graph_type == 'classic':
        g = NaturalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()
    if graph_type == 'horizontal':
        g = HorizontalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()

# calculate density
    dens = nx.density(nxg)

    Density.append(dens)

ind_names = ['Density']

indicators = [Density]

measure_labels = [r'$\rho$']

file_names = []

for i in range(len(ind_names)):
    name = f"{ind_names[i]}_symbol={symbol}_wind={window}_step={tstep}_series
type={ret_type}_graph_type={graph_type}"
    np.savetxt(name +".txt", indicators[i])
    file_names.append(name)
```

6.2.4.9.1 Density

The **density** of a graph can help determine how densely populated the graph is with different edges. The higher it is, the greater the connectivity of the graph under study. It can be calculated as

$$\rho = E/E_{max},$$

where E is the number of edges in G, and $E_{max} = N(N-1)/2$ is the maximum number of edges in a simple undirected graph.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[0],
          ylabel,
          measure_labels[0],
          xlabel,
          file_names[0],
          clr="black")
```

In Fig. 6.19 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their density indicator.

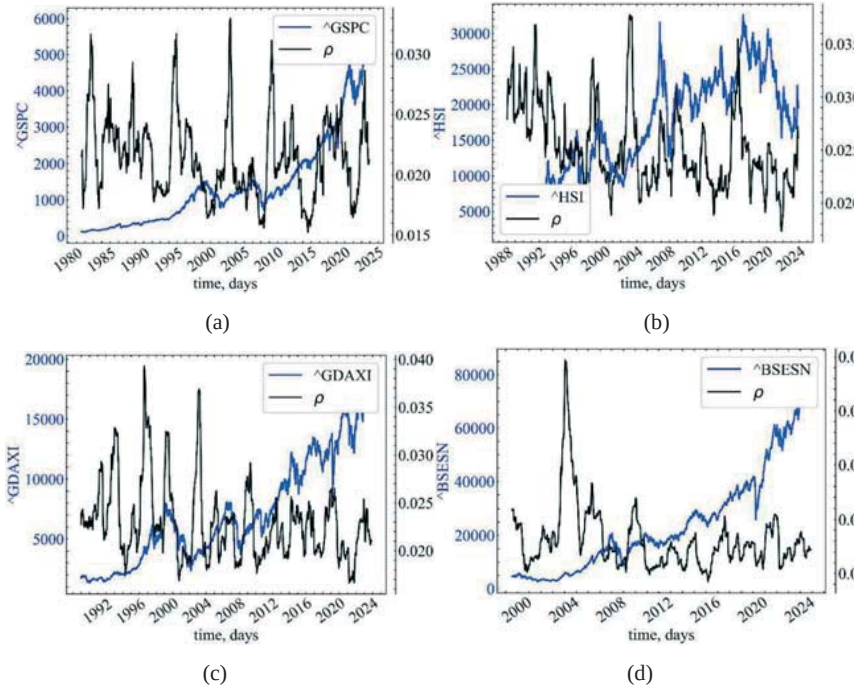


Fig. 6.19: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their density indicator

The figure shows that the global market connectivity remains quite low ($\rho < 0.10$), which indicates an insufficiently high level of connectivity between current and past nodes of price fluctuations in the stock market. The windowed dynamics of ρ indicates that at the pre-crisis point in time, the degree of density of market participants' connections increases, making the stock indices more stable.

6.2.4.10 Distance measures

Based on the length of a graph's shortest path, we can get many other indicators of its efficiency or the distance of its vertices from the center of connectivity of the graph under study.

```
Diameter = []
Radius = []

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

    if graph_type == 'classic':
        g = NaturalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()
    if graph_type == 'horizontal':
        g = HorizontalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()

# calculate diameter
    diameter = nx.diameter(nxg)

# radius
    rad = nx.radius(nxg)

    Diameter.append(diameter)
    Radius.append(rad)

ind_names = ['Diameter', 'Radius']

indicators = [Diameter, Radius]

measure_labels = [r'$diam$', r'rad']

file_names = []

for i in range(len(ind_names)):
    name = f"{ind_names[i]}_symbol={symbol}_wind={window}_step={tstep}_series
type={ret_type}_graph_type={graph_type}"
    np.savetxt(name +".txt", indicators[i])
    file_names.append(name)
```

6.2.4.10.1 Diameter

Note that the shortest path, which is a characteristic of the distance between the studied nodes i and j, can be used to characterize the overall size of the network. The value that determines the largest distance between vertex i and any other vertex is called **eccentricity**:

$$\varepsilon(i) = \max_j l_{ij}.$$

The size of the network can be characterized in terms of **diameter** and is defined as

$$diam = \max_i \varepsilon(i) = \max_i \max_j l_{ij}.$$

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[0],
          ylabel,
          measure_labels[0],
          xlabel,
          file_names[0],
          clr="magenta")
```

In Fig. 6.20 is presented the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their network diameter indicator.

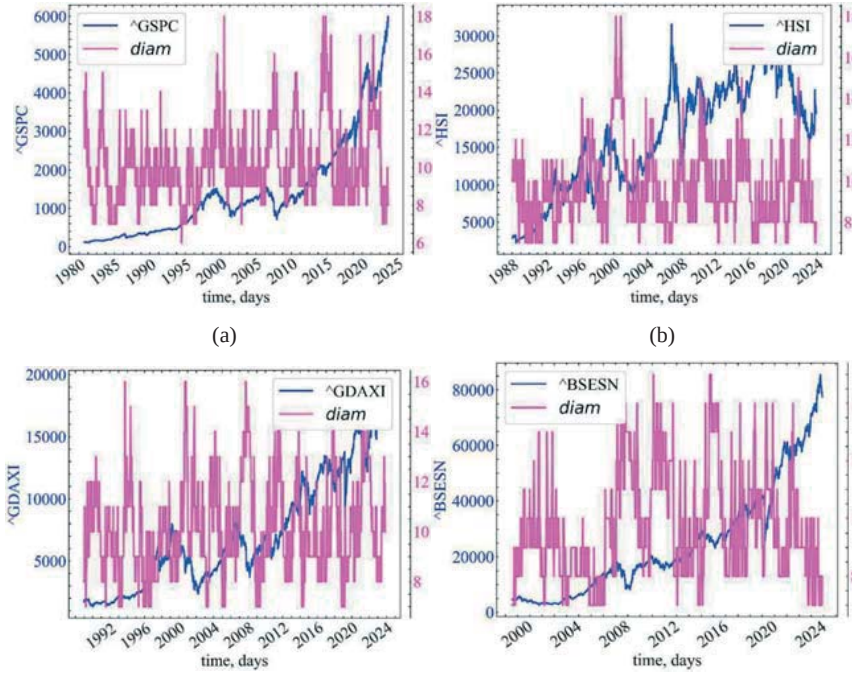


Fig. 6.20: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their network diameter

Fig. 6.20 shows that the diameter of the graph decreases in the pre-crisis period, which indicates that the upper boundary of the graph is approaching its center. This means that information passing from one trader to another in the stock market will take much fewer steps. In other words, in pre-crisis periods, traders rely less and less on intermediaries from various news resources and spend more time directly studying trading patterns in the stock market.

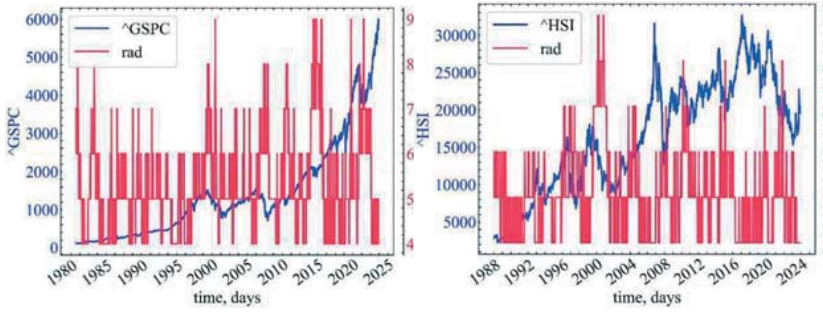
6.2.4.10.2 Radius

Thus, the diameter is the largest (maximum) path length in the network. Therefore, we can determine the smallest eccentricity of the network under study, which is called the **radius**:

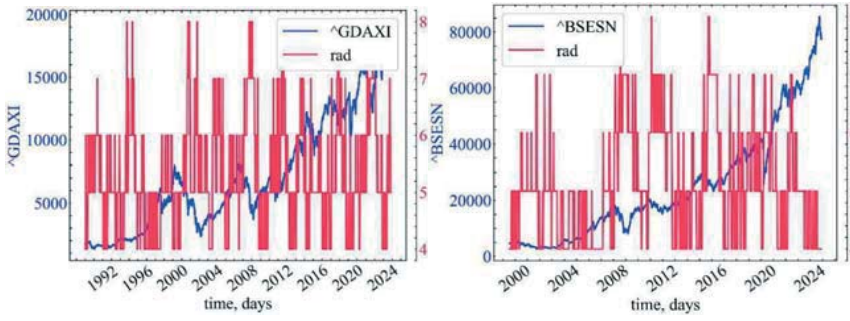
$$rad = \max_i \varepsilon(i) = \min_i \max_j l_{ij}.$$

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[1],
          ylabel,
          measure_labels[1],
          xlabel,
          file_names[1],
          clr="crimson")
```

Fig. 6.21 provides results on the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their network radius.



(a) (b)



(c) (d)

Fig. 6.21: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their network radius

Since the radius of the graph is the smallest eccentricity of the network, and the diameter is the largest, a similar conclusion can be drawn. If you look closely, you will notice that the radius represents about half the diameter, but the trend of these two indicators is identical.

6.2.4.11 Network efficiency

In the field of network science, **network efficiency**, also called communication efficiency, is a key metric. This concept is based on the assumption that the farther apart two nodes in a network are, the less efficient their communication becomes. Efficiency can be analyzed at both the local and global levels of the network. At the global level, the overall exchange of information throughout the network is evaluated, where information flows in parallel. At the local level, the network's resilience to failures on a smaller scale is measured. In particular, the local efficiency of a node i reflects how efficiently its neighbors exchange information in its absence.

```
LocalEfficiency = []
GlobalEfficiency = []

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)
```

```
    if graph_type == 'classic':
        g = NaturalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()
    if graph_type == 'horizontal':
        g = HorizontalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()

# calculate local efficiency
    local_eff = nx.local_efficiency(nxg)

# calculate global efficiency
    glob_eff = nx.global_efficiency(nxg)

    LocalEfficiency.append(local_eff)
    GlobalEfficiency.append(glob_eff)

ind_names = ['LocalEfficiency', 'GlobalEfficiency']

indicators = [LocalEfficiency, GlobalEfficiency]

measure_labels = [r'$E_{loc}$', r'$E_{glob}$']

file_names = []

for i in range(len(ind_names)):
    name = f"{ind_names[i]}_symbol={symbol}_wind={window}_step={tstep}_series
type={ret_type}_graph_type={graph_type}"
    np.savetxt(name +".txt", indicators[i])
    file_names.append(name)
```

6.2.4.11.1 Global efficiency

The definition of small-world behavior according to [161] can be expressed in terms of the efficiency E of the network. The efficiency ε_{ij} between nodes i and j is defined as $1/l_{ij}$. When $l_{ij} = \infty$ and, consistently, when $1/l_{ij} = 0$, i and j are considered disconnected. According to the efficiency formalism, it can be quantified for both global and local scales G. Latorre and Marchiori emphasized that $1/L$ and C can be viewed as first approximations of **global** $(\boldsymbol{E_{glob}})$ **and local** $(\boldsymbol{E_{loc}})$ **efficiency**.

The **average (global) efficiency** of G can be defined as $E_{glob} = \sum_{i,j=1}(l_{ij})^{-1}/N(N-1)$.

For the most efficient graph, where information is disseminated most efficiently, E_{glob} takes on a maximum value, and otherwise it takes on a minimum value.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[1],
          ylabel,
          measure_labels[1],
          xlabel,
          file_names[1],
          clr="indigo")
```

Fig. 6.22 provides results on the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their global network efficiency indicator.

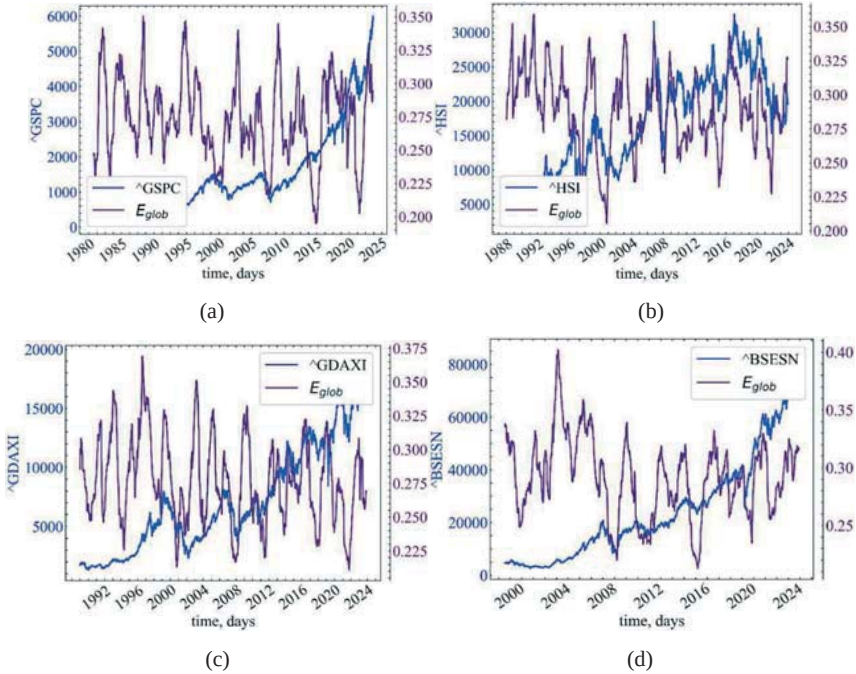


Fig. 6.22: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their global network efficiency

Fig. 6.22 shows that the degree of global network efficiency increases in the pre-crisis periods, indicating an increase in the degree of information flow in the

network. From the point of view of the visibility graph, stock indices begin to act in a more deterministic way, where the connectivity of its visibility graph becomes close to the topology of an ideal graph, where all information is transmitted in the most efficient way.

6.2.4.11.2 Local efficiency

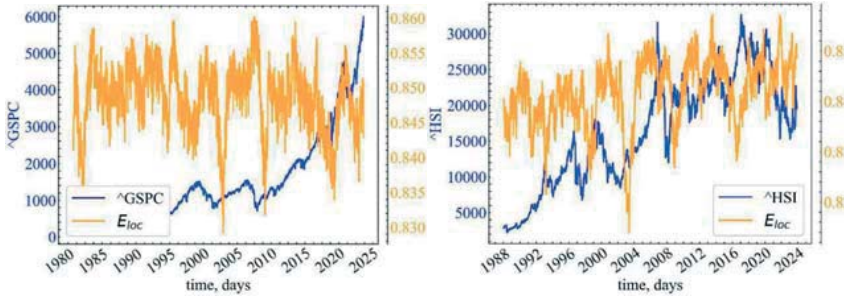
Local efficiency plays a role similar to the global clustering coefficient. Local efficiency E_{loc} can be quantified as

$$E_{loc} = \frac{1}{N}\sum_{i \in G_i} E_{glob}(G_i),$$

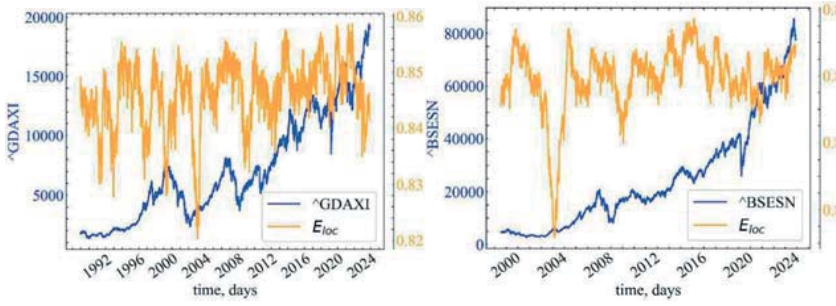
where G_i is a local subgraph of G, and $E_{glob}(G_i)$ characterizes the efficiency of this particular subgraph. Similar to the global clustering coefficient, E_{loc} determines how fault-tolerant the system under study is, i.e., how efficiently information is transported between the first neighbors of the i-th node when it is removed.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[0],
          ylabel,
          measure_labels[0],
          xlabel,
          file_names[0],
          clr="orange")
```

Fig. 6.23 illustrates the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their local network efficiency indicator.



(a) (b)



(c) (d)

Fig. 6.23: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their local network efficiency

Globally, $E_{loc} \approx 0.85$ for the stock market, which indicates the global resilience of the stock market network to possible attacks and exclusion of market traders from global trade. The windowed procedure shows that E_{loc} decreases in the pre-crisis periods, indicating a decline in local network efficiency. As already mentioned, since most attention is focused on one or more major market players, their potential disconnection from global trading could destabilize the entire stock market.

6.2.4.12 Shortest path

```
AvgPathLength = []

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()
    fragm = transformation(fragm, ret_type)

    if graph_type == 'classic':
        g = NaturalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()
    if graph_type == 'horizontal':
        g = HorizontalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()

# calculate the average shortest path length
    avg_path_len = nx.average_shortest_path_length(nxg)

    AvgPathLength.append(avg_path_len)
```

```
ind_names = ['AvgPathLength']

indicators = [AvgPathLength]

measure_labels = [r'$ApLen$']

file_names = []

for i in range(len(ind_names)):
    name = f"{ind_names[i]}_symbol={symbol}_wind={window}_step={tstep}_series
type={ret_type}_graph_type={graph_type}"
    np.savetxt(name +".txt", indicators[i])
    file_names.append(name)
```

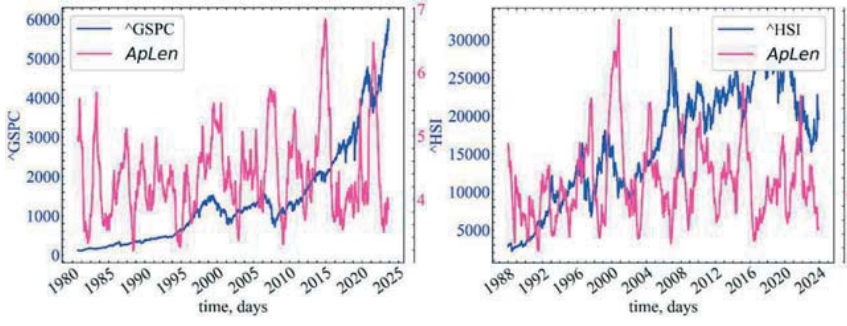
6.2.4.12.1 Average shortest path length

Paying attention to the length of the shortest path between two vertices i and j, we can define a measure called the **average shortest path length**:

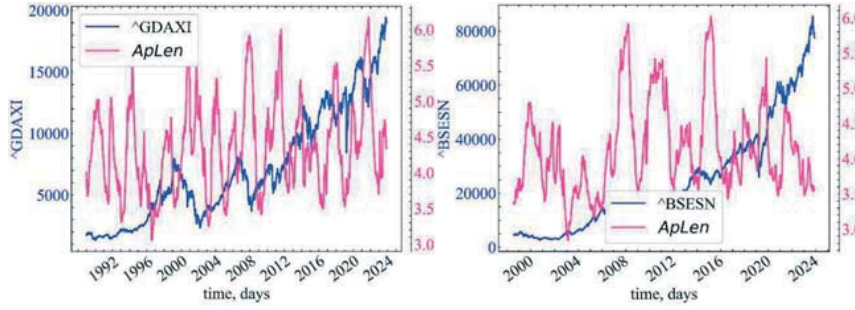
$$ApLen = \frac{1}{N(N-1)} \sum_{i \neq j} l_{ij}.$$

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[0],
          ylabel,
          measure_labels[0],
          xlabel,
          file_names[0],
          clr="deeppink")
```

Fig. 6.24 provides the comparative dynamics of S&P 500, Hang Seng index, DAX, BSE Sensex, and their average shortest path length indicator.



(a) (b)



(c) (d)

Fig. 6.24: Dynamics of S&P 500 (a), Hang Seng index (b), DAX (c), BSE Sensex (d), and their average shortest path length index

Fig. 6.24 shows that $ApLen$ is characterized by a decline in pre-crisis periods and an increase in crisis and post-crisis periods. Similar to the previous indicators, which only relied on the length of the shortest path between pairs of nodes, $ApLen$ indicates an increase in the efficiency of information transfer between market traders. It can also be said that on the built stock indices visibility graphs, past values "see" the present ones in a more efficient way, which is reflected in the market's persistence in the pre-crisis period.

6.3 Conclusions on network analysis

This chapter demonstrates the possibility of studying complex socio-economic systems within the framework of the network paradigm of complexity. The time series of stock indices were represented in an equivalent way – by a visibility network that has a wide range of characteristics: both spectral and topological. Examples of stock indices crashes have shown that most network indicators can serve as precursor indicators of crisis phenomena and can be used for possible early warning of unwanted crises in stock market.

References

1. A. A. Ganchuk, V. N. Soloviev, D. Chabanenko, *Forecasting Methods: A Study Guide* (Cherkasy: Brama-Ukraine, 2012).
2. A. Barrat and M. Weigt, *On the Properties of Small-World Network Models*, The European Physical Journal B-Condensed Matter and Complex Systems **13**, 547 (2000).
3. A. Berman and R. J. Plemmons, *Nonnegative Matrices in the Mathematical Sciences* (Society for Industrial; Applied Mathematics, 1994).
4. A. Bielinskyi, S. Semerikov, O. Serdyuk, V. Solovieva, V. N. Soloviev, and L. Pichl, *Econophysics of Sustainability Indices*, in *Proceedings of the Selected Papers of the Special Edition of International Conference on Monitoring, Modeling & Management of Emergent Economy (M3E2-MLPEED 2020), Odessa, Ukraine, July 13-18, 2020*, edited by A. Kiv, Vol. 2713 (CEUR-WS.org, 2020), pp. 372–392.
5. A. Bielinskyi, V. Soloviev, V. Solovieva, A. Matviychuk, and S. Semerikov, *The Analysis of Multifractal Cross-Correlation Connectedness Between Bitcoin and the Stock Market*, in *Information Technology for Education, Science, and Technics*, edited by E. Faure, O. Danchenko, M. Bondarenko, Y. Tryus, C. Bazilo, and G. Zaspa (Springer Nature Switzerland, Cham, 2023), pp. 323–345.
6. A. Faini, G. Parati, and P. Castiglioni, *Multiscale Assessment of the Degree of Multifractality for Physiological Time Series*, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **379**, 20200254 (2021).
7. A. Kalauzi, T. Bojić, and L. Rakić, *Extracting Complexity Waveforms from One-Dimensional Signals*, Nonlinear Biomedical Physics **3**, 1 (2009).
8. A. Kasprzak, R. Kutner, J. Perelló, and J. Masoliver, *Higher-Order Phase Transitions on Financial Markets*, The European Physical Journal B: Condensed Matter and Complex Systems **76**, 513 (2010).
9. A. Kiv, A. Bryukhanov, V. Soloviev, A. Bielinskyi, T. Kavetskyy, D. Dyachok, I. Donchev, and V. Lukashin, *Complex Network Methods for Plastic Deformation Dynamics in Metals*, Dynamics **3**, 34 (2023).
10. A. Lempel and J. Ziv, *On the Complexity of Finite Sequences*, IEEE Transactions on Information Theory **22**, 75 (1976).
11. A. Merchant et al. *Scaling deep learning for materials discovery*, Nature **624**, 80-90 (2023).
12. A. N. Kolmogorov, *Three Approaches to the Quantitative Definition of Information*, International Journal of Computer Mathematics **2**, 157 (1968).

13. A. O. Bielinskyi and V. N. Soloviev, *Complex Network Precursors of Crashes and Critical Events in the Cryptocurrency Market*, in *Proceedings of St Student Workshop on Computer Science and Software Engineering, CS and SE@SW 2018, Kryvyi Rih, Ukraine, November 30, 2018*, edited by S. O. Semerikov, A. M. Striuk, V. N. Soloviev, and A. E. Kiv, Vol. 2292 (CEUR-WS.org, 2028), pp. 37–45.
14. A. O. Bielinskyi, A. V. Matviychuk, O. A. Serdyuk, S. O. Semerikov, V. V. Solovieva, and V. N. Soloviev, *Correlational and Non-Extensive Nature of Carbon Dioxide Pricing Market*, in *ICTERI 2021 Workshops*, edited by O. Ignatenko, V. Kharchenko, V. Kobets, H. Kravtsov, Y. Tarasich, V. Ermolayev, D. Esteban, V. Yakovyna, and A. Spivakovsky, Vol. 1635 (Springer International Publishing, Cham, 2022), pp. 183–199.
15. A. O. Bielinskyi, O. A. Serdyuk, S. O. Semerikov, and V. N. Soloviev, *Econophysics of Cryptocurrency Crashes: A Systematic Review*, in *Proceedings of the Selected and Revised Papers of 9th International Conference on Monitoring, Modeling & Management of Emergent Economy (M3E2-MLPEED 2021), Odessa, Ukraine, May 26-28, 2021*, edited by A. E. Kiv, V. N. Soloviev, and S. O. Semerikov, Vol. 3048 (CEUR-WS.org, 2021), pp. 31–133.
16. A. O. Bielinskyi, V. N. Soloviev, S. O. Semerikov, and V. V. Solovieva, *Identifying Stock Market Crashes by Fuzzy Measures of Complexity*, Neuro-Fuzzy Modeling Techniques in Economics **10**, 3 (2021).
17. A. O. Bielinskyi, V. N. Soloviev, S. V. Hushko, A. E. Kiv, and A. V. Matviychuk, *High-Order Network Analysis for Financial Crash Identification*, in *Proceedings of the Selected and Revised Papers of 10th International Conference on Monitoring, Modeling & Management of Emergent Economy (M3E2-MLPEED 2022), Virtual Event, Kryvyi Rih, Ukraine, November 17-18, 2022*, edited by H. B. Danylchuk and S. O. Semerikov, Vol. 3465 (CEUR-WS.org, 2022), pp. 132–149.
18. A. O. Bielinskyi, V. N. Soloviev, V. Solovieva, S. O. Semerikov, and M. A. Radin, *Recurrence Quantification Analysis of Energy Market Crises: A Nonlinear Approach to Risk Management*, in *Proceedings of the Selected and Revised Papers of 10th International Conference on Monitoring, Modeling & Management of Emergent Economy (M3E2-MLPEED 2022), Virtual Event, Kryvyi Rih, Ukraine, November 17-18, 2022*, edited by H. B. Danylchuk and S. O. Semerikov, Vol. 3465 (CEUR-WS.org, 2022), pp. 110–131.
19. A. Orozco-Duque, D. Novak, V. Kremen, and J. Bustamante, *Multifractal Analysis for Grading Complex Fractionated Electrograms in Atrial Fibrillation*, Physiological Measurement **36**, 2269 (2015).

20. A. Petrosian, *Kolmogorov Complexity of Finite Sequences and Recognition of Different Preictal EEG Patterns*, in *Proceedings Eighth IEEE Symposium on Computer-Based Medical Systems* (1995), pp. 212–217.
21. A. Prieto-Guerrero and G. Espinosa-Paredes, *Dynamics of BWRs and Mathematical Models*, in *Linear and Non-Linear Stability Analysis in Boiling Water Reactors*, edited by A. Prieto-Guerrero and G. Espinosa-Paredes (Woodhead Publishing, 2019), pp. 193–268.
22. A. Tomashin, G. Leonardi, and S. Wallot, *Four Methods to Distinguish Between Fractal Dimensions in Time Series Through Recurrence Quantification Analysis*, Entropy **24**, (2022).
23. A. Vulpiani, *Lewis Fry Richardson: Scientist, Visionary and Pacifist*, Lettera Matematica **2**, 121 (2014).
24. A. Wolf, J. B. Swift, H. L. Swinney, and J. A. Vastano, *Determining Lyapunov Exponents from a Time Series*, Physica D: Nonlinear Phenomena **16**, 285 (1985).
25. B. B. Mandelbrot and J. A. Wheeler, *The Fractal Geometry of Nature*, American Journal of Physics **51**, 286 (1983).
26. B. B. Mandelbrot, C. J. G. Evertsz, and Y. Hayakawa, *Exactly Self-Similar Left-Sided Multifractal Measures*, Phys. Rev. A **42**, 4528 (1990).
27. B. Hayes, *Computing Science: Statistics of Deadly Quarrels*, American Scientist **90**, 10 (2002).
28. B. Hjorth, *EEG Analysis Based on Time Domain Properties*, Electroencephalography and Clinical Neurophysiology **29**, 306 (1970).
29. B. K. Hillen, G. T. Yamaguchi, J. J. Abbas, and R. Jung, *Joint-Specific Changes in Locomotor Complexity in the Absence of Muscle Atrophy Following Incomplete Spinal Cord Injury*, Journal of NeuroEngineering and Rehabilitation **10**, 1 (2013).
30. B. Luque, L. Lacasa, F. Ballesteros, and J. Luque, *Horizontal Visibility Graphs: Exact Results for Random Time Series*, Phys. Rev. E **80**, 046103 (2009).
31. B. Mandelbrot, *How Long Is the Coast of Britain? Statistical Self-Similarity and Fractional Dimension*, Science **156**, 636 (1967).
32. C. Anteneodo and C. Tsallis, *Breakdown of Exponential Sensitivity to Initial Conditions: Role of the Range of Interactions*, Phys. Rev. Lett. **80**, 5313 (1998).
33. C. Bandt and B. Pompe, *Permutation Entropy: A Natural Complexity Measure for Time Series*, Phys. Rev. Lett. **88**, 174102 (2002).
34. C. Berge, *Théorie Des Graphes Et Ses Applications* (Dunod, 1958).
35. C. E. Shannon, *A Mathematical Theory of Communication*, Bell System Technical Journal **27**, 379 (1948).

36. C. F. Vega and J. Noel, *Parameters Analyzed of Higuchi's Fractal Dimension for EEG Brain Signals*, in *2015 Signal Processing Symposium (SPSympo)* (2015), pp. 1–5.
37. C. Goh, B. Hamadicharef, G. T. Henderson, and E. C. Ifeachor, *Comparison of Fractal Dimension Algorithms for the Computation of EEG Biomarkers for Dementia*, in *2nd International Conference on Computational Intelligence in Medicine and Healthcare (CIMED2005)* (Professor José Manuel Fonseca, UNINOVA, Portugal, Lisbon, Portugal, 2005).
38. C. J. Gavilán-Moreno and G. Espinosa-Paredes, *Using Largest Lyapunov Exponent to Confirm the Intrinsic Stability of Boiling Water Reactors*, Nuclear Engineering and Technology **48**, 434 (2016).
39. C. L. Webber and J. P. Zbilut, *Dynamical Assessment of Physiological Systems and States Using Recurrence Plot Strategies*, Journal of Applied Physiology **76**, 965 (1994).
40. C. Sevcik, *A Procedure to Estimate the Fractal Dimension of Waveforms*, (2010).
41. C. Taufemback, R. Giglio, and S. D. Silva, *Algorithmic complexity theory detects decreases in the relative efficiency of stock markets in the aftermath of the 2008 financial crisis*, Economics Bulletin **31**, 1631 (2011).
42. C. Tsallis, *Beyond Boltzmann–Gibbs–Shannon in Physics and Elsewhere*, Entropy **21**, (2019).
43. C. Tsallis, *Dynamical Scenario for Nonextensive Statistical Mechanics*, Physica A: Statistical Mechanics and Its Applications **340**, 1 (2004).
44. C. Tsallis, *Economics and Finance: Q-Statistical Stylized Features Galore*, Entropy **19**, (2017).
45. C. Tsallis, M. Gell-Mann, and Y. Sato, *Asymptotically Scale-Invariant Occupancy of Phase Space Makes the Entropy* S_q *Extensive*, Proceedings of the National Academy of Sciences **102**, 15377 (2005).
46. C. Tsallis, *Possible Generalization of Boltzmann-Gibbs Statistics*, Journal of Statistical Physics **52**, 479 (1988).
47. C. Tsallis, *Some Open Problems in Nonextensive Statistical Mechanics*, International Journal of Bifurcation and Chaos **22**, 1230030 (2012).
48. C.-K. Peng, S. Havlin, H. E. Stanley, and A. L. Goldberger, *Quantification of Scaling Exponents and Crossover Phenomena in Nonstationary Heartbeat Time Series*, Chaos: An Interdisciplinary Journal of Nonlinear Science **5**, 82 (1995).
49. C.-K. Peng, S. V. Buldyrev, S. Havlin, M. Simons, H. E. Stanley, and A. L. Goldberger, *Mosaic Organization of DNA Nucleotides*, Phys. Rev. E **49**, 1685 (1994).

50. D. G. Bonchev, *Information Theoretic Complexity Measures*, in *Encyclopedia of Complexity and Systems Science*, edited by R. A. Meyers (Springer New York, New York, NY, 2009), pp. 4820–4839.
51. D. J. Watts and S. H. Strogatz, *Collective Dynamics of 'Small-World' Networks*, Nature **393**, 440 (1998).
52. D. M. Cvetkovic, M. Doob, and H. Sachs, *Spectra of Graphs. Theory and Application* (Academic Press, 1980).
53. D. Nychka, S. Ellner, A. R. Gallant, and D. McCaffrey, *Finding Chaos in Noisy Systems*, Journal of the Royal Statistical Society. Series B (Methodological) **54**, 399 (1992).
54. D. Stosic, D. Stosic, T. B. Ludermir, and T. Stosic, *Nonextensive Triplets in Cryptocurrency Exchanges*, Physica A: Statistical Mechanics and Its Applications **505**, 1069 (2018).
55. Derbentsev V. D., Serdyuk O. A., Soloviev V. N., Sharapov O. D., *Synergetic and econophysical methods of studying the dynamic and structural characteristics of economic systems: a monograph* (Cherkasy: Brama-Ukraine, 2010).
56. E. A. Ihlen, *Introduction to Multifractal Detrended Fluctuation Analysis in Matlab*, Frontiers in Physiology **3**, (2012).
57. E. Canessa, *Multifractality in Time Series*, Journal of Physics A: Mathematical and General **33**, 3637 (2000).
58. E. Estevez-Rams, R. Lora Serrano, B. Aragón Fernández, and I. Brito Reyes, *On the non-randomness of maximum Lempel Ziv complexity sequences of finite size*, Chaos: An Interdisciplinary Journal of Nonlinear Science **23**, 023118 (2013).
59. E. Estrada, *Spectral Scaling and Good Expansion Properties in Complex Networks*, Europhysics Letters **73**, 649 (2006).
60. E. G. Pavlos, O. E. Malandraki, O. V. Khabarova, L. P. Karakatsanis, G. P. Pavlos, and G. Livadiotis, *Non-Extensive Statistical Analysis of Energetic Particle Flux Enhancements Caused by the Interplanetary Coronal Mass Ejection-Heliospheric Current Sheet Interaction*, Entropy **21**, (2019).
61. E. Maiorino, L. Livi, A. Giuliani, A. Sadeghian, and A. Rizzi, *Multifractal Characterization of Protein Contact Networks*, Physica A: Statistical Mechanics and Its Applications **428**, 302 (2015).
62. F. Hasselman, *When the Blind Curve Is Finite: Dimension Estimation and Model Inference Based on Empirical Waveforms*, Frontiers in Physiology **4**, (2013).

63. F. Liao and Y.-K. Jan, *Using Multifractal Detrended Fluctuation Analysis to Assess Sacral Skin Blood Flow Oscillations in People with Spinal Cord Injury*, The Journal of Rehabilitation Research and Development **48**, 787 (2011).
64. F. Mormann, T. Kreuz, C. Rieke, R. G. Andrzejak, A. Kraskov, P. David, C. E. Elger, and K. Lehnertz, *On the Predictability of Epileptic Seizures*, Clinical Neurophysiology **116**, 569 (2005).
65. F. R. K. Chung, *Spectral Graph Theory* (American Mathematical Society, 1997).
66. F. Takens, *Detecting Strange Attractors in Turbulence*, in *Dynamical Systems and Turbulence, Warwick 1980*, edited by D. Rand and L.-S. Young (Springer Berlin Heidelberg, Berlin, Heidelberg, 1981), pp. 366–381.
67. G. Bianconi et al., *Complex systems in the spotlight: next steps after the 2021 Nobel Prize in Physics*, J. Phys. Complex. **4**, 010201 (2023).
68. G. Bounova and O. de Weck, *Overview of Metrics and Their Correlation Patterns for Multiple-Metric Topology Analysis on Heterogeneous Graph Ensembles*, Phys. Rev. E **85**, 016117 (2012).
69. G. L. Ferri, M. F. Reynoso Savio, and A. Plastino, *Tsallis' q-Triplet and the Ozone Layer*, Physica A: Statistical Mechanics and Its Applications **389**, 1829 (2010).
70. G. Nicolis, I. Prigogine, W. H. Freeman, and Company, *Exploring Complexity: An Introduction* (W.H. Freeman, 1989).
71. G. Pavlos, A. Iliopoulos, L. Karakatsanis, M. Xenakis, and E. Pavlos, *Complexity of Economical Systems.*, Journal of Engineering Science & Technology Review **8**, (2015).
72. G. R. Jafari, P. Pedram, and L. Hedayatifar, *Erratum: Long-Range Correlation and Multifractality in Bach's Inventions Pitches*, Journal of Statistical Mechanics: Theory and Experiment **2012**, E03001 (2012).
73. H. D. I. Abarbanel, R. Brown, J. J. Sidorowich, and L. Sh. Tsimring, *The Analysis of Observed Chaotic Data in Physical Systems*, Rev. Mod. Phys. **65**, 1331 (1993).
74. H. E. Hurst, *A Suggested Statistical Model of Some Time Series Which Occur in Nature*, Nature **180**, 494 (1957).
75. H. E. Hurst, *Long-Term Storage Capacity of Reservoirs*, Transactions of the American Society of Civil Engineers **116**, 770 (1951).
76. H. F. Jelinek, N. Elston, and B. Zietsch, *Fractal Analysis: Pitfalls and Revelations in Neuroscience*, in *Fractals in Biology and Medicine*, edited by G. A. Losa, D. Merlini, T. F. Nonnenmacher, and E. R. Weibel (Birkhäuser Basel, Basel, 2005), pp. 85–94.

77. H. Kantz and T. Schreiber, *Nonlinear Time Series Analysis* (Cambridge University Press, 2004).
78. H. Steinhaus, *Length, Shape and Area*, in *Colloquium Mathematicum*, Vol. 3 (Polska Akademia Nauk. Instytut Matematyczny PAN, 1954), pp. 1–13.
79. H.-B. Xie, W.-X. He, and H. Liu, *Measuring Time Series Regularity Using Nonlinear Similarity-Based Sample Entropy*, Physics Letters A **372**, 7140 (2008).
80. I. Gutman, *The Energy of a Graph*, Ber. Math.— Statist. Sekt. Forschungsz **103**, 1-22. (1978).
81. I. J. Schoenberg, *Publications of Edmund Landau*, in *Number Theory and Analysis: A Collection of Papers in Honor of Edmund Landau (1877–1938)*, edited by P. Turán (Springer US, Boston, MA, 1969), pp. 335–355.
82. I. T. Pedron, *Correlation and Multifractality in Climatological Time Series*, Journal of Physics: Conference Series **246**, 012034 (2010).
83. I. V. Bezsudnov and A. A. Snarskii, *From the Time Series to the Complex Networks: The Parametric Natural Visibility Graph*, Physica A: Statistical Mechanics and Its Applications **414**, 53 (2014).
84. J. C. Crepeau and L. K. Isaacson, *Spectral Entropy Measurements of Coherent Structures in am Evolving Shear Layer*, Journal of Non-Equilibrium Thermodynamics 16, 137 (1991).
85. J. P. Zbilut and C. L. Webber, *Embeddings and Delays as Derived from Quantification of Recurrence Plots*, Physics Letters A **171**, 199 (1992).
86. J. S. Richman and J. R. Moorman, *Physiological Time-Series Analysis Using Approximate Entropy and Sample Entropy*, American Journal of Physiology-Heart and Circulatory Physiology **278**, H2039 (2000).
87. J. W. Kantelhardt, E. Koscielny-Bunde, H. H. A. Rego, S. Havlin, and A. Bunde, *Detecting Long-Range Correlations with Detrended Fluctuation Analysis*, Physica A: Statistical Mechanics and Its Applications **295**, 441 (2001).
88. J. W. Kantelhardt, *Fractal and Multifractal Time Series*, in *Mathematics of Complexity and Dynamical Systems*, edited by R. A. Meyers (Springer New York, New York, NY, 2011), pp. 463–487.
89. J. W. Kantelhardt, S. A. Zschiegner, E. Koscielny-Bunde, S. Havlin, A. Bunde, and H. E. Stanley, *Multifractal Detrended Fluctuation Analysis of Nonstationary Time Series*, Physica A: Statistical Mechanics and Its Applications **316**, 87 (2002).

90. J. Wu, M. Barahona, Y.-J. Tan, and H.-Z. Deng, *Spectral Measure of Structural Robustness in Complex Networks*, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans **41**, 1244 (2011).
91. J.-L. Blanc, L. Pezard, and A. Lesne, *Delay Independence of Mutual-Information Rate of Two Symbolic Sequences*, Phys. Rev. E **84**, 036214 (2011).
92. J.-P. Eckmann and D. Ruelle, *Ergodic Theory of Chaos and Strange Attractors*, Rev. Mod. Phys. **57**, 617 (1985).
93. J.-P. Eckmann, S. O. Kamphorst, and D. Ruelle, *Recurrence Plots of Dynamical Systems*, Europhysics Letters **4**, 973 (1987).
94. J.-P. Eckmann, S. O. Kamphorst, D. Ruelle, and S. Ciliberto, *Liapunov Exponents from Time Series*, Phys. Rev. A **34**, 4971 (1986).
95. K. Falconer, *Fractal Geometry: Mathematical Foundations and Applications* (John Wiley & Sons, 2003).
96. K. Shockley and M. Riley, *In Recurrence Quantification Analysis: Theory and Best Practices*, 1st ed. (Springer, New York, 2015).
97. L. Boltzmann, *Weitere Studien über Das wärmegleichgewicht Unter Gasmolekülen*, in *Kinetische Theorie II: Irreversible Prozesse Einführung Und Originaltexte* (Vieweg+Teubner Verlag, Wiesbaden, 1970), pp. 115–225.
98. L. Lacasa, A. Nuñez, É. Roldán, J. M. R. Parrondo, and B. Luque, *Time Series Irreversibility: A Visibility Graph Approach*, The European Physical Journal B **85**, (2012).
99. L. Lacasa, B. Luque, F. Ballesteros, J. Luque, and J. C. Nuño, *From Time Series to Complex Networks: The Visibility Graph*, Proceedings of the National Academy of Sciences **105**, 4972 (2008).
100. L. T. Lui, G. Terrazas, H. Zenil, C. Alexander, and N. Krasnogor, *Complexity Measurement Based on Information Theory and Kolmogorov Complexity*, Artificial Life **21**, 205 (2015).
101. L. Telesca, V. Lapenna, and M. Macchiato, *Multifractal Fluctuations in Seismic Interspike Series*, Physica A: Statistical Mechanics and Its Applications **354**, 629 (2005).
102. M. Balcerzak, D. Pikunov, and A. Dabrowski, *The Fastest, Simplified Method of Lyapunov Exponents Spectrum Estimation for Continuous-Time Dynamical Systems*, Nonlinear Dynamics **94**, 3053 (2018).
103. M. Borowska, *Multiscale Permutation Lempel–Ziv Complexity Measure for Biomedical Signal Analysis: Interpretation and Application to Focal EEG Signals*, Entropy **23**, (2021).

104. M. D. Costa, C.-K. Peng, and A. L. Goldberger, *Multiscale Analysis of Heart Rate Dynamics: Entropy and Time Irreversibility Measures*, Cardiovascular Engineering **8**, 88 (2008).

105. M. Dai, C. Zhang, and D. Zhang, *Multifractal and Singularity Analysis of Highway Volume Data*, Physica A: Statistical Mechanics and Its Applications **407**, 332 (2014).

106. M. Dai, J. Hou, and D. Ye, *Multifractal Detrended Fluctuation Analysis Based on Fractal Fitting: The Long-Range Correlation Detection Method for Highway Volume Data*, Physica A: Statistical Mechanics and Its Applications **444**, 722 (2016).

107. M. E. J. Newman, *Assortative Mixing in Networks*, Phys. Rev. Lett. **89**, 208701 (2002).

108. M. G. Kendall, *Further Contributions to the Theory of Paired Comparisons*, Biometrics **11**, 43 (1955).

109. M. J. Katz, *Fractals and the Analysis of Waveforms*, Computers in Biology and Medicine **18**, 145 (1988).

110. M. Li and P. Vitányi, *Preliminaries*, in *An Introduction to Kolmogorov Complexity and Its Applications* (Springer New York, New York, NY, 2008), pp. 1–99.

111. M. Rostaghi and H. Azami, *Dispersion Entropy: A Measure for Time-Series Analysis*, IEEE Signal Processing Letters **23**, 610 (2016).

112. M. S. Kanwal, J. A. Grochow, and N. Ay, *Comparing Information-Theoretic Measures of Complexity in Boltzmann Machines*, Entropy **19**, (2017).

113. M. S. Movahed, F. Ghasemi, S. Rahvar, and M. R. R. Tabar, *Long-Range Correlation in Cosmic Microwave Background Radiation*, Phys. Rev. E **84**, 021103 (2011).

114. M. Sano and Y. Sawada, *Measurement of the Lyapunov Spectrum from a Chaotic Time Series*, Phys. Rev. Lett. **55**, 1082 (1985).

115. M. T. Rosenstein, J. J. Collins, and C. J. De Luca, *A Practical Method for Calculating Largest Lyapunov Exponents from Small Data Sets*, Physica D: Nonlinear Phenomena **65**, 117 (1993).

116. M. Wątorek, S. Drożdż, J. Kwapień, L. Minati, P. Oświęcimka, and M. Stanuszek, *Multiscale Characteristics of the Emerging Global Cryptocurrency Market*, Physics Reports **901**, 1 (2021).

117. N. Biggs, *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics 92)*, Bulletin of the London Mathematical Society **30**, 197 (1998).

118. N. H. Packard, J. P. Crutchfield, J. D. Farmer, and R. S. Shaw, *Geometry from a Time Series*, Phys. Rev. Lett. **45**, 712 (1980).

119. N. Marwan, N. Wessel, U. Meyerfeldt, A. Schirdewan, and J. Kurths, *Recurrence-Plot-Based Measures of Complexity and Their Application to Heart-Rate-Variability Data*, Phys. Rev. E **66**, 026702 (2002).

120. P. Bonacich, *Technique for Analyzing Overlapping Memberships*, Sociological Methodology **4**, 176 (1972).

121. P. G. Lind, M. C. González, and H. J. Herrmann, *Cycles and Clustering in Bipartite Networks*, Phys. Rev. E **72**, 056127 (2005).

122. P. Grassberger and I. Procaccia, *Characterization of Strange Attractors*, Phys. Rev. Lett. **50**, 346 (1983).

123. P. Grassberger and I. Procaccia, *Measuring the Strangeness of Strange Attractors*, Physica D: Nonlinear Phenomena **9**, 189 (1983).

124. P. Grassberger, *Generalized Dimensions of Strange Attractors*, Physics Letters A **97**, 227 (1983).

125. P. H. Figueirêdo, E. Nogueira, M. A. Moret, and S. Coutinho, *Multifractal Analysis of Polyalanines Time Series*, Physica A: Statistical Mechanics and Its Applications **389**, 2090 (2010).

126. P. Holme, C. R. Edling, and F. Liljeros, *Structure and Time Evolution of an Internet Dating Community*, Social Networks **26**, 155 (2004).

127. P. Holme, F. Liljeros, C. R. Edling, and B. J. Kim, *Network Bipartivity*, Phys. Rev. E **68**, 056107 (2003).

128. P. Oświęcimka, L. Livi, and S. Drożdż, *Right-Side-Stretched Multifractal Spectra Indicate Small-Worldness in Networks*, Communications in Nonlinear Science and Numerical Simulation **57**, 231 (2018).

129. P. W. Anderson, *More is different*, New Series **77**, 393-396 (1972).

130. P. Zhang, J. Wang, X. Li, M. Li, Z. Di, and Y. Fan, *Clustering Coefficient and Community Structure of Bipartite Networks*, Physica A: Statistical Mechanics and Its Applications **387**, 6869 (2008).

131. Q. Xuan, J. Zhou, K. Qiu, D. Xu, S. Zheng, and X. Yang, *CLPVG: Circular limited penetrable visibility graph as a new network model for time series*, Chaos: An Interdisciplinary Journal of Nonlinear Science **32**, 013130 (2022).

132. R. A. Fisher and E. J. Russell, *On the Mathematical Foundations of Theoretical Statistics*, Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character **222**, 309 (1922).

133. R. Clausius, T. A. Hirst, and J. Tyndall, *The Mechanical Theory of Heat: With Its Applications to the Steam-Engine and to the Physical Properties of Bodies* (J. Van Voorst, 1867).

134. R. de Oliveira, S. Brito, L. da Silva, and C. Tsallis, *Connecting Complex Networks to Nonadditive Entropies*, Scientific Reports **11**, 1130 (2021).

135. R. Esteller, G. Vachtsevanos, J. Echauz, and B. Litt, *A Comparison of Waveform Fractal Dimension Algorithms*, IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications **48**, 177 (2001).

136. R. F. Voss, *Fractals in Nature: From Characterization to Simulation*, in *The Science of Fractal Images*, edited by H.-O. Peitgen and D. Saupe (Springer New York, New York, NY, 1988), pp. 21–70.

137. R. Giglio and S. Da Silva, Ranking the Stocks Listed on Bovespa According to Their Relative Efficiency, MPRA Paper, University Library of Munich, Germany, 2009.

138. R. Giglio, R. Matsushita, A. Figueiredo, I. Gleria, and S. D. Silva, *Algorithmic Complexity Theory and the Relative Efficiency of Financial Markets*, Europhysics Letters **84**, 48005 (2008).

139. R. Pastor-Satorras, A. Vázquez, and A. Vespignani, *Dynamical and Correlation Properties of the Internet*, Phys. Rev. Lett. **87**, 258701 (2001).

140. R. Rak, S. Drożdż, J. Kwapień, and P. Oświęcimka, *Detrended Cross-Correlations Between Returns, Volatility, Trading Activity, and Volume Traded for the Stock Market Companies*, Europhysics Letters **112**, 48001 (2015).

141. R. V. Donner, M. Small, J. F. Donges, N. Marwan, Y. Zou, R. Xiang, and J. Kurths, *Recurrence-Based Time Series Analysis by Means of Complex Network Methods*, International Journal of Bifurcation and Chaos **21**, 1019 (2011).

142. S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwang, *Complex Networks: Structure and Dynamics*, Physics Reports **424**, 175 (2006).

143. S. Butler, *Interlacing for Weighted Graphs Using the Normalized Laplacian*, Electronic Journal of Linear Algebra **16**, 90 (2007).

144. S. Drożdż and P. Oświęcimka, *Detecting and Interpreting Distortions in Hierarchical Organization of Complex Time Series*, Phys. Rev. E **91**, 030902 (2015).

145. S. Drożdż, R. Kowalski, P. Oświęcimka, R. Rak, and R. Gębarowski, *Dynamical Variety of Shapes in Financial Multifractality*, Complexity **2018**, 1 (2018).

146. S. Dutta, *Multifractal Properties of ECG Patterns of Patients Suffering from Congestive Heart Failure*, Journal of Statistical Mechanics: Theory and Experiment **2010**, P12021 (2010).

147. S. J. Roberts, W. Penny, and I. Rezek, *Temporal and Spatial Complexity Measures for Electroencephalogram Based Brain-Computer Interfacing*, Medical & Biological Engineering & Computing **37**, 93 (1999).

148. S. M. Pincus, *Approximate Entropy as a Measure of System Complexity*, Proceedings of the National Academy of Sciences **88**, 2297 (1991).

149. S. M. Pincus, I. M. Gladstone, and R. A. Ehrenkranz, *A Regularity Statistic for Medical Data Analysis*, Journal of Clinical Monitoring **7**, 335 (1991).

150. S. Maslov and K. Sneppen, *Specificity and Stability in Topology of Protein Networks*, Science **296**, 910 (2002).

151. S. Umarov, C. Tsallis, and S. Steinberg, *On Aq-Central Limit Theorem Consistent with Nonextensive Statistical Mechanics*, Milan Journal of Mathematics **76**, 307 (2008).

152. S. V. Bozhokin and D. A. Parshin, *Fractals and Multifractals: Textbook* (Scientific; Publishing Center "Regular; Chaotic Dynamics", 2001).

153. S. Zozor, P. Ravier, and O. Buttelli, *On Lempel–Ziv Complexity for Multidimensional Data Analysis*, Physica A: Statistical Mechanics and Its Applications **345**, 285 (2005).

154. T. C. Halsey, M. H. Jensen, L. P. Kadanoff, I. Procaccia, and B. I. Shraiman, *Fractal Measures and Their Singularities: The Characterization of Strange Sets*, Nuclear Physics B - Proceedings Supplements **2**, 501 (1987).

155. T. Gneiting, H. Ševčíková, and D. B. Percival, *Estimators of Fractal Dimension: Assessing the Roughness of Time Series and Spatial Data*, Statistical Science **27**, 247 (2012).

156. T. H. Wei, *The Algebraic Foundations of Ranking Theory* (University of Cambridge, 1952).

157. T. Higuchi, *Approach to an Irregular Time Series on the Basis of the Fractal Theory*, Physica D: Nonlinear Phenomena **31**, 277 (1988).

158. T. Rawald, Scalable and Efficient Analysis of Large High-Dimensional Data Sets in the Context of Recurrence Analysis, PhD thesis, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät, 2018.

159. T. T. Zhou, N. D. Jin, Z. K. Gao, and Y. B. Luo, *Limited Penetrable Visibility Graph for Establishing Complex Network from Time Series*, Acta Physica Sinica **61**, 2012-3-030506 (2012).

160. U. Parlitz, *Identification of True and Spurious Lyapunov Exponents from Time Series*, International Journal of Bifurcation and Chaos **02**, 155 (1992).

161. V. Latora and M. Marchiori, *Efficient Behavior of Small-World Networks*, Phys. Rev. Lett. **87**, 198701 (2001).

162. V. Marmelat, K. Torre, and D. Delignieres, *Relative Roughness: An Index for Testing the Suitability of the Monofractal Model*, Frontiers in Physiology **3**, (2012).

163. V. N. Soloviev and A. Belinskiy, *Complex Systems Theory and Crashes of Cryptocurrency Market*, in *Information and Communication Technologies in Education, Research, and Industrial Applications*, edited by V. Ermolayev, M. C. Suárez-Figueroa, V. Yakovyna, H. C. Mayr, M. Nikitchenko, and A. Spivakovsky (Springer International Publishing, Cham, 2019), pp. 276–297.

164. V. N. Soloviev and A. Belinskyi, *Methods of Nonlinear Dynamics and the Construction of Cryptocurrency Crisis Phenomena Precursors*, in *Proceedings of the 14th International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer. Volume II: Workshops, Kyiv, Ukraine, May 14-17, 2018*, edited by V. Ermolayev, M. C. Suárez-Figueroa, V. Yakovyna, V. S. Kharchenko, V. Kobets, H. Kravtsov, V. S. Peschanenko, Y. Prytula, M. S. Nikitchenko, and A. Spivakovsky, Vol. 2104 (CEUR-WS.org, 2018), pp. 116–127.

165. V. N. Soloviev, A. Bielinskyi, and V. Solovieva, *Entropy Analysis of Crisis Phenomena for DJIA Index*, in *Proceedings of the 15th International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer. Volume II: Workshops, Kherson, Ukraine, June 12-15, 2019*, edited by V. Ermolayev, F. Mallet, V. Yakovyna, V. S. Kharchenko, V. Kobets, A. Kornilowicz, H. Kravtsov, M. S. Nikitchenko, S. Semerikov, and A. Spivakovsky, Vol. 2393 (CEUR-WS.org, 2019), pp. 434–449.

166. V. N. Soloviev, A. Bielinskyi, O. Serdyuk, V. Solovieva, and S. Semerikov, *Lyapunov Exponents as Indicators of the Stock Market Crashes*, in *Proceedings of the 16th International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer. Volume II: Workshops, Kharkiv, Ukraine, October 06-10, 2020*, edited by O. Sokolov, G. Zholtkevych, V. Yakovyna, Y. Tarasich, V. Kharchenko, V. Kobets, O. Burov, S. Semerikov, and H. Kravtsov, Vol. 2732 (CEUR-WS.org, 2020), pp. 455–470.

167. V. N. Soloviev, A. O. Bielinskyi, and N. A. Kharadzjan, *Coverage of the Coronavirus Pandemic Through Entropy Measures*, in *3rd Workshop for Young Scientists in Computer*

Science and Software Engineering (CS and SE and SW 2020), Kryvyi Rih, Ukraine, November 27, 2020, edited by A. E. Kiv, S. O. Semerikov, V. N. Soloviev, and A. M. Striuk, Vol. 2832 (CEUR-WS.org, 2021), pp. 24–42.

168. V. N. Soloviev, A. O. Bielinskyi, *Complex Systems Modeling in Python: A manual for self-study of the discipline* (Cherkasy: O.M. Tretyakov, 2024).
169. V. N. Soloviev, *Mathematical economics: a study guide for self-study* (B. Khmelnytsky ChNU Publishing House, 2008).
170. V. N. Soloviev, O. A. Serdyuk, H. B. Danilchuk, *Modeling of Complex Systems: A Study Guide for Independent Study of the Discipline* (Cherkasy: Vovchok Publishing House, 2016).
171. W. Chen, Z. Wang, H. Xie, and W. Yu, *Characterization of Surface EMG Signal Based on Fuzzy Entropy*, IEEE Transactions on Neural Systems and Rehabilitation Engineering **15**, 266 (2007).
172. W. Jun, M. Barahona, T. Yue-Jin, and D. Hong-Zhong, *Natural Connectivity of Complex Networks*, Chinese Physics Letters **27**, 078902 (2010).
173. X. Lan, H. Mo, S. Chen, Q. Liu, and Y. Deng, *Fast transformation from time series to visibility graphs*, Chaos: An Interdisciplinary Journal of Nonlinear Science **25**, 083105 (2015).
174. X. Sun, H. Chen, Z. Wu, and Y. Yuan, *Multifractal Analysis of Hang Seng Index in Hong Kong Stock Market*, Physica A: Statistical Mechanics and Its Applications **291**, 553 (2001).
175. Y. Bai, Z. Liang, and X. Li, *A Permutation Lempel-Ziv Complexity Measure for EEG Analysis*, Biomedical Signal Processing and Control **19**, 102 (2015).
176. Y. Holovatch, R. Kenna, S. Thurner, *Complex systems: physics beyond physics*, Eur. J. Phys. **38**, 023002 (2017).
177. Z. Chu, H. Guo, X. Zhou, Y. Wang, F. Yu, H. Chen, W. Xu, X. Lu, Q. Cui, L. Li, J. Zhou, *Data-centric financial large language models*. arXiv preprint arXiv:2310.17784 (2023).
178. Z.-Q. Jiang, W.-J. Xie, and W.-X. Zhou, *Testing the Weak-Form Efficiency of the WTI Crude Oil Futures Market*, Physica A: Statistical Mechanics and Its Applications **405**, 235 (2014).
179. Z.-Q. Jiang, W.-J. Xie, W.-X. Zhou, and D. Sornette, *Multifractal Analysis of Financial Markets: A Review*, Reports on Progress in Physics **82**, 125901 (2019).

Наукове видання

Моніторинг і попередження кризових явищ у складних соціально-економічних системах

Монографія

Англійською мовою

Автори:

СОЛОВЙОВ Володимир Миколайович
БЄЛІНСЬКИЙ Андрій Олександрович
МАТВІЙЧУК Андрій Вікторович

Технічний редактор *Олександр Третяков*

Підп. до друку 25.11.2024. Формат 60×84/16. Папір офсетний.
Гарнітура Times New Roman. Обл.-вид. арк. 22,44. Умов. друк. арк. 19,9.
Наклад 100 прим. Вид. № 24-24.

Видавець Третяков Олександр Миколайович.
Свідоцтво про внесення до Державного реєстру видавців
Серія ДК No 4862 від 11.03.2015 р.
Україна, 18001, м. Черкаси, вул. Слави, 1, к. 24.
Тел.: 067 4701314. E-mail: book_brama@ukr.net