Uniwersytet Jagielloński w Krakowie
Wydział Fizyki, Astronomii i Informatyki Stosowanej

Karolina Góra
Nr albumu: 1143418

Badanie metod wyznaczania wykładnika Hursta

Praca licencjacka
na kierunku Informatyka

Praca wykonana pod kierunkiem
dr hab. Paweł Góra prof. UJ
Wydział Fizyki, Astronomii i Informatyki Stosowanej

Kraków 2020

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Kraków, dnia Podpis autora pracy

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Kraków, dnia Podpis kierującego pracą

Abstract

Implementation of algorithms *Rescaled Range* (R/S), *Detrended Fluctuation Analysis* (DFA) and *Detrended Moving Average* (DMA). Observed how, for different data series, each method creates a straight, which directional factor is a value of Hurst exponent and what possible discrepancies there can appear.

All methods are programmed in Python3, with use of ready numerical library (numpy) for calculations and graphic library (matplotlib) for generating final charts.

Abstrakt

Zaimplementowano algorytmy *Przeskalowanego zasięgu (*R/S), *Zniekształconej analizy fluktuacji* (DFA) i *Zniekształconej średniej ruchomej* (DMA). Przeprowadzono obserwacje, jak dla różnych ciągów danych, poszczególne metody wyznaczają proste, których współczynniki kierunkowe odpowiadają wartości wykładnika Hursta oraz jakie rozbieżności mogą się przy tym pojawić.

Wszystkie metody zostały zaprogramowane w języku Python3, z wykorzystaniem gotowych bibliotek numerycznych (numpy) do obliczeń oraz biblioteki graficznej (matplotlib) do generowania wykresów wynikowych.

Spis treści

1. Wstęp

1.1. O wykładniku Hursta

1.1.1. Geneza

Badanie serii danych zebranych w odstępach czasowych jest kluczową kwestią dla lepszego zrozumienia zjawisk występujących w naturze. Analizujemy wszystko co da się poddać obserwacji, w fizyce, biologii, a w szczególności kwestie finansowe i ekonomiczne. Podstawowym problemem pojawiającym się w kontekście analizy jest to, czy dane mają powiązania między sobą. Istnieje wiele technik, pozwalających na rozstrzygnięcie tego.

Wykładnik Hursta nazwany został po Haroldzie Edwinie Hurstcie, który był głównym członkiem zespołu prowadzącego badania. Można spotkać się z określeniami, że wyznacznik ten, jest ‘indeksem zależności’ lub ‘indeksem długo-zasięgowej (long-range) zależności’. W geometrii fraktalnej wykładnik Hursta jest oznaczany przez H , co jest uhonorowaniem Harolda Hursta oraz Ludwiga Otto Höldera przez Benoit’a Mandelbrota. Jest to oznaczenie bezpośrednio powiązane z wymiarem fraktalnym i mierzy ‘moc’ losowości ciągu. Dzięki temu możemy oszacować prawdopodobieństwo, czy trend danego ciągu danych utrzyma się.

1.1.2. Interpretacja wartości wykładnika

Wykładnik ma wartości prawidłowe, jeżeli znajduje się w zakresie $[0,1]$ i w nim wyznaczamy trzy klasy według których określamy nasz rezultat:

- ➔ $H=0.5$ – biały szum, co oznacza, że proces zachowuje się jak błądzenie losowe i nie posiada ukierunkowanego trendu
- ➔ $0<H<0.5$ – ciąg posiada negatywną korelację. Oznacza to, że dane będą często i szybko zmieniać swój kierunek – gdy ciąg jest rosnący zacznie szybko maleć i odwrotnie.
- ➔ $0.5<H<1$ – ciąg przedstawia długo-zasięgową zależność (Long Range Dependence). Oznacza to, że gdy ciąg ma tendencję wzrostową (lub malejącą), to prawdopodobnie ten trend się utrzyma. Badane zdarzenie potrzebuje silnego bodźca, aby spowodować zmianę trendu.

1.1.3. Metody wyznaczania wykładnika Hursta

Istnieje wiele metod pozwalających na wyprowadzenie wartości wykładnika H . Podstawową metodą jest, wyprowadzona przez samego Hursta, metoda ‘Przeskalowanego zakresu’ (Rescaled Range) w skrócie R/S. W tej pracy skupimy się właśnie na miej, oraz na dwóch od niej pochodnych DFA (Detrended Fluctuation Analysis) i DMA (Detrended Moving Average). Wszystkie te algorytmy odpierają się na podobnej filozofii, dzielenia ciągu badanych danych na

mniejsze i przemieszczaniu się po nich. Celem każdego z algorytmów, jak takie przetworzenie danych, aby po przebadaniu ciągu dla różnych wielkości podciągów, wyniki ułożyły się w linii prostej, której współczynnik kierunkowy będzie odpowiadał wartości naszego wykładnika H. Zależność między współczynnikiem kierunkowym prostej a wykładnikiem Hursta można opisać na dwa sposoby.

1. Na wykresie podwójnie logarytmicznym (log-log) przedstawiany jest wynik działania algorytmu. Prosta, dopasowana do widma mocy naszych danych, ma współczynnik kierunkowy α . Wtedy $H=\frac{1-\alpha}{2}$.
2. Na wykresie log-log przedstawione dane są serią ułamkową i tworzą zależność $F(L)\sim L^{\alpha}$. Wtedy dla ułamkowego szumu Gaussa $H=\alpha$. Z tego systemu odczytywania wykładnika Hursta będziemy korzystać.

1.2. Technologie i narzędzia

1.2.1. Język programowania

Algorytmy, którymi się zajmujemy, są metodami numerycznymi, dlatego przy wyborze języka programowania ważne było, aby radził on sobie z przetwarzaniem dużej ilości danych.

➔ **Python3 -** posiada technologię, pozwalającą na swobodne i łatwe odczytywanie danych tekstowych. Posiada modele do gromadzenia dużych zbiorów danych i wbudowane funkcję do ich analizy. Dobrze zaimplementowane biblioteki numeryczne i graficzne dają możliwość szybkiej implementacji naszych algorytmów i wizualizacji ich wyników.

1.2.2. Biblioteki pomocnicze

W ramach języka Python3 do implementacji naszych metod wykorzystane zostały dwie biblioteki wbudowane:

➔ **numpy** – jest to rozbudowana biblioteka metod numerycznych, zaprojektowana do wykonywania działań w najkorzystniejszej złożoności obliczeniowej. Metody wykorzystane w naszym przypadku:

- **average()** – funkcja zwracająca średnią wartości z tablicy.
- **polyfit()** – funkcja jako argumenty przyjmuje: tablicę indeksów dla danych (Index[]), tablicę danych (Data[]), odpowiadającym indeksom z tablicy pierwszej i stopień równania krzywej, jaka ma zostać dopasowana do podanych punktów (Index[i], Data[i]). Zwraca współczynniki wyżej opisanego równania.
- **log()** – zwraca logarytm z podanego argumentu. Może także przekształcić tablicę z wartościami, a tablicę z logarytmami tych wartości.
- **sqrt()** – zwraca pierwiastek z podanego argumentu.

➔ **pyplot** – jest to biblioteka funkcji do tworzenia wykresów danych. Wykorzystane metody to:

- **scatter()** – tworzy zestawienie punktowe danych
- **title()** – nadaje tytuł wykresu()
- **ylabel()** – nadaje nazwę osi Y
- **xlabel()** – nadaje nazwę osi X
- **text()** – pozwala na dodanie tekstu w polu wykresu. Funkcja ta została wykorzystana do wyświetlania na wykresie współczynnika kierunkowego prostej.
- **plot()** – rysuje na wykresie prostą z punktu a do punktu b .
- **show()** – generuje wykres o określonych wyżej wymienionymi funkcjami parametrach i wyświetla po wywołaniu funkcji.

1.2.3. Środowisko pracy

Jako środowisko pracy wykorzystany został PyCharm. Program jest przystosowany specjalnie do pracy z językiem Python. Pozwala on na estetyczne pisanie kodu i jego automatyczne formatowanie. Co najbardziej pomocne w naszym wypadku, posiada on tryb pracy pozwalający na systematyczny podgląd i zapis generowanych przez nasz program wykresów.

1.3. Dane

1.3.1. Typ danych

Dane, na których możemy wyznaczać współczynnik Hursta, muszą spełniać dwa założenia.

1. Zbiór danych musi być ciągiem, gdzie pomiary zostały wykonane w równych odstępach czasowych.
2. Jeśli to możliwe, dane powinny być ciągiem ułamkowym, dla zwiększenia dokładności wykonywanych obliczeń.

1.3.2. Wybrane zbiory danych

Wszystkie algorytmy, które zostaną przedstawione w drugiej części pracy, zostały wykonane dla różnych zbiorów danych i porównano wyniki, jakie dają poszczególne metody dla tych samych danych. Wybrane dane zapisane zostały w plikach:

➔ nile.txt – dane na temat wylewu rzeki Nil

➔ births.txt – dane na temat zmian ilości narodzin w czasie

➔ temp.txt –

➔ zurich.txt –

➔ assigment4.txt – dane zebrane specjalnie pod badanie algorytmu DFA

2. Implementacje

2.1. Funkcje pomocnicze

2.1.1. prepareData() - przygotowanie tablic z danymi to przetwarzania

Każdy z zaimplementowanych algorytmów korzysta z funkcji przygotowującej dane do przetwarzania.

```
#!/usr/bin/python
# -*- coding: iso-8859-2 -*-


def prepareData(file, start_length, end_length):
    array = []
    indexes = []
    with open(file) as data:
        for line in data:
            i, value = line.split()
            indexes.append(int(i))
            array.append(float(value))

    L = []
    if start_length:
        w = start_length
    else:
        w = len(array)

    while w / 2 > end_length:
        if w % 2 == 0:
            L.append(int(w / 2))
            w = w / 2
        else:
            w -= 1

    return indexes, array, L

```

funkcja przygotowująca dane tekstowe

Funkcja przyjmuje trzy atrybuty:

- ➔ file – nazwa pliku tekstowego, w którym zapisane są dane
- ➔ start_length – maksymalna długość przedziału, na które podzielimy dane
- ➔ end_length – minimalna długość przedziału, na jakie podzielimy dane

i zwraca trzy tablice:

- ➔ indexes[] – tablica indeksów, według których posortowane są badane dane
- ➔ array[] – tablica zebranych danych, które poddamy obserwacji
- ➔ L[] – tablica długości badanych przedziałów

Korzystając z możliwości importowania funkcji między plikami, dzięki *from prepareFiles import* * importujemy wyżej zaimplementowaną funkcję do użytku każdego z naszych algorytmów.

2.2. Algorytm R/S (Rescaled Range)

Algorytm *R/S*, jest to podstawowa metoda wyznaczania wykładnika Hursta. Opiera się ona na podziałach danych na równe części i modyfikacji danych w obrębie tych podziałów. Jak nazwa (*ang. rescaled range*) wskazuje, algorytm bada zachowanie asymptotycznego przeskalowania zakresu jako funkcji na seriach czasowych. Zależność między funkcją R/S, a wykładnikiem H przedstawia wzór:

$$\frac{R(n)}{S(n)} = C\,n^{H} \qquad n \to \infty \qquad C = const.$$

gdzie:

- ➔ **n** – to czas obserwacji (ilość punktów danych z serii czasowej), długość badanego aktualnie przedziału
- ➔ **R(n)** – to zakres pierwszych *n* skumulowanych odchyleń od średniej
- ➔ **S(n)** – jest odchyleniem standardowym danej wielkości (*n*) podziału.

1. Aby rozpocząć procedurę algorytmu, musimy zacząć od przygotowania danych. W tej metodzie potrzebujemy tablicy ze zbiorem danych $X[\,]$ oraz tablicę $L[\,]$ z długościami *n*, dla których będziemy powtarzać algorytm.

```
#0
indexes, X, L = prepareData('nile.txt', 165, 4)        # pobranie danych
N = len(X)                                             # N = ilość badanych danych
AVG = []                                               # tablica, w której zbieramy końcowe wyniki dla każdego n
```

Istotną częścią podzielenia naszych danych na przedziały o długości *n*, jest fakt, że nie mogą one na siebie zachodzić, czyli muszą być rozłączne. (1) Z tak przygotowanymi danymi, wykonujemy kolejne kroki w pętli FOR po tablicy L.

2. (2) Dla każdego przedziału o długości n w serii danych L, obliczamy średnią: $m = \frac{1}{n}\sum_{i=1}^{n} X_i$.
3. (3) Dane w każdym przedziale regulujemy wyliczoną dla niego średnią: $Y_t = X_t - m$.
4. Obliczamy sumę serii z odchyleniem: $Z_t = \sum_{i=1}^{t} Y_i$.
5. Obliczamy standardowe odchylenie S: $S(n) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i - m)^2}$. W tym celu mamy koleją funkcję pomocniczą *cumulativeSum()*, aby kod był bardziej czytelny. Funkcja ta zwraca wartość wyrażenia podpierwiastkowego. Funkcja jako argumenty przyjmuje:
   - ➔ **size** – rozmiar przedziału, czyli size = n.
   - ➔ **array** – tablica o rozmiarze *n*, zawierająca wartości wybranego przedziału z ciągu wszystkich danych.
   - ➔ **m** – (*ang. mean*) średnia, którą obliczyliśmy w kroku ***2***.

```
54  def cumulativeSum(size, array, m):
55      cumulative_sum = 0
56      for w in range(0, size):
57          cumulative_sum += (array[w] - m) * (array[w] - m)       # (10)
58      return cumulative_sum
```

Po tym jak funkcja zostanie wywołana (4) obliczona jest wartość pierwiastka i zapisania jako odchylenie standardowe w tablicy *total_S*.

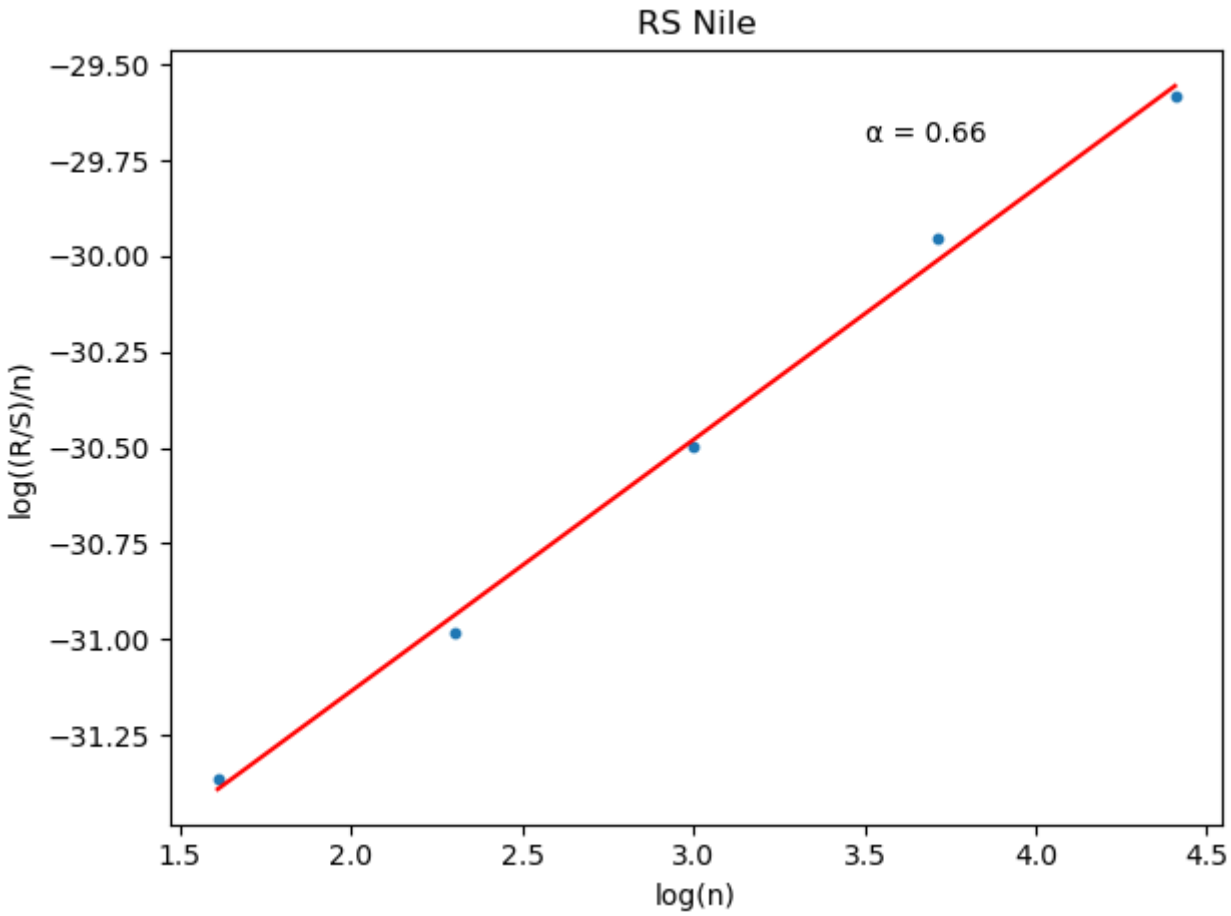
6. Punkty od 2-5 wykonujemy dla wszystkich możliwych przedziałów rozłącznych o długości $n \rightarrow$ czyli wyjdzie ich około $\frac{N}{n}$.
7. Kiedy mamy już pełne tablice Z (z sumami odchyleń wszystkich podziałów o długości n) obliczmy współczynnik R, który równa się największej różnicy odchyleń (6).
8. Następnie dla każdego $S \neq 0$ ze zbioru odchyleń standardowych (7), obliczamy R/S (8) i zapisujemy w tablicy R_S dla danego wymiaru n.
9. Na koniec obliczamy nasz ostateczny wynik dla wymiaru n, który jest równy wartości średniej obliczonych w punkcie wyżej R/S. Wartość tę zapisujemy w tablicy AVG[] (9), na podstawie której później wygenerujemy wyniki.
10. Kroki 2-9 powtarzamy dla każdej wartości *n* z tablicy L.

```
14  for n in L:                                 # (1)
15      R_S = []                                # wartości R/S dla n, wartość R_S[1] odpowiada długości n z L[1]
16      Z = []                                  # tablica sum odchyleń dla wszystkich serii o długości n
17      total_S = []                            # zbiór wartości S dla przedziału o długości n, S[1] -> L[1]
18      i = 0
19      while i <= N - n:                       # (2)
20          segment = X[i:i+n]                  # wybranie kolejnego segmentu o długości n
21          m = (np.average(segment))           # wyliczenie średniej dla wybranego segmentu o długości n
22          Y = []                              # Seria odchyleń dla danego segmentu
23          for s in range(i, i+n):             # (3)
24              Y.append(X[s] - m)
25
26          Z.append(np.sum(Y))                 # zapisanie pełnego odchylenia średniej dla przedziału
27
28          S = cumulativeSum(n, segment, m)    # (4)
29          S = np.sqrt(S/n)                    # Odchylenie standardowe dla wyznaczonego przedziału
30          total_S.append(S)                   # (5)
31
32          i += n                              # wybranie początku następnego przedziału o długości n
33
34      R = max(Z) - min(Z)                     # (6) Największa różnica odchyleń dla wszystkich zbadanych podziałów
35      for s in total_S:                       # (7)
36          if s != 0:
37              R_S.append(R/s)                 # (8) wyznaczenie R/S dla każdego przedziału o długości n
38
39      AVG.append(np.average(R_S))             # (9) zapisanie śreniej ze wszystkich zebranych wartości R_S[n]
```

Aby odczytać z tych obliczeń nasz wykładnik H, tworzymy wykres podwójnie logarytmiczny, dla długości segmentów danych (n) i średniej R/S jaką dla niej uzyskaliśmy.

```
41      plt.scatter(np.log(L), np.log(AVG), s=10)
42      plt.title('RS Nile')
43      plt.ylabel('log((R/S)/n)')
44      plt.xlabel('log(n)')
45      result = np.polyfit(np.log(L), np.log(AVG), 1)
46
47      plt.text(3.5, -29.7, '\u03B1 = {}'.format(round(result[0], 2)))
48      x1 = np.log(L[0])
49      x2 = np.log(L[-1])
50      plt.plot([np.log(L[0]), np.log(L[-1])], [result[0] * x1 + result[1], result[0] * x2 + result[1]], 'red')
51      plt.show()
```

Przykładem takiego rozwiązania może być wykonanie algorytmu dla danych zebranych na temat nilu.



H jest współczynnikiem α prostej jaką tworzą przedstawione na wykresie wartości. Wynosi on ~0.66, czyli należy do zakresu od 0.5 do 1, co pozwala założyć, że w przyszłości trend utrzyma się.

2.3. Algorytm DFA (Detrended Fluctuation Analysis)

Algorytm DFA jest, jak nazwa wskazuje, metodą analizy fluktuacji danych. Kluczowymi momentami w jego procesie są:

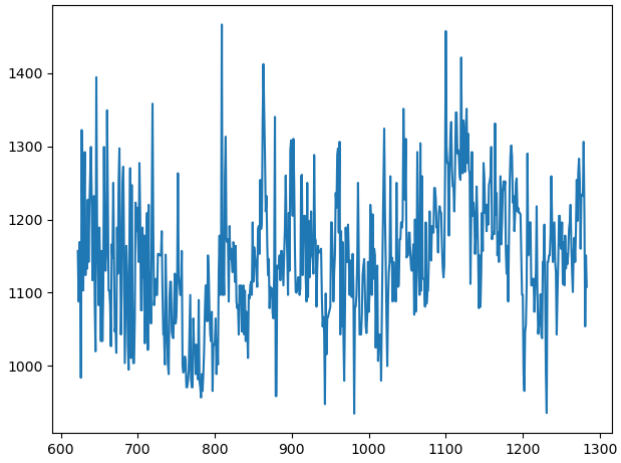
➔ modyfikacja danych, korzystając z tak zwanego Random Walk-u.

➔ wyznaczenie wartości fluktuacji dla podciągów danych i wyliczenie jej średniej.

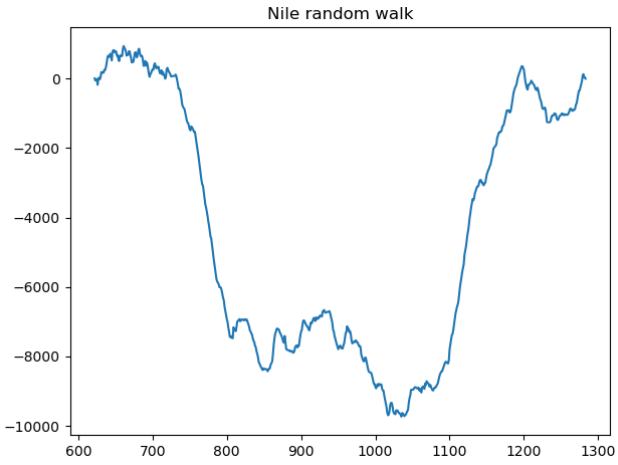
Przygotowanie danych wygląda podobnie jak w algorytmie R/S. Dzielimy zebrane dane na podzbiory o równej długości n, dla różnych n i dla każdego przechodzimy kolejne kroki algorytmu, na koniec zestawiając ze sobą średnie wyników.

```
9   def DFA():
10      # 0
11      indexes, array, L = prepareData('zurich.txt', None, 2)
12      N = len(array)
13
14      # 1 średnia wszystkich danych
15      avg = np.average(array)
```

Do zmiany danych na random walk, potrzebujemy wartości średniej *avg* (1) ze wszystkich pomiarów. Jest to pierwszy krok algorytmu. Random Walk, czyli ruch losowy lub błądzenie losowe, ma pozwolić na uporządkowanie danych, które potrafią być mocno rozproszone i przedstawić je w mniej chaotycznej postaci, która zacznie wskazywać na potencjalne zachowanie trendu. Poniżej zobrazowanie danych nilu w swojej pierwotnej postaci (rys. 1) i te same dane po przerobieniu na Random Walk (rys. 2).



rys 1. Dane oryginalne



rys 2. Dane - random walk

W naszej metodzie, błądzenie losowe wyprowadzimy zamieniając daną na pozycji *i* na daną wyznaczoną na podstawie sumy poprzedzających ją wartości zmodyfikowanych o średnią całego ciągu (wyznaczone powyżej *avg*).

$$X_n = \sum_{k=1}^{n} \left(x_k - \langle x_n \rangle \right) \quad \leftarrow \quad \langle x_n \rangle = avg$$

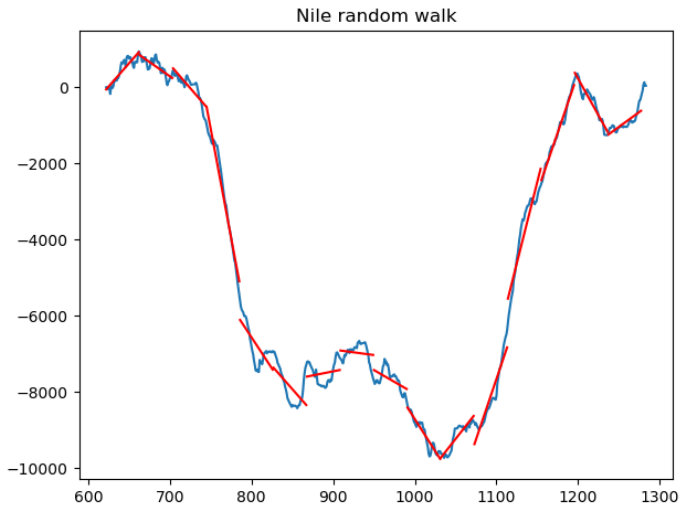
```
    # 2 zmiana danych na random walk
    randomWalk = []
    cumulative_sum = 0.0
    for i in range(0, N):
        cumulative_sum += array[i] - avg
        randomWalk.append(cumulative_sum)
```

Po wykonaniu powyższych działań, otrzymamy dane w postaci random walk'u i na nich dopiero będziemy stosować właściwy algorytm DFA.

```
    # petla do wybierania długości segmentów
    F_avg = []
    for segment_size in L:
        # 3
        temp_array = randomWalk.copy()
        X = indexes.copy()
        i = 0
        k = indexes[0]
        F = []
        while i <= N - segment_size:
            # znalezienie prostej w segmencie: line[0]=a; line[1]=b;
            line = np.polyfit(X[0:segment_size], temp_array[0:segment_size], 1)

            del temp_array[0:segment_size]
            del X[0:segment_size]

            # 4 wyliczenie F
            F.append(calculateF(line, segment_size, i, k, randomWalk))
            k = k + segment_size
            i = i + segment_size

        # 5 obliczenie sredniej fluktuacji dla danej dlugosci segmentu
        F_avg.append(np.average(F))
        print(segment_size, np.average(F))
```

Zaczynamy jak wcześniej, od podzielenia ciągu na równe części (wybrane wielkości n zebrane są w tablicy L[]). Do każdego segmentu o długości n, dopasowujemy prostą.

$$X_n = a \cdot n + b$$



Każda prosta dopasowana do danego segmentu reprezentuje lokalny trend. Na podstawie równań prostych, wyprowadzimy wartość fluktuacji dla danego segmentu o wymiarze n (4).

$$F(L) = \sqrt{\frac{1}{L} \sum_{i=i_0}^{i_0+L-1} (X_i - i \cdot a - b)^2}$$

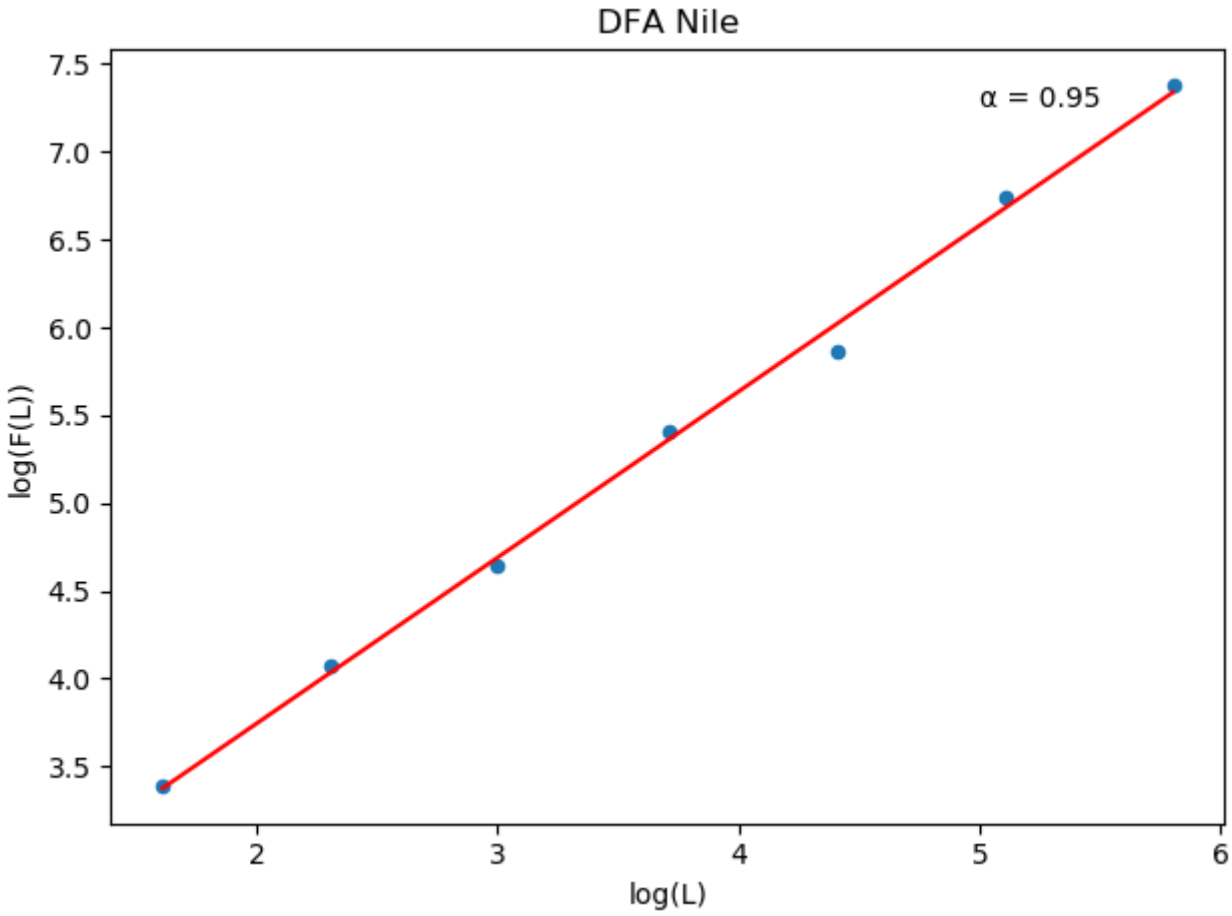
Wartość fluktuacji dla segmentów o danym n zbieramy w tablicy F . Po przejściu przez wszystkie segmenty, wyprowadzamy średnią fluktuację dla podziałów o wymiarze $n \rightarrow \overline{F}(L)$.

$$\overline{F}(L) = np.average(F)$$

W tablicy F_avg zbieramy średnie wartości fluktuacji dla poszczególnych n (5).

Po wykonaniu całego algorytmu dla każdej wartości z tablicy L, tworzymy zestawienie w postaci wykresu podwójnie logarytmicznego (log-log), gdzie przedstawiamy zależność długości (n) segmentów do średniej fluktuacji dla niej obliczonej (6).

```
56      # 6 double logaritmic plot
57      plt.scatter(np.log(L), np.log(F_avg), s=20)
58      plt.title('DFA nile')
59      plt.ylabel('log(F(L))')
60      plt.xlabel('log(L)')
61
62      result = np.polyfit(np.log(L), np.log(F_avg), 1)
63      print('alfa = ', result[0])
64      print(result)
65
66      plt.text(5, 7.25, '\u03B1 = {}'.format(round(result[0], 2)))
67      x1 = np.log(L[0])
68      x2 = np.log(L[-1])
69      plt.plot([np.log(L[0]), np.log(L[-1])], [result[0]*x1+result[1], result[0]*x2+result[1]], 'red')
70      plt.show()
```



Wynik DFA - nile.txt

Wynikiem algorytmu dla danych nilu jest wykres powyżej. Możemy z niego wyczytać, że wykładnik Hursta wynosi 0.95. Oznacza to, że trend danych prawdopodobnie utrzyma się, ponieważ wynik znajduje się w zakresie (0.5, 1].

2.4. Algorytm DMA (Detrended Moving Average)

Literatura

1. WYKORZYSTANIE WYKŁADNIKA HURSTA DO PROGNOZOWANIA ZMIAN CEN NA GIEŁDZIE PAPIERÓW WARTOŚCIOWYCH
2. STATISTICAL PROPERTIES OF OLD AND NEW TECHNIQUES IN DETRENDED ANALYSIS OF TIME SERIES
3. Wikipedia – Hurst Exponent
4. UJ wykłady Paweł Góra
5. Metody Analizy długozasięgwej
6. Detrending Moving Average variance: a derivation of the scaling law