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

Przewidywanie przyszłości zaprzątało umysły i rozbudzało wyobraźnię ludzi od najdawniejszych czasów. Obok poszukiwania kamienia filozoficznego oraz eliksiru młodości było przedmiotem niezliczonych badań i dociekań najbardziej przenikliwych umysłów na przestrzeni dziejów, by wymienić tylko takich gigantów jak Fermat, Pascal, Jakub Bernoulli, Gauss, Laplace czy Boltzmann. Wprawdzie cel nie został do dzisiaj osiągnięty i być może nigdy do tego nie dojdzie, tym niemniej w ostatnich stuleciach dokonano liczących się odkryć oraz odniesiono sporo sukcesów, mających swój bardzo praktyczny wymiar.

Podstawą prowadzenia badań, które mają na celu lepsze poznanie i zrozumienie zjawisk występujących w naturze, jest ich monitorowanie i odnotowywanie wszelkich zmian. Najbardziej powszechną techniką obserwacji jest zapisywanie dokonanych pomiarów w określonych, równych odstępach czasowych. Daje ona możliwość wyciągnięcia wniosków o zachodzących anomaliach jak i powtarzających się prawidłowościach.

Takim analizom człowiek poddaje wszystko, co tylko rozbudza jego ciekawość, poczynając od zjawisk fizycznych, a kończąc na kwestiach finansowych i ekonomicznych. Podstawowym problemem pojawiającym się w kontekście analizy jest to, czy obserwowane zjawisko posiada jakąkolwiek zależność, czy jest całkowicie przypadkowe i w związku z tym nieprzewidywalne. W takim wypadku nie ważne, jak dokładnie zbierane są dane, jeśli nie posiadają powiązania między sobą, nie pozwolą nam na wyciągnięcie wniosków i przewidzenia jak mogą się zachować w przyszłości. Na szczęście istnieje wiele technik umożliwiających rozstrzygnięcie tego problemu.

Jedną z metod służących do określania, czy w danym ciągu czasowym istnieje jakakolwiek zależność, jest metoda wyznaczania wykładnika Hursta. Badania z jego wykorzystaniem pierwotnie pojawiły się w dziedzinie hydrologii, kiedy przez długi czas zbierano dane w celu określania optymalnych wymiarów tamy do warunków niestabilnego deszczu i suszy na Nilu. Co więcej, przedsięwzięcie to udowodniło, że zmiany poziomu wody nie są zjawiskiem całkowicie losowym, oraz odkryto istnienie długoterminowej pamięci zdarzeń dla nieskończenie długich szeregów czasowych. Warto nadmienić, że do wykonywanych obliczeń Harold Edwin Hurst miał do dyspozycji dane z 847 lat. Wykładnik swoją nazwę otrzymał po wyżej wspomnianym hydrologu brytyjskim, który był głównym członkiem zespołu prowadzącego owe badania. Można spotkać się z określeniami, że wykładnik ten, jest ‘indeksem zależności’ lub ‘indeksem długo-zasięgowej (long-

range) zależności'. W geometrii fraktalnej wykładnik Hursta jest oznaczany przez symbol **H**, co jest uhonorowaniem Harolda Hursta oraz Ludwiga Otto Höldera przez Benoit'a Mandelbrota.

W geometrii fraktalnej wykładnik H mierzy *moc łagodnej* lub *dzikiej* losowości serii danych. Dzięki niemu możemy oszacować prawdopodobieństwo, czy trend obserwowanego zjawiska zachowa się – czy szereg czasowy posiada tendencję do silnej regresji do średniej, czy jednak dąży w określonym kierunku.

1.1.2. Interpretacja wartości wykładnika

Podczas badań nad wylewami Nilu, dostrzeżono potęgową zależność pomiędzy odchyleniem standardowym σ , a długością analizowanego szeregu czasowego *n,* która wyznacza omawiany wykładnik Hursta H.

$$\sigma = n^H$$

Z tej zależności wywodzi się koncepcja przedstawiania wyników na wykresach podwójnie logarytmicznych. „Obkładając" logarytmem obie strony równania $\ln(\sigma) = \ln(n^H)$, zbliżamy się do jego przekształcenia do pożądanej postaci, którą jest równanie prostej $\ln(\sigma) = H \cdot \ln(n) + const$ Taki zapis równania pozwala na zlokalizowanie regresji liniowej badanych danych, której współczynnik kierunkowym jest szukanym wykładnikiem Hursta H.

W praktyce zależność pomiędzy wyznacznikiem, a współczynnikiem kierunkowym prostej można zdefiniować na dwa sposoby:

1. Przedstawiając widmo mocy badanego ciągu na wykresie podwójnie logarytmicznym i dopasowanie do niego prostej. Wtedy $H = \frac{1-\alpha}{2}$, gdzie α to współczynnik kierunkowy.
2. Przedstawiane na wykresie podwójnie logarytmicznym dane są serią ułamkową i tworzą zależność $F(L) \sim L^{\alpha}$, gdzie $H = \alpha$.

Jak łatwo zauważyć, wywodząca się z algorytmu R/S zależność, odpowiada drugiemu z opisanych sposobów i to właśnie nim zostaną odczytane wszystkie współczynniki H, wyliczane na potrzeby tej pracy.

Wyznaczając wykładnik H oczekujemy, że jego wartość znajdzie się w zakresie od 0 do 1. Wynika to z własności prawdopodobieństwa, które maksymalnie może wynosić 1. Uzyskanie wartości brzegowych tj. 0, 0.5, 1 jest rzadko spotykane, przez co często uważane są za czysto teoretyczne.

Niektóre metody wymagają przemyślanego dopasowania okresów obserwacji, które mogą zależeć od algorytmu, jak i od rodzaju czy ilości zgromadzonych danych. Manipulując zmiennymi odpowiadającymi za te czynniki, możemy dostosować układ naszych danych końcowych do najbardziej pożądanego rezultatu.

Ostatecznie, po wyliczeniu indeksu zależności, trzeba go zinterpretować. Wyróżniamy trzy konkluzje, do jakich doprowadza nas wartość wykładnika H:

- ➔ $H=0.5$ – szereg nieskorelowany, inaczej: dane tworzą biały szum. Oznacza to, że proces zachowuje się jak błądzenie losowe i nie posiada ukierunkowanego trendu.
- ➔ $0<H<0.5$ – korelacja negatywna – szereg czasowy, w którym przez długi czas dane będą często i szybko zmieniać swój kierunek – po wartości wysokiej prawdopodobnie nastąpi wartość niska, a po niej znów wysoka z tą samą tendencją do przełączania się między wartościami wysokimi i niskimi.
- ➔ $0.5<H<1$ – długoterminowa dodatnia autokorelacja. Ciąg przedstawia długo-zasięgową zależność (Long Range Dependence), to znaczy, że gdy ciąg ma tendencję wzrostową (lub malejącą), to prawdopodobnie ten trend się utrzyma; po wysokiej wartości prawdopodobnie pojawi się kolejna wysoka wartość. 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. Dobiera się je w zależności, jakie szeregi poddaje się badaniom. Do szeregów czasowych jednowymiarowych, do których ograniczona została niniejsza praca, można zastosować algorytmy takie jak:

- ➔ R/S – metoda analizy długozasięgowej (ang. Rescaled Range);
- ➔ DFA – metoda analizy odtrędowionych fluktuacji (Detrended Fluctuation Analysis);
- ➔ DMA – metoda odtrędowionej średniej kroczącej (Detrended Moving Average).

Pierwsza z wymienionych metod – R/S, jest pierwotną metodą estymacji wykładnika Hursta, opisana przez samego Harolda. Została zdefiniowana na zasadach asymptotycznego zachowania przedziałów o długim zasięgu, jako funkcja przedziałów czasowych w szeregu czasowym:

$$\frac{R(n)}{S(n)} = C \cdot n^H \quad przy \quad n \to \infty$$

gdzie:

- ➔ R(n) – jest to *zasięg* pierwszych *n* złożonych *odchyleń* od średniej;
- ➔ S(n) – jest ich odchyleniem standardowym;
- ➔ n – wielkość przedziału czasowego, segmentu, na którym przeprowadzane są obserwacje;
- ➔ C – pewna stała.

Każdy z wyżej wspomnianych algorytmów opiera się na wielokrotnym przebadaniu serii danych, dla przedziałów o różnej wielkości *n*. Celem użytych w nich metod jest zastosowanie, opisanej w podrozdziale wcześniej, zależności potęgowej między długością serii danych, a ich

odchyleniem standardowym. Można powiedzieć, że algorytm R/S przedstawia ów relację w najbardziej dosłowny sposób, co widać na zamieszczonym powyżej opisie.

Odmiennie, metoda DFA dochodzi do pożądanej zależności skupiając się na fluktuacji trendu danych, zamiast na zasięgu sygnałów. Z tego powodu może być stosowania nie tylko do wyznaczania wykładnika Hursta, ale także posiada szersze zastosowanie, przy badaniu sygnałów, których podstawowe statystyki (np. średnia, wariacja) lub dynamika zmieniają się w czasie. Jest ona rozszerzeniem zwykłej analizy fluktuacji (FA) i została wprowadzona przez fizyka Chung-Kang Peng w 1994 roku. Aby wyznaczyć wykładnik, należy wyprowadzić średnią wartość fluktuacji, dla różnych wielkości przedziałów czasowych. Dzięki pozyskanemu zbiorowi par – długość przedziału i jego średnia fluktuacja - odnajdywana jest interesująca badacza korelacja.

Ostatnia z metod, metoda DMA, jest algorytmem najmniej skomplikowanym. Opiera się ona bowiem na doborze przedziału – okna badawczego – i przemieszczeniu się nim, jak suwakiem, po badanych danych. Taką operacją uzyskuje się tytułową średnią kroczącą, dla okna o wybranej wielkości n. Następnie modyfikuje się wartości szeregu czasowego, o wyznaczone średnie. Na podstawie powstałego zmodyfikowanego ciągu, odnajduje się szukaną zależność, pomiędzy długością okna czasowego, a średnią wartości się w nim znajdujących.

Oprócz wspomnianych już metod, do wyznaczania wykładnika H, służą także algorytmy falkowe. Jest ich bardzo wiele, żeby nadmienić kilka z nich takich jak falka Haar’a, Daubechie rzędu 4, Coiflet’a rzędu 1 czy Symlet rzędu 10. Bazują one na Dyskretnej transformacji Falkowej oraz wyżej określonych bazach falkowych. Celem tych algorytmów jest wyznaczenie tzw. widma falkowego i dopasowania do niego prostej, która spełnia zadanie takie samo jak w opisanych na początku metodach.

1.2. Wyjaśnienie pojęć

1.2.1. Korelacja

Korelacja, jest to statystyczna zależność zmiennych losowych. Przedstawia ona związek pomiędzy dwiema zmiennymi losowymi X i Y. Podchodząc do zagadnienia intuicyjnie, można powiedzieć, iż oznacza to, że znając wartość jednej z nich, powinno być się w stanie określić (przynajmniej w niektórych przypadkach) dokładną lub przybliżoną wartość tej drugiej.

W statystyce, analiza korelacji polega na zbadaniu, czy dwie zmienne są ze sobą istotnie powiązane. Doszukuje się współzależności pomiędzy dowolnymi dwoma cechami, atrybutami czy własnościami. To dlatego wspomniane wcześniej trzy metody estymacji wykładnika Hursta, mimo że każda opiera się na badaniu innej własności wartości w szeregu, pozwalają na otrzymanie tego samego wniosku.

Celem analizy korelacji jest dowiedzenie, czy zachodzi związek pomiędzy dwoma zmiennymi, a nie doszukuje się przyczyny, dla której wartości opisujące zdarzenie zachowują się w zaobserwowany sposób.

Analizując korelację, badacza interesuje odpowiedzenie sobie na trzy pytania:

1. Czy w badanym zjawisku występuje związek?
2. Jaki jest współczynnik korelacji? W odniesieniu do niniejszej pracy, jest on równy wykładnikowi Hursta. Co za tym idzie, jeśli znajduje się w zakresie od 0 do 0.5, jest to korelacja negatywna, a jeśli w zakresie od 0.5 do 1, jest to korelacja pozytywna.
3. Jak silny jest związek pomiędzy badanymi zmiennymi?

Wynik analizy korelacji graficznie prezentuje się na tak zwanym wykresie rozrzutu. W przypadku wykładnika Hursta, jest to wcześniej już wspomniany, wykres podwójnie logarytmiczny, obrazujący zależność między długością przedziału czasowego, a badaną własnością szeregu.

1.2.2. Odchylenie standardowe

Odchylenie standardowe (σ) jest kolejnym pojęciem z dziedziny statystyki. Zaliczamy je do miar rozproszenia, przeznaczonych do badania stopnia zróżnicowania wartości zmiennej. Jej zadaniem jest określenie, o ile średnio, wartości w badanym szeregu czasowym, odchylają się od średniej arytmetycznej badanej zmiennej.

Niskie wartości σ świadczą o małym rozproszeniu danych, czyli ich niewielkim zróżnicowaniu. Wysokie wartości odchylenia przeciwnie, oznaczają silne rozproszenie.

Biorąc za przykład metodę R/S, która wprost opiera się na wyliczaniu odchylenia standardowego dla badanych przedziałów czasowych, obserwujemy bezpośrednią zależność pomiędzy wyżej opisanym rozproszeniem wartości, a korelacją jaką na ich podstawie analizujemy.

1.2.3. Random Walk

Błądzenie losowe (ang. Random Walk) jest pojęciem z zakresu matematyki i fizyki, określającym ruch losowy. Oznacza to, że w kolejnych momentach czasu, cząstka (obserwowany obiekt) przemieszcza się z aktualnego położenia, do innego, losowo wybranego.

W niniejszej pracy, błądzenie losowe będzie wykorzystane w celu uporządkowana chaotycznych danych, przed rozpoczęciem ich przetwarzania. Algorytm pozwalający na transformację pierwotnego szeregu danych na Random Walk, będzie opierał się na zamienieniu każdej wartości, na sumę wartości ja poprzedzających, zmodyfikowanych o średnią arytmetyczną całej serii czasowej. Zapis tej operacji przedstawia równanie $X_n = \sum_{k=1}^{N} \left(x_k - \langle x_n \rangle \right)$, gdzie:

- X_n - zamieniamy element serii czasowej
- N - długość szeregu czasowego
- x_k - k-ta wartość szeregu czasowego

➔ $\langle x_n \rangle$ - średnia arytmetyczna z całej serii czasowej

1.2.4. Fluktuacja

Fluktuacja, inaczej wahania przypadkowe, są to niedające się przewidzieć, odchylenia od wartości średniej zmiennej losowej, podlegającej losowym zmianom w czasie i nie wykazujące żadnej tendencji. Fluktuacje są ściśle związane z błędami statystycznymi oraz prognostycznymi. Pojawiają się jako składowa szeregu czasowego (trend + okresowość + wahania przypadkowe). Dla wielkości proporcjonalnych do liczby N serii danych, rozproszenie danej wielkości A(t) związane z fluktuacjami $\sigma^2(A)=\langle A^2\rangle-\langle A\rangle^2$ jest proporcjonalne do N. Z tego wynika względna fluktuacja: $\frac{\sigma(A)}{A}=\frac{\sqrt{N}}{N}=\frac{1}{\sqrt{N}}$. Taką zależność zaobserwować będzie można w rozdziale 2, podczas opracowywania metody DFA, która oblicza fluktuację danych w celu wyprowadzenia wykładnika Hursta.

1.3. Technologie i narzędzia

1.3.1. Język programowania

Metody numeryczne wymagają zadbania o prędkość wykonania, przejrzystość zapisu i ograniczenie ich złożoności do minimum. Jako, że praca opiera się na badaniu danych, istotne jest efektywne czytanie plików, w których zawarte są tysiące rekordów do przetworzenia. Ponad to, potrzebna jest możliwość zobrazowania osiągniętych rezultatów. To wszystko zapewnia język Python.

Python3 - posiada technologię, pozwalającą na swobodne i łatwe odczytywanie danych tekstowych. Pozwala na modelowanie dużych zbiorów danych i oferuje wbudowane funkcje do ich analizy. Dobrze zaimplementowane biblioteki numeryczne (numpy) i graficzne (pyplot), dają możliwość szybkiej i wydajnej implementacji algorytmów oraz wizualizacji ich wyników.

1.3.2. Biblioteki pomocnicze

Korzystając z Python3, do implementacji opisywanych w pracy metod, dla poprawności obliczeń i usprawnienia wizualizacji danych, użyte zostały dwie z wbudowanych bibliotek:

➔ **numpy** – rozbudowana biblioteka służąca do usprawnienia obliczeń numerycznych i nadania im najmniejszej możliwej złożoności obliczeniowej. Funkcje wykorzystane do implementacji omawianych metod, to:

- **average()** – funkcja zwraca średnią z wartości przekazanych w tablicy;
- **polyfit()** – funkcja zwraca współczynniki równania krzywej, o pożądanym stopniu, na podstawie przekazanych do niej tablicy z indeksami X oraz tablicy z wartościami Y,

odpowiadającymi kolejno indeksom z X. Wykorzystywana do wyprowadzania regresji liniowej dla wartości wynikowych opracowywanych algorytmów;

- **log()** – funkcja zwraca logarytm naturalny z podanego argumentu. Może także wykonać mapowanie, przekształcając tablicę z wartościami, na tablicę z logarytmami tych wartości;
- **sqrt()** – funkcja zwraca pierwiastek z podanego argumentu.

➔ **pyplot** – biblioteka funkcji do tworzenia wykresów danych. Wykorzystane zostały funkcje:

- **scatter()** – tworzy punktowe wykresy 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 współczynnika kierunkowego α 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.

1.3.3. Środowisko pracy

Jako środowisko pracy wykorzystane zostały JetBrains PyCharm oraz Visual Studio Code.

Program PyCharm jest przystosowany specjalnie do pracy z językiem Python oraz jego bibliotekami. Pozwala on na estetyczne pisanie kodu, jego automatyczne formatowanie i wykrywanie błędów. Posiada wygodny tryb pracy, umożliwiający systematyczny podgląd i zapis generowanych przez program wykresów.

1.4. Dane

1.4.1. Typ danych

Tak jak zostało wcześniej wyjaśnione, analizy przeprowadzane kluczowymi dla tej pracy algorytmami, odnoszą się do szeregów czasowych jednowymiarowych. Taki szereg musi spełniać jeden kluczowy warunek:

➔ seria danych musi być szeregiem, gdzie pomiary zostały wykonane w równych odstępach czasowych.

Dodatkowo. jeśli jest to możliwe, powinien być to szereg ułamkowy, dla zwiększenia dokładności wykonywanych obliczeń.

1.4.2. Wybrane zbiory danych

Wszystkie metody estymacji współczynnika Hursta, przedstawione w drugiej części pracy, zostały wykonane dla różnych serii danych. W części trzeciej przedstawiono i porównano uzyskane wyniki. Wybrane dane wejściowe zapisane zostały w plikach:

- ➔ **nile.txt** – dane na temat wylewu rzeki Nil;
- ➔ **births.txt** – liczba urodzeń (dziennie) dziewczynek w roku 1959 w Californii;
- ➔ **temp.txt** – minimalna dzienna temperatura w Melbourne (lata 1981-1990);
- ➔ **zurich.txt** – liczba zaobserwowanych plam słonecznych w danym miesiącu w Zurichu (lata 1749- 1983);
- ➔ **assigment4.txt** – dane przygotowane specjalnie do algorytmu DFA, pobrane ze źródła 4;
- ➔ **nino.txt** – temperatury powierzchni morza (264 rekordy);
- ➔ **warner.txt, mishmash.txt** – dane oferowane przez bibliotekę python – *pywelvets* dotyczącą zagadnień falek.

2. Implementacje

2.1. Funkcje pomocnicze

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

W pliku o nazwie *prepareData.py* znajduje się funkcja, przygotowująca do przetwarzania dane, z podanego źródła. Każdy zapisany rekord składać musi się z dwóch wartości. Jedna jest wartością porządkującą, a druga zapisaną obserwacją, która zostanie poddana analizie. Każdy rekord zapisany jest w nowej linii, a jego dane oddzielone spacją.

```
5  def prepareData(file, max_n, min_n):
6      X = []
7      indexes = []
8      with open(file) as data:
9          for line in data:
10             i, value = line.split()
11             indexes.append(int(i))
12             X.append(float(value))
13
14     L = []
15     if max_n:
16         w = max_n
17     else:
18         w = len(X)
19
20     while w / 2 > min_n:
21         if w % 2 == 0:
22             L.append(int(w / 2))
23             w = w / 2
24         else:
25             w -= 1
26
27     return indexes, X, L
```

funkcja przygotowująca dane tekstowe do przetwarzania przez algorytmy

Funkcja przyjmuje trzy argumenty:

- ➔ file – nazwa pliku tekstowego, w którym zapisane są dane;
- ➔ max_n – maksymalna długość przedziałów (n), na jakie dzielone są dane;
- ➔ min_n – minimalna długość przedziałów (n), na jakie dzielone są dane;

i zwraca trzy tablice:

- ➔ indexes – tablica indeksów pobranych z przekazanego pliku. Jej wartości świadczą o tym, czy dane zostały zebrane w równych odstępach czasowych i szereg spełnia założenie z punktu 1.4.1;

- ➔ X – tablica zgromadzonych danych, które poddane zostaną obserwacji. Kolejność danych w tablicy X, jest w ścisłym związku z kolejnością danych tablicy *indexes*. Argumentowi *indexes[i]* odpowiada wartość *X[i]* i zależność ta nie może zostać zaburzona;
- ➔ L – tablica wybranych długości (n) badanych przedziałów, tak że ich ilość w przetwarzanej serii danych, jest potęgą 2.

Korzystając z możliwości importowania funkcji między plikami (*from prepareData import **), udostępniono wyżej zaimplementowaną funkcję każdemu zaimplementowanych algorytmów.

2.2. Algorytm R/S (Rescaled Range)

Algorytm *R/S*, jest podstawową metodą wyznaczania wykładnika Hursta. Wymaga ona dużej ilości zebranych rekordów – Hurst przeprowadzał analizę na podstawie danych z kilkuset lat i mógł wykonywać obliczenia dla dużych *n*. Przy mniej licznych seriach, algorytm może nie działać zgodnie z oczekiwaniami.

Metoda R/S opiera się na dzieleniu szeregu czasowego na równe przedziały (ang. ranges) o długości *n* i modyfikacji danych w ich obrębie. Jak wspomniane zostało już w rozdziale 1.1.3, algorytm ten analizuje zależność pomiędzy wielkościami badanych przedziałów czasowych, a odchyleniem standardowym wartości się w nim znajdujących. Przypominając, powiązanie z wykładnikiem H przedstawia równanie:

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

gdzie:

- ➔ **n** – wielkość przedziałów (ranges) czasowych, na których przeprowadzane są obserwacje (ilość *danych* w nich zawartych);
- ➔ **R(n)** – jest to *zasięg* pierwszych *n* złożonych *odchyleń* od średniej;
- ➔ **S(n)** – jest ich odchyleniem standardowym.

1. Algorytm wymaga odpowiedniego przygotowania danych. Korzystając z opisanej w rozdziale 2.1.1 funkcji *prepareData*, otrzymano tablicę danych *X* oraz tablicę *L* z wielkościami przedziałów, dla których zostanie przeprowadzona analiza. W metodzie R/S indeksowanie danych jest zbędnie, ale nie oznacza to, że możemy zaburzyć kolejność badanych wartości X.

```
9    #0
10   indexes, X, L = prepareData('nile.txt', None, 2)        # pobranie danych
11   N = len(X)                                               # N = ilość badanych danych
12   AVG = []                                                 # tablica, w której zbieramy końcowe wyniki dla każdego n
```

Implementacja deklaracji danych wyjściowych

Dla zilustrowania działania algorytmu, wykorzystano szereg czasowy z obserwacjami na temat wylewów Nilu. Dane wyjściowe przedstawiają się w następującej postaci:

- ➔ $X=[1157.0, 1088.0, 1169.0, 1169.0, 984.0, \ \ldots, \ 1205.0, 1054.0, 1151.0, 1108.0]$ - tablica o długości $N=662$ pomiarów, oraz
- ➔ tablica $L=[331, 165, 82, 41, 20, 10, 5]$ z wybranymi długościami *n* segmentów.

Istotną własnością wybranych przedziałów z serii, jest ich rozłączność, czyli kolejne przedziały nie mogą nachodzić na siebie nawzajem.

Przygotowane w powyższy sposób dane, wykorzystano następnie do wykonania kolejnych kroków obliczeniowych.

```
13    for n in L:                                   # (1)
14        R_S = []                                  # wartości R/S dla serii o długości n
15        R = []                                    # zbiór największych różnic odchyleń w przedziałach długości n
16        S = []                                    # zbiór wartości odchyleń standardowych dla przedziałów o długości n
17        i = 0
18        while i <= N - n:                         # (2)
19            segment = X[i:i+n]                    # wybranie kolejnego segmentu o długości n
20            m = np.average(segment)               # wyliczenie średniej dla wybranego segmentu o długości n
21            Y = []                                # Seria odchyleń dla danego segmentu
22            Z = []                                # tablica sum odchyleń dla wszystkich serii o długości n
23            for s in range(i, i+n):               # (3)
24                Y.append(X[s] - m)
25                Z.append(np.sum(Y))               # zapisanie pełnego odchylenia średniej dla przedziału
26
27            R.append(max(Z) - min(Z))             # (6) Największa rónica odchyleń dla zbadanego podziału
28            S.append(satndardDeviation(n, Y))     # (4) Odchylenie standardowe dla wyznaczonego przedziału
29                                                  # (5)
30            i += n                                # wybranie początku następnego przedziału o długości n
31
32        for r, s in zip(R, S):                    # (7)
33            if s != 0:
34                R_S.append(r/s)                   # (8) wyznaczenie R/S dla każdego przedziału o długości n
35
36        AVG.append(np.average(R_S))               # (9) zapisanie średniej ze wszystkich zebranych wartości R_S[n]
```

2. Dla każdego (2) przedziału o długości n, w serii danych X, obliczono średnią arytmetyczną: $m=\frac{1}{n}\sum_{i=1}^{n} X_i$. Przykładowo, dla segmentów o długości $n=82$ średnia dla pierwszego n danych, będzie wynosić $m=1153.62$ *.
3. W kolejnym kroku wykonano przeskalowanie, gdzie dane w każdym przedziale o długości *n* skorygowano o wyliczoną dla niego średnią: $Y_t=X_t-m$ (3). Otrzymano tym sposobem nowy szereg czasowy. Dla pierwszych 82 wartości serii danych X, powstał nowy przedział:
   $Y=[32.05, -36.95, 44.05, \ \ldots, \ -43.95, 71.05, 71.05]$ *.
4. Na podstawie zbudowanej w kroku 3 tablicy *Y*, stworzono serię sum odchyleń Z, zgodnie z równaniem: $Z_t=\sum_{i=1}^{t} Y_i$. Oznacza ono, że każdemu elementowi z_t, przypisuje się sumę wartości tablicy Y od y_0 do y_t. Oba szeregi mają długość *n* oraz $y_0 = z_0$. Odnosząc się do przykładu z danymi dla Nilu, przy $n=82$ szereg sum odchyleń dla pierwszych *n* elementów badanego szeregu czasowego składa się z wartości:
   $Z=[32.05, -4.90, 39.15, \ \ldots, \ -142.10, -71.05, 4.55\text{e-}13]$ *.

5. W kolejnym etapie wyznaczone zostało odchylenie standardowe S: $S(n)=\sqrt{\frac{1}{n}\sum_{i=1}^{n} Y_i^2}$. W celu zachowania przejrzystości kodu, zaimplementowano dodatkową funkcję pomocniczą: *standardDeviation (10)*. Funkcja ta zwraca wartość odchylenia standardowego dla badanego przedziału. Jako argumenty przyjmuje:
   - ➔ **n** – rozmiar przedziału.
   - ➔ **Y** – tablica o rozmiarze *n*, zawierająca skorygowane w kroku 3 wartości serii danych X.

```
52  def satndardDeviation(n, Y):
53      cumulative_sum = 0
54      for i in range(0, n):
55          cumulative_sum += Y[i] * Y[i]                # (10)
56      return np.sqrt(cumulative_sum / n)
```

Implementacja funkcji standardDeviation

Po wywołaniu funkcji (4), zwracana przez nią wartość zapisywana jest w tabeli S, jako odchylenie standardowe badanego segmentu danych. Dla analizowanego przykładu $S_{82}[0]=90.64$ *

6. Mając wyliczone wartości S dla wszystkich przedziałów o długości *n*, do obliczenia lewej strony równania R/S brakuje już tylko wartości R, będącej największą różnicy sum odchyleń w badanym segmencie: $R=max(Z)-min(Z)$. Do jej wyliczenia służy tablica Z, zbudowana w kroku 4. Uzyskaną wartość R zapisuje się do tablicy R, ze zbiorem różnic odchyleń w segmentach dla analizowanego *n*. Dla danych o Nilu $R_{82}[0]=1077.77$
7. Procedura przedstawiona w punktach od 2 do 6 jest powtarzana dla wszystkich kolejnych, rozłącznych przedziałów o długości *n*, czyli dla łącznej liczby segmentów wynoszącej $\lfloor\frac{N}{n}\rfloor$. Poniższa tabela zawiera zestawienie wyników będących rezultatem wykonania powyższych kroków dla wszystkich przedziałów szeregu czasowego X, o $n=82$:

n = 82				
i	Badany zakres	Średnia przedziału [m] *	Największa różnica sum odchyleń w przedziale - $R_n[i]$*	Odchylenie standardowe - $S_n[i]$*
0	0 - 81	1153.62	1077.77	90.64
1	82 - 163	1073.99	2033.66	79.13
2	164 - 245	1129.02	1541.54	89.34
3	246 - 327	1141.97	1502.19	75.34
4	328 - 409	1125.08	1288.69	77.20
5	410 - 491	1190.05	1389.27	74.56

6	492 - 573	1226.57	1481.23	64.48
7	574 - 655	1143.27	64.48	69.20

Tabela 1: Wyniki obliczeń dla danych o Nilu, przy badaniu przedziałów o długości 82 rekordów

8. Dla każdej pary R_i i S_i, obliczona następnie została wartość R/S (8) i zapisana w tablicy R_S, dla danego wymiaru n. Wartości R/S dla poszczególnych segmentów analizowanego przykładu zestawiono w tabeli 2:

n = 82				
i	Badany zakres	Największa różnica sum odchyleń w przedziale - $R_n[i]^*$	Odchylenie standardowe - $S_n[i]^*$	$\frac{R_n[i]}{S_n[i]}^*$
0	0 - 81	1077.77	90.64	11.89
1	82 - 163	2033.66	79.12	25.70
2	164 - 245	1541.54	89.34	17.25
3	246 - 327	1502.19	75.34	19.94
4	328 - 409	1288.69	77.20	16.69
5	410 - 491	1389.27	74.55	18.63
6	492 - 573	1481.23	64.47	22.97
7	574 - 655	64.48	69.20	20.89

Tabela 2: Wynik obliczeń wartości RS dla segmentów długości 82

9. W ostatnim etapie dotyczącym obliczeń, wyliczona została średnia wartość R/S z przedziałów o tej samej długości. Otrzymano tym samym końcową wartość $(\frac{R}{S})_n$, którą zapisano w tablicy wynikowej AVG (9).
10. Kroki od 2 do 9 powtórzone zostały dla każdej kolejnej wartości *n* z tablicy L. Ostatecznie otrzymamy tablicę wynikową AVG o równej długości co tablica L i wiążącą je zależnością $AVG_j = C \cdot L_j^H \quad C = const$. W tabeli 3 przedstawiono końcowe średnie wartości R/S dla poszczególnych wielkości *n*:

X		
j	$L_j = n$	$\frac{R/S}{n} = AVG_j$ *
0	331	72.89
1	165	41.37
2	82	19.25
3	41	11.02
4	20	5.92
5	10	3.41
6	5	1.99

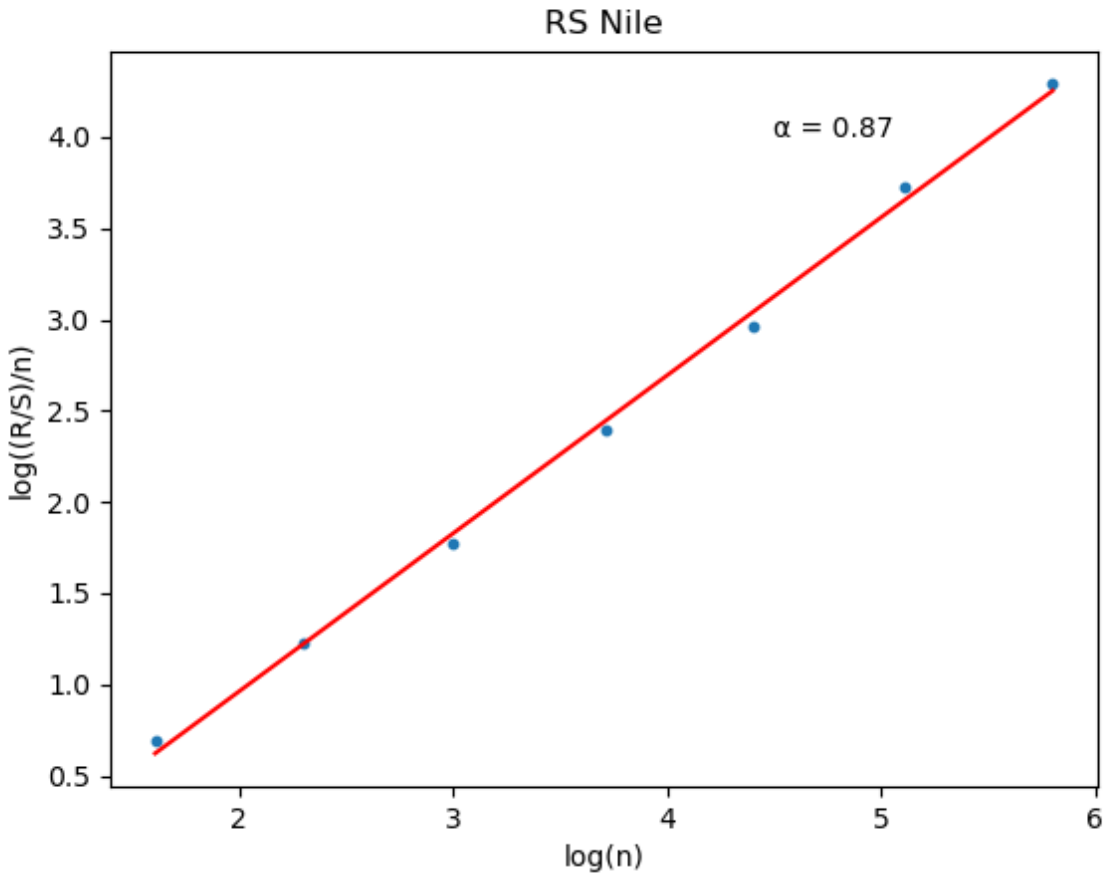
Tabela 3: Zestawienie danych końcowych agorytmu, na podstawie których odczytujemy wniosek

Aby odczytać z wykonanych obliczeń nasz wykładnik H, tworzymy wykres podwójnie logarytmiczny, dla długości segmentów danych (n) i średniej R/S jaką dla niej uzyskaliśmy. Poniżej przedstawiony fragment kodu odpowiada za generowanie wizualizacji danych przy pomocy biblioteki pyplot.

```
38    plt.scatter(np.log(L), np.log(AVG), s=10)
39    plt.title('RS Nile')
40    plt.ylabel('log((R/S)/n)')
41    plt.xlabel('log(n)')
42    result = np.polyfit(np.log(L), np.log(AVG), 1)
43
44    plt.text(4.5, 4, '\u03B1 = {}'.format(round(result[0], 2)))
45    x1 = np.log(L[0])
46    x2 = np.log(L[-1])
47    plt.plot([np.log(L[0]), np.log(L[-1])], [result[0] * x1 + result[1], result[0] * x2 + result[1]], 'red')
48    plt.show()
```

Implementacja generowania wizualnego przedstawienia wyniku algorytmy RS

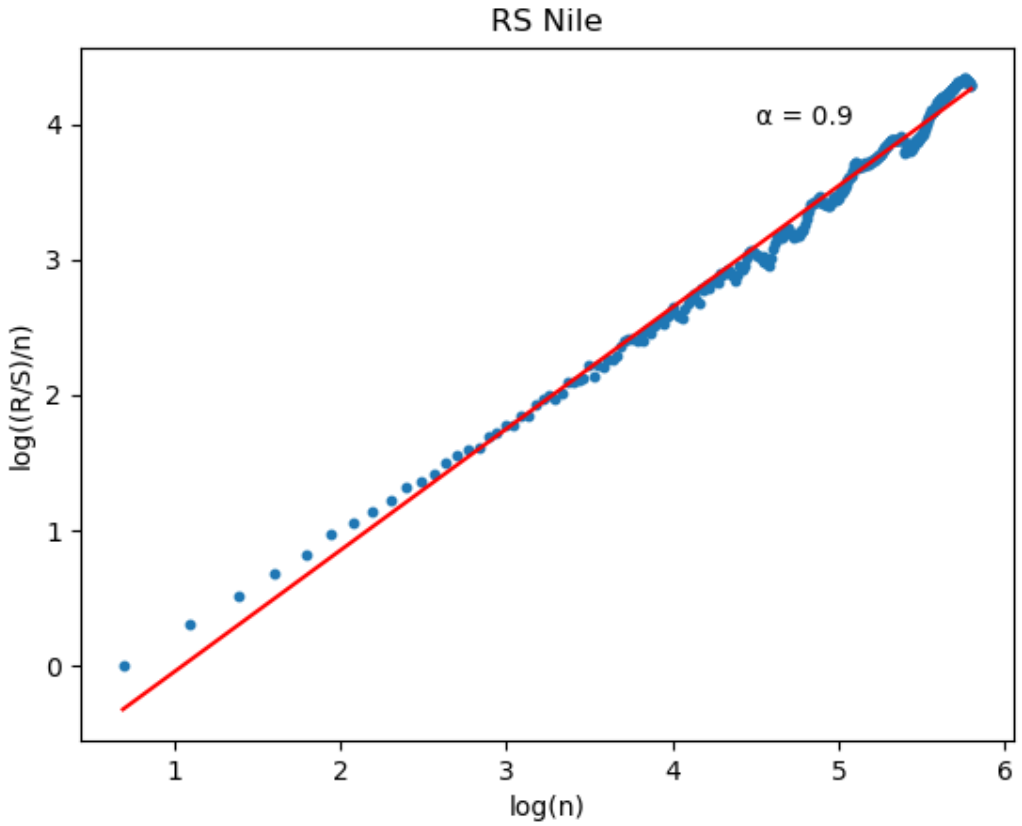
W opisany sposób otrzymujemy wykres zamieszczony poniżej, zawierający ostateczny wynik analizy danych nilu, które zebraliśmy w tabeli w kroku 10 naszego postępowania.



Zobrazowanie danych wynikowych nilu, dla wyżej określonych długości serii danych

Interesujące nas H, czyli współczynnik α prostej stworzonej przy pomocy regresji liniowej dla naszych końcowych danych wynosi ~0.87, czyli należy do zakresu od 0.5 do 1, co pozwala założyć, że w przyszłości trend danych utrzyma się.

Poprawność działania algorytmu możemy zaobserwować uruchomiając go dla dużo większej ilości badanych przedziałów. Interesującą obserwacją jest, że wynik nie zostaje zaburzony, dane wynikowe nadal układają się w niemalże idealnej prostej, a współczynnik zbliżył się do wartości 0.9, wskazując na jeszcze silniejszą możliwość zachowania trendu. Poniżej zamieszczony zostaje wykres, gdzie do obliczeń wykorzystano przedziały o kolejnych długościach $n \in [2, 331]$ przy czym $n \in \mathbb{Z}$, co daje 329 serii obliczeniowych.



Zobrazowanie danych wynikowych nilu dla 329 badanych serii

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.

1. Przygotowanie danych (0) wygląda podobnie jak w algorytmie R/S. Korzystając z funkcji pomocniczej *prepareData* odczytujemy:

➔ **indeksy** jakimi oznaczone są zebrane w we wskazanym pliku pomiary. W przeciwieństwie do algorytmu RS, teraz będą one odgrywać istotną rolę w obliczeniach i prezentacji danych;

➔ Tablicę z wartościami pomiarów, które będziemy badać. Dla uniknięcia późniejszej kolizji oznaczeń, nazwiemy ją ***D****;*

➔ Tablicę **L** z wybranymi długościami przedziałów, na jakie będziemy dzielić dane w poszczególnych seriach badawczych. Podobnie jak w metodzie RS, przyjmujemy założenie, że przy N równym ilości wszystkich wczytanych danych (długość tablicy D), szukamy takich długości n, że ilość podziałów $\lfloor \frac{N}{n} \rfloor$ jest potęgą 2.

Odnosząc się do naszego przykładu z Nilem, przygotowane dane będą takie same jak w przypadku algorytmu RS.

Pierwsza operacja jaką wykonujemy jest przygotowaniem do zamiany danych na Random Walk. W tym celu obliczamy średnią przygotowanego zbioru danych D (1). Dla naszego przykładu średnia ciągu wyniesie $avg = 1148.20$ [*].

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

Implementacja przygotowania danych do przetwarzania

2. Zaopatrzeni w dane z punktu pierwszego, możemy przygotować zbiór danych D do obróbki, zamieniając go na Random Walk, czyli ruch losowy lub błądzenie losowe. Działanie to ma na celu pozwolić na ułożenie danych, które potrafią być mocno rozproszone i przedstawić je w mniej chaotycznej postaci. Sama ta operacja pozwoli zaobserwować potencjalne zachowanie trendu.

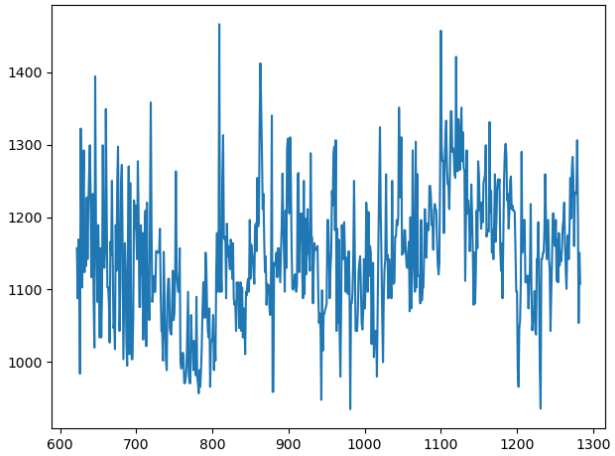
   Do uzyskania wspomnianego przedstawienia danych stosujemy prosty wzór, który opisuje, że każdemu elementowi ciągu D, przypisujemy jego sumę z poprzedzającymi go danymi, pomniejszonymi o wyliczoną wcześniej średnią całego ciągu.

$$D_n = \sum_{k=1}^{n} \left(d_k - \langle d_n \rangle \right) \quad \leftarrow \quad \langle d_n \rangle = avg \quad (2)$$

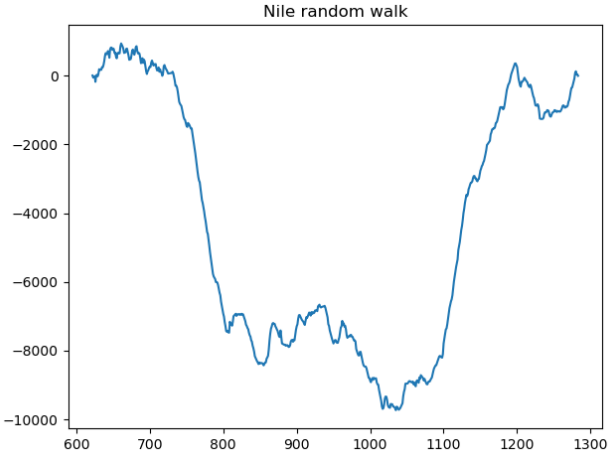
```
16      # 2 zmiana danych na random walk
17      randomWalk = []
18      cumulative_sum = 0.0
19      for i in range(0, N):
20          cumulative_sum += array[i] - avg
21          randomWalk.append(cumulative_sum)
```

Implementacja zamiany danych na tzw. Random Walk

Poniżej przedstawione zostało zestawienie, jak graficznie prezentowały się dane oryginalne (rys.1), a na jakich danych będziemy wykonywać dalsze obliczenia, dzięki zamianie postaci danych na Random Walk.



rys 1. Dane oryginalne [X]



rys 2. Dane zmodyfikowane - random walk

3. Z tak przygotowanymi danymi, zaczynamy działania należące do algorytmu DFA.

```
23      # petla do wybierania długości segmentów
24      F_avg = []
25      for segment_size in L:
26          # 3
27          temp_array = randomWalk.copy()
28          X = indexes.copy()
29          i = 0
30          k = indexes[0]
31          F = []
32          while i <= N - segment_size:
33              # znalezienie prostej w segmencie: line[0]=a; line[1]=b;
34              line = np.polyfit(X[0:segment_size], temp_array[0:segment_size], 1)
35
36              del temp_array[0:segment_size]
37              del X[0:segment_size]
38
39              # 4 wyliczenie F
40              F.append(calculateF(line, segment_size, i, k, randomWalk))
41              k = k + segment_size
42              i = i + segment_size
43
44          # 5 obliczenie sredniej fluktuacji dla danej dlugosci segmentu
45          F_avg.append(np.average(F))
46          print(segment_size, np.average(F))
```

Implementacja obliczeń algorytmu DFA

Ponieważ działanie algorytmu opera się na badaniu przedziałów, dzielimy zbiór danych na podzbiory o długości n, gdzie n jest wartością przygotowanej w koku 1 tabeli L. Tak jak wcześniej, rozpatrzymy działanie algorytmy dla $n=82$.

Dodatkowym manewrem jaki zostaje wykonany, jest utworzenie przy każdej nowej serii badawczej kopii tablicy z wartościami RandomWalku w tablicy Y (sugerując się nazewnictwem osi układu kartezjańskiego), ponieważ są to wartości oraz kopii tablicy indeksów, analogicznie do tablicy X, ponieważ dane te odpowiadają osi X. Wykonujemy te operacje, ponieważ w czasie działania algorytmu, będziemy usuwać przebadane już ciągi danych, a ponieważ algorytm wykonujemy wielokrotnie, nie chcemy utracić oryginalnego zbioru danych.

Ostatnią zmienną jaką przygotowujemy dla danej serii, jest tablica F, w której zbierane będą wartości fluktuacji dla poszczególnych przedziałów wielkości n.

4. Z takim przygotowaniem dla serii, rozpoczynamy badanie jej przedziałów. W każdym przedziale musimy odnaleźć lokalny trend. W tym celu do każdego segmentu o długości n, dopasowujemy prostą.

$$Y_i = a \cdot i + b \quad \leftarrow \quad i \in X \quad (4)$$

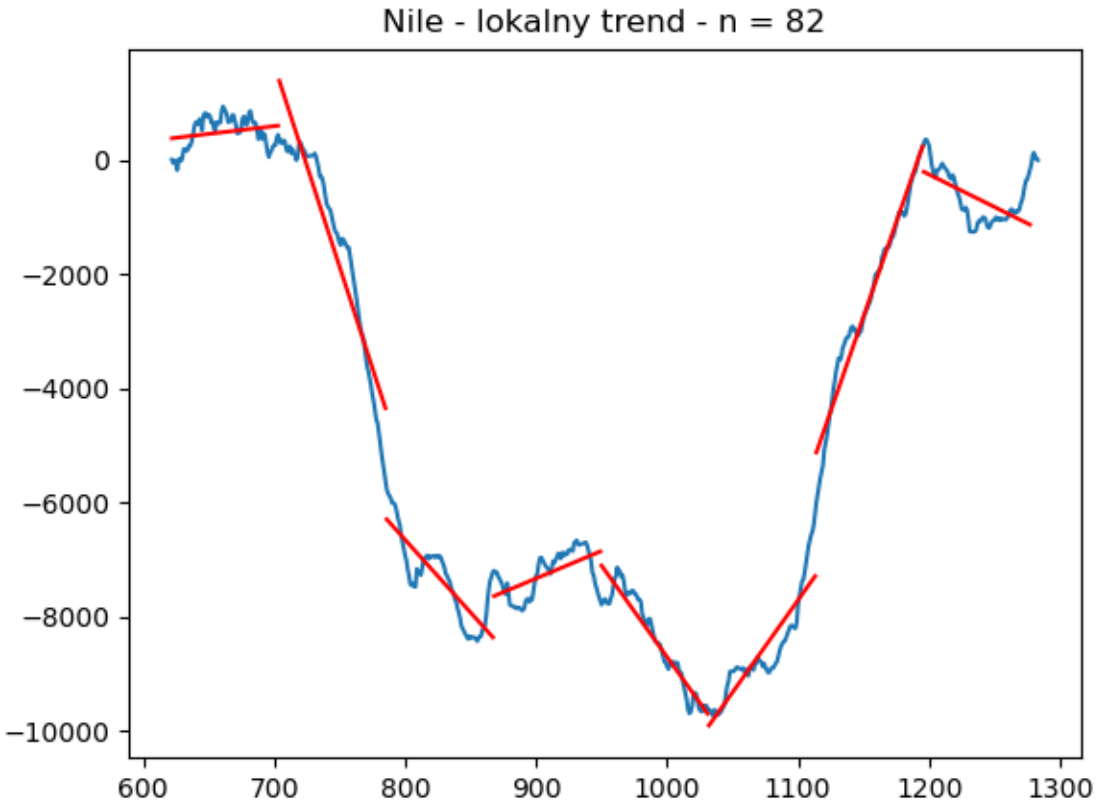
Po uzyskaniu interesujących nas współrzędnych, przechodzimy do kroku 5, w którym na ich podstawie obliczymy fluktuację badanego przedziału. Kroki 4-5 wykonujemy dla wszystkich segmentów o długości n. Zestawienie segmentów i współrzędnych prostych określających ich trendy można zobaczyć w tabeli poniżej.

n = 82					
id	Badany przedział	Zakres indeksów badanego przedziału [X]	Wartości skrajne przedziału [Y] *	a *	b *
0	0 - 81	622 - 703	8.80, 444.40	2.72	-1311.41
1	82 - 163	704 - 785	372.20, -5641.20)	-70.82	51250.62
2	164 - 245	786 - 867	-5767.40, -7213.79	-25.50	13750.07
3	246 - 327	868 - 949	-7196.00, -7724.79	9.64	-1.60
4	328 - 409	950 - 1031	-7791.59, -9620.00	-32.05	23341.44
5	410 - 491	1032 - 1113	-9626.19, -6188.60	3.23	-4.32
6	492 - 573	1114 - 1195	-5990.79, -237.81	6.61	-7.88
7	574 - 655	1196 - 1277	298.61, -166.78	-1.14	1.34

Tabela 4: Wartości prostych określających trend lokalny w przedziałach o wielkości 82 danych

Dzięki pracy na kopiach danych, utworzonych w kroku 3, po wyliczeniu współrzędnych prostej dla danego przedziału, możemy usunąć go ze zbiorów X i Y. Sprawia to, że kolejny przedział, który będziemy badać znajdzie się na początku tablic X i Y, a to pozwala nam na przechodzenie kroku 4

zawsze wybierając pierwsze n wartości z tych tablic, dzięki czemu zmniejszamy prawdopodobieństwo pomyłki przy szukaniu właściwych indeksów.



Zobrazowanie trendu na badanych przedziałach

5. Na podstawie wyprowadzonego równia prostej dla aktualnego przedziału, wyprowadzamy wartość fluktuacji F, korzystając z poniższego wzoru:

$$F(n)=\sqrt{\frac{1}{n}\sum_{i=i_0;\,j=k}^{i_0+n-1;\,j=k+n-1}\left(RW_j-i\cdot a-b\right)^2} \quad (8)$$

$$i_0=X_0;\quad k=id\cdot n;\quad RW-RandomWalk$$

Dla przykładu, biorąc pierwszy ciąg 82 danych, równanie wyglądało by w następujący sposób:

$$F(82)=\sqrt{\frac{1}{82}\sum_{i=622;\,j=0}^{i=703;\,j=81}\left(RW_j-i\cdot 2.72+1211.41\right)^2}$$

W tym momencie znów korzystamy z pierwotnie przygotowanego do obliczeń ciągu danych, czyli Random Walk-u. Ponownie mamy zaimplementowaną funkcję pomocniczą

calculateF, która zwraca nam wartość fluktuacji dla określonego przedziału. Funkcja przyjmuje 5 argumentów:

- **line** – tablica ze współrzędnymi *a* i *b*, które otrzymaliśmy w kroku 4 przy użyciu funkcji *polyfit*, do wyprowadzenia równania prostej;
- **n –** wielkość badanego przedziału
- **i –** określa początkową wartość x, w równaniu prostej, dla badanego przedziału danych
- **k –** określa indeks początkowy, od którego należy zacząć pobierać dane z tablicy danych RW;
- **RW** – pełna tablica danych – Random Walk

```
77  # 8
78  def calculateF(line, n, i, k, RW):
79      segment_sum = 0
80      for n in range(i, i + n):
81          segment_sum += (RW[n] - (line[0] * k) - line[1]) * (RW[n] - (line[0] * k) - line[1])
82          k = k + 1
83
84      F = np.sqrt((1 / n) * segment_sum)
85      return F
```

Implementacja funkcji pomocniczej wyliczającej wartość fluktuacji

Wartość fluktuacji dla poszczególnych segmentów o wielkości n zbieramy w tablicy F (5).

n = 82					
id	a *	b *	i_0	0	Wartość fluktuacji $[F_{id}]$ *
0	2.72	-1311.41	622	0	270.09
1	-70.82	51250.62	704	82	587.09
2	-25.50	13750.07	786	164	403.90
3	9.64	-1.60	868	246	313.87
4	-32.05	23341.44	950	328	249.05
5	3.23	-4.32	1032	410	380.49
6	6.61	-7.88	1114	492	290.24
7	-1.14	1.34	1196	574	364.27

6. Po przejściu przez wszystkie segmenty i uzyskaniu pełnej tablicy F, wyprowadzamy średnią fluktuację dla podziałów o wielkości *n*.

```
57          # 6 obliczenie sredniej fluktuacji dla danej dlugosci segmentu
58          F_avg.append(np.average(F))
```

Wyliczenie średniej fluktuacji dla podziałów o wielkości n

Jest to wartość końcowa jaką chcemy otrzymać z naszych obliczeń dla wybranego n. Dodajemy ją do tablicy F_avg (6), w której zbieramy średnie wartości fluktuacji, dla każdej badanej wielkości n.

7. Kroki od 3-6 powtarzamy dla każdej wielkości n, która zawiera się w tablicy L.
8. 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 (7).

D		
j	$n = L_j$	$F(L) = F\_avg_j^*$
0	331	1592.46
1	165	854.18
2	82	357.37
3	41	226.09
4	20	104.35
5	10	59.97
6	5	29.31

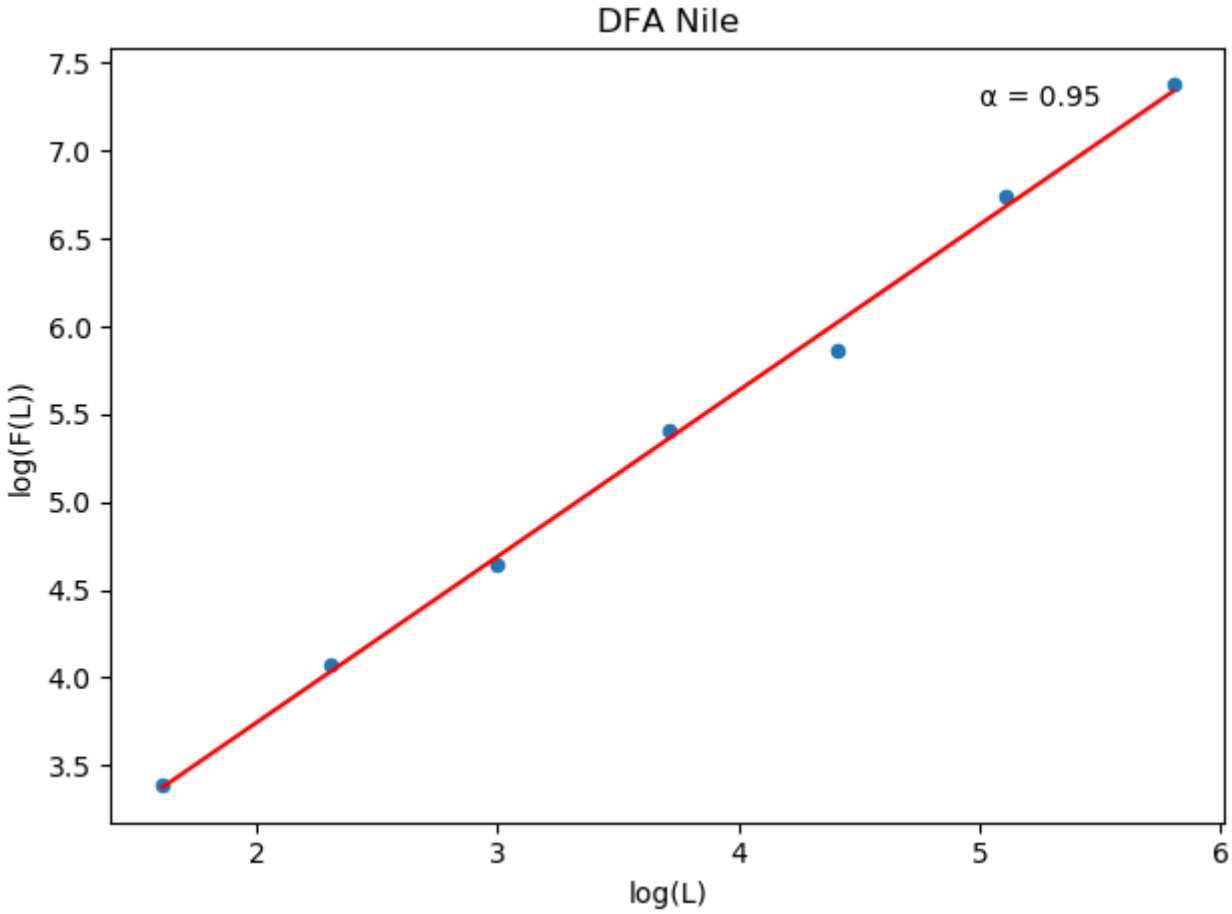
Tabela 5: Zestawienie zależności średniej fluktuacji przedziałów od ich długości n

Z tak przygotowanych danych generujemy wizualizację wyników, przy pomocy poniższej implementacji,

```
48    # 6 double logaritmic plot
49    plt.scatter(np.log(L), np.log(F_avg), s=20)
50    plt.title('DFA nile')
51    plt.ylabel('log(F(L))')
52    plt.xlabel('log(L)')
53
54    result = np.polyfit(np.log(L), np.log(F_avg), 1)
55    print('alfa = ', result[0])
56    print(result)
57
58    plt.text(5, 7.25, '\u03B1 = {}'.format(round(result[0], 2)))
59    x1 = np.log(L[0])
60    x2 = np.log(L[-1])
61    plt.plot([np.log(L[0]), np.log(L[-1])], [result[0]*x1+result[1], result[0]*x2+result[1]], 'red')
62    plt.show()
```

Implementacja generowania wizualnego przedstawienia wyniku algorytmy DFA

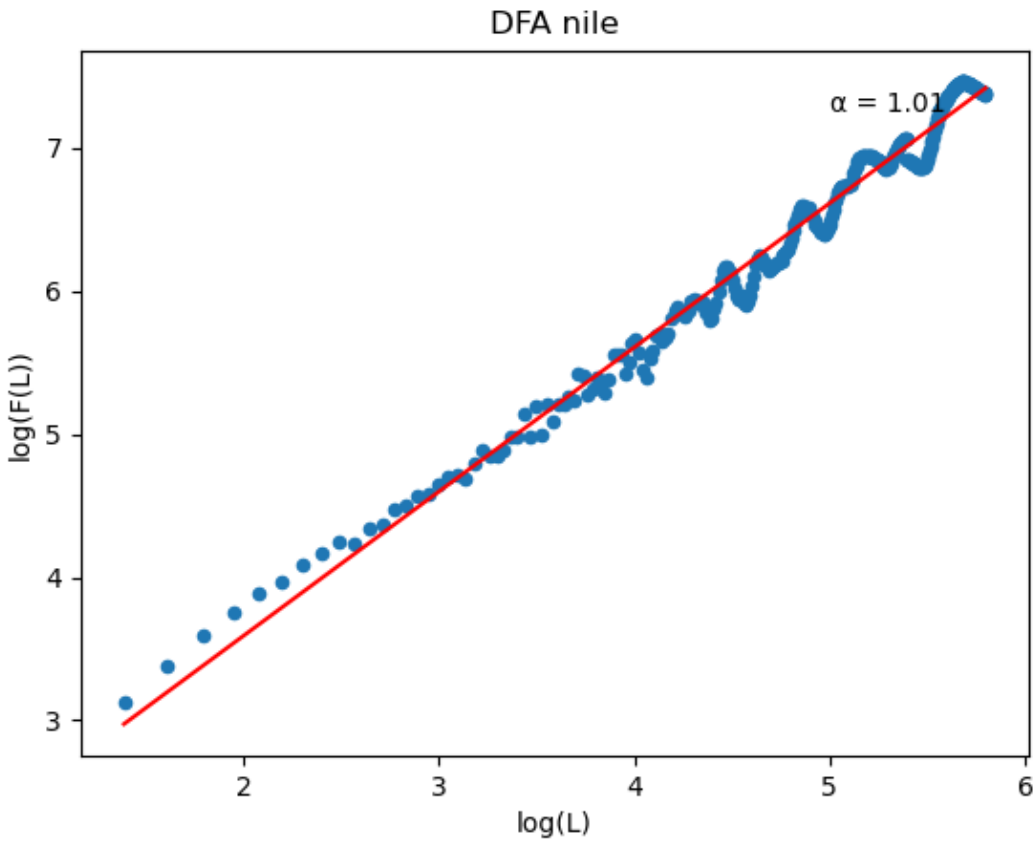
której efekt można zobaczyć na wykresie ponieżej.



Wynik algorytmy DFA dla danych nilu, przy określonych wyżej przedziałach czasowych

Możemy z niego wyczytać, że wykładnik Hursta wynosi 0.95. Oznacza to wysokie prawdopodobieństwo, że trend danych utrzyma się, ponieważ wynik znajduje się w zakresie (0.5, 1]. Co dobre, wynik tej metody nie odstaje mocno od wyniku jaki uzyskaliśmy z metody RS. Możemy zatem założyć, że obie metody działają poprawnie.

Przeprowadzając podobne doświadczenie jak w metodzie RS i wywołując algorytm DFA dla 327 kolejnych, rożnych przedziałów czasowych, zaobserwujemy zbliżenie się wykładnika Hursta do 1, a nawet jego przekroczenie. Zasadniczo takie rezultaty nie powinny się pojawiać, ale obserwując widoczne zmiany w układzie danych, możemy założyć, że przyszły ruch danych obniży ponownie współczynnik, jednak będzie on stale oscylował w okolicy 1, dopóki nie nastąpi gwałtowna zmiana zachowania danych, współczynnik zacznie spadać i trend danych się zmieni.



Rezultat algorytmu DFA dla danych nilu, przy segmentach wielkości od 4 do 331

2.4. Algorytm DMA (Detrended Moving Average)

Algorytm DMA jest widocznie najprościej zbudowany i łatwy do implementacji. Dane ulegają wygładzeniu za pomocą tytułowej ruchomej średniej. Cel tej operacji jest taki sam jak w przypadku stosowania błądzenia losowego przy algorytmie DFA – uzależnienie od siebie danych i ich uporządkowanie.

O ile jest to metoda w implementacji banalna, w samym zastosowaniu już nie. Pierwszą czynnością, która może sprawić kłopot w wypadku tego algorytmu, jest wybór zakresu wielkości n segmentu, okienka, którym będzie się przemieszczać po wczytanych danych. Druga sprawa, to dobór danych, na których przeprowadzamy obliczenia.

1. Do uruchomienia algorytmu przygotowujemy się w podobny sposób jak wcześniej – pobieramy dane z pliku. Interesują nas wartości zebranych obserwacji w tablicy **X**, oraz tablica **L** ze specjalnie wybranymi wielkościami, powiedzmy, okienka badawczego, którym będziemy przemieszczać się po szeregu czasowym. Tym razem jednak wartości te będą następującymi po sobie kolejnymi wielkościami, czyli np. zbiór $L=[300,301,...,400]$.

```
8   def DMA():
9       # 0
10      indexes, X, L = prepareData('nile.txt', 20, 4)
11      N = len(X)
12
13      modifiedStandardDeviation = []
14      for n in L:
15          # 1 srednia ruchoma dla przedziałów dlugosci n
16          i = n-1
17          movingAverage = X.copy()
18          while i < N:
19              cumulative_sum = 0.0
20              for k in range(0, n):
21                  cumulative_sum += X[i-k]
22              movingAverage[i] = cumulative_sum/n
23              i += 1
24
25          #2
26          sum = 0.0
27          for i in range(n, N):
28              sum += (X[i-1] - movingAverage[i-1]) * (X[i-1] - movingAverage[i-1])
29          modifiedStandardDeviation.append(np.sqrt(sum / (N - n)))
```

Implementacja algorytmu DMA

2. W algorytmie DMA nie dzielimy zbioru danych X na przedziały o równej długości, lecz wybieramy długość segmentu, na obszarze którego w danym kroku będziemy liczyć średnią i przesuwać się z każdym krokiem o jeden element, aż dojdziemy do końca naszej tablicy z danymi.

   Celem tego kroku, jest obliczenie średnich, dla elementów od n do N w tablicy X. Pętlę taką wykonujemy stosując wzór, który opisuje, że elementom tablicy X, zaczynając od elementu

i=n (przy założeniu, że tablicę indeksujemy od 1), przypisujemy średnią $\tilde{x}_i$ jako średnią z sumy wyrazu i-tego i n-1 poprzedzających go elementów.

$$\tilde{x}(i)=\frac{1}{n}\sum_{k=0}^{n} X(i-k) \quad (1)$$

```
16          # 1 srednia ruchoma dla przedziałów dlugosci n
17          i = n-1
18          movingAverage = X.copy()
19          while i < N:
20              cumulative_sum = 0.0
21              for k in range(0, n):
22                  cumulative_sum += X[i-k]
23              movingAverage[i] = cumulative_sum/n
24              i += 1
```

Implementacja obliczania średniej ruchomej ale segmentu o wielkości n

Czyli np. dla n=300, pierwszą wartość średniej ruchomej wyliczymy dla X(299), sumując wartości od X(0) do X(299) i dzieląc tę sumę przez 300. Kolejno obliczymy średnią ruchomą dla X(300) sumując wartości od X(1) do X(300) i dzieląc przez 300. Tak przesuwając się, dojdziemy to wyliczania ostatniej wartości średniej dla X(661), gdzie sumę elementów od X(360) do X(661) podzielimy przez 300.

3. Kolejnym krokiem, jest wyliczenie zmodyfikowanego odchylenia standardowego. Jest to połączenie modyfikacji danych wejściowych o wyliczoną w kroku 2 średnią i na ich podstawie obliczania odchylenia dla aktualnie rozpatrywanego n. Działanie to przedstawia wzór:

$$\sigma_{DMA}(n)=\sqrt{\frac{1}{N-n}\sum_{i=n}^{N}(x_i-\tilde{x}_i)^2¿}$$

Jak dobrze widać, wyrażenie po którym sumujemy, jest częścią równania, która odpowiada za modyfikację szeregu danych X o wyliczoną wyżej średnią.

```
26          #2
27          sum = 0.0
28          for i in range(n, N):
29              sum += (X[i-1] - movingAverage[i-1]) * (X[i-1] - movingAverage[i-1])
30          modifiedStandardDeviation.append(np.sqrt(sum / (N - n)))
```

Implementacja wyprowadzania zmodyfikowanego odchylenia standardowego dla segmentu o wielkości n

4. Kroki 2 i 3 powtarzamy dla kolejnych wielkości n z tablicy L.

   Wykonując algorytm dla danych Nilu, przy n z zakresu [300, 400), otrzymamy poniższe wyniki:

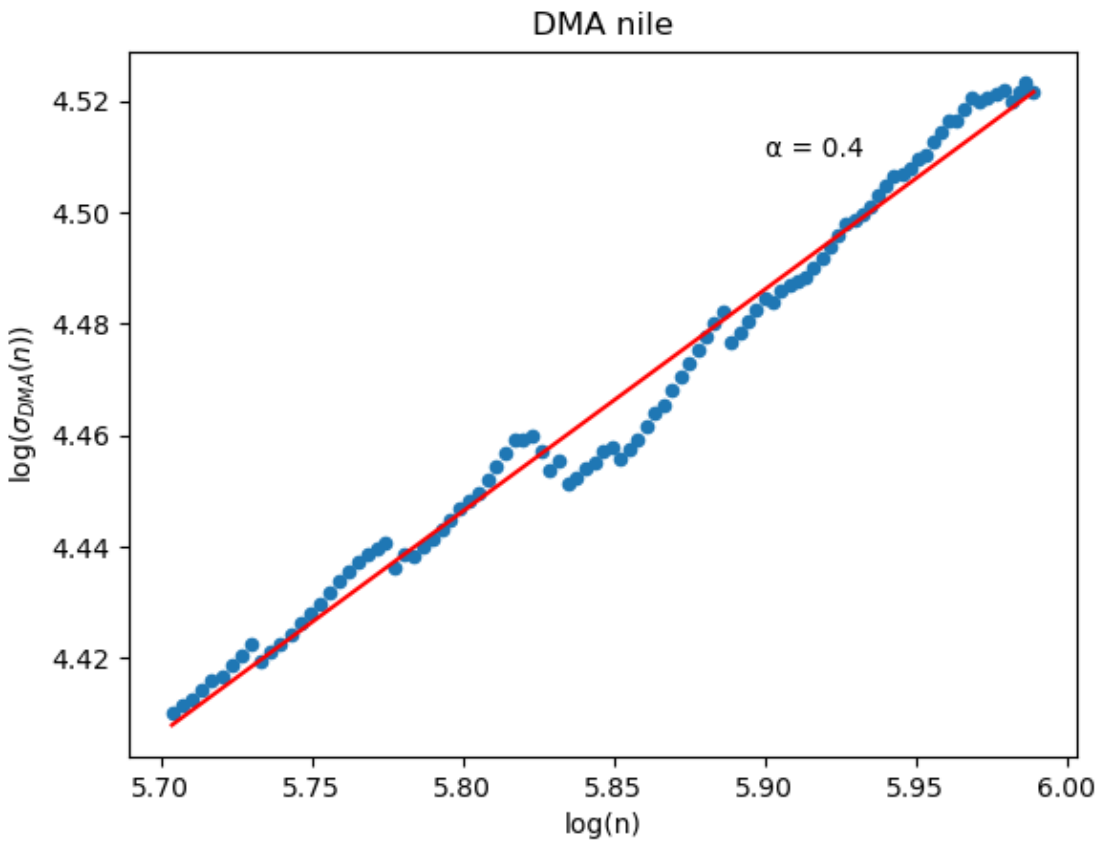
$n \in L$	$\sigma(n)$ *
300	82.29
301	82.41

302	82.47
303 - 397	...
398	92.13
399	91.10

5. Na koniec przedstawiamy dane na wykresie podwójnie logarytmicznym (3).

```
34    # 3 double logaritmic plot
35    plt.scatter(np.log(L), np.log(modifiedStandardDeviation), s=20)
36    plt.title('DMA nile')
37    plt.ylabel(r'log($\sigma_{DMA}(n)$)')
38    plt.xlabel('log(n)')
39
40    result = np.polyfit(np.log(L), np.log(modifiedStandardDeviation), 1)
41    print('alfa = ', result[0])
42    print(result)
43
44    plt.text(4, 4.46, '\u03B1 = {}'.format(round(result[0], 2)))
45    x1 = np.log(L[0])
46    x2 = np.log(L[-1])
47    plt.plot([np.log(L[0]), np.log(L[-1])], [result[0] * x1 + result[1], result[0] * x2 + result[1]], 'red')
48    plt.show()
```

Implementacja generowania wizualizacji wyniku algorytmu DMA



Wynik algorytmu DMA dla danych nilu przy n z zakresu 300-399

Przy pierwszym spojrzeniu rzuca się w oczy różnica wartości współrzędnej α i chaotyczność układu danych, w porównaniu z wynikami metod RS i DFA. Tak jak wspomniałam wcześniej, jest to metoda w zastosowaniu niełatwa. Ciężko bowiem jest dobrać rozmiary okienka badawczego do zebranych przez nas danych. Mimo, że jest to metoda oficjalnie uznawana za jedną z metod do wyznaczanie wykładnika Hursta, jest także tą najczęściej odrzucaną, ponieważ jej rezultat zazwyczaj odstaje silnie od innych algorytmów. Jak widać w omawianym tutaj przykładzie, przy algorytmie DMA wylądowaliśmy wręcz po przeciwnej stronie wartości 0.5, kiedy przy RS i DFA dążyliśmy do okrągłej wartości 1.

Dodatkową wadą rezultatów jakie daje ta metoda, jest brak ścisłej zależności w przedstawionych danych. Wybierając okna badawcze o różnych rozpiętościach i w różnej ilości, możemy zbliżyć się do $\alpha=0$, a nawet zejść poniżej, ale równocześnie jesteśmy w stanie osiągnąć wyniki z kategorii $\alpha>4$ gwałtownie wychodzące poza oczekiwany przez nas zakres.

3. Analiza różnych szeregów czasowych

Żeby wyżej opisane implementacje nie sprawiały wrażenia, iż działają specjalnie pod nasz wiodący przykład danych o wylewach Nilu, w tym rozdziale przyjrzymy się ich działaniu dla innych szeregów czasowych. Po algorytmach RS i DFA oczekujemy mniej lub bardziej zgodności, co do wartości do której zdążają. Przy algorytmie DMA, jako tym najbardziej zawodnym, spróbujemy odnaleźć takie wielkości okien badawczych, które zbliżą wartość wykładnika do rezultatów wcześniejszych metod.

4. Podsumowanie

Przypisy

* – wszystkie podane wyniki są w przybliżeniu do 2 miejsc po przecinku.

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
7. https://quantdare.com/demystifying-the-hurst-exponent/
8. Korelacje
9. Odchylenia standardowe
10. Fluktuacje
11. Random Walk