Szanowni Państwo,

poniżej znajduje się kilka linków do stron, z których można skorzystać w razie pojawienia się trudności z dopasowaniem funkcji sinusoidalnej do waszych danych.

Ogólnie metoda curve_fit minimalizuje funkcję straty, w celu znalezienia najlepszego dopasowania funkcji do danych za pomocą metod numerycznych. Jak wiemy, takie metody mogą działać nieskutecznie w przypadku, gdy punkty początkowe znajdują się daleko od prawdziwych wartości, które estymujemy. Zatem, aby uzyskać jak najlepszą estymację powinniśmy zadać punkty początkowe, które będą możliwie blisko rozwiązania.

Poniżej znajdują się dwa wątki na forach, w których poruszana jest kwestia dopasowywania funkcji sinusoidalnej do danych.

- **[Warto zwrócić uwagę szczególnie na ten wątek] Przykład metody opartej na FFT, która dopasowuje funkcję sinusoidalną do danych bez konieczności wprowadzenia parametrów wejściowych. Można sobie przetestować to podejście. W wielu przypadkach może okazać się skuteczne oraz nieść wiele korzyści związanych z tym, że nie trzeba ręcznie wprowadzać domniemanych punktów początkowych.**
  https://stackoverflow.com/questions/16716302/how-do-i-fit-a-sine-curve-to-my-data-with-pylab-and-numpy

- **Drugim podejściem jest użycie funkcji curve_fit w takiej samej formie, jak na zajęciach – czyli z koniecznością dodania parametrów wejściowych :**
  https://stackoverflow.com/questions/61710790/curve-fitting-of-sine-function-in-python-using-scipy-is-not-yielding-desired-out

  Jeśli chcą się Państwo zapoznać ze szczegółami działania 2 podejścia to poniżej opisałam bardziej szczegółowo jego działanie (bazując na tym co omawialiśmy na zajęciach – zadanie 4 z listy 5).

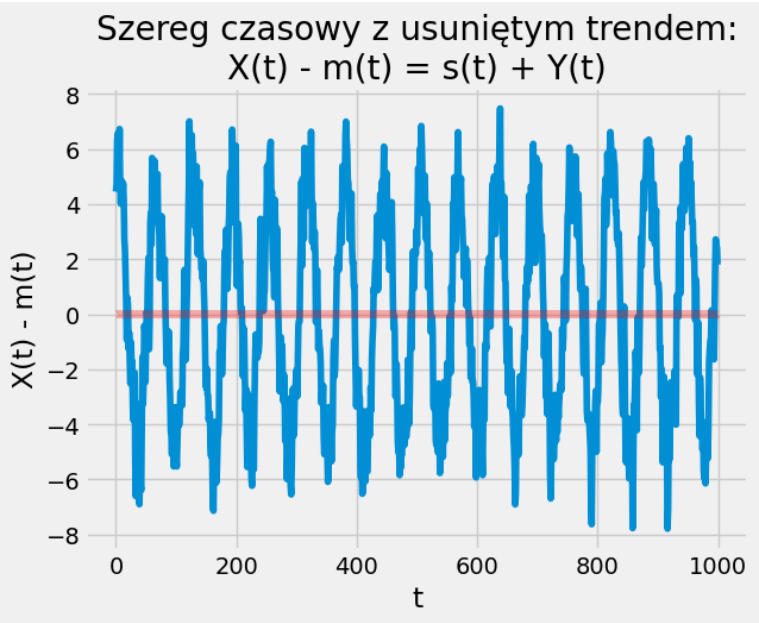
  Jeśli zdecydujemy się korzystać z drugiego rozwiązania, to konieczne będzie samodzielne dobranie parametrów wejściowych. Teoretycznie metoda curve_fit umożliwia działanie bez podania tych parametrów (ale wówczas sama wprowadza w ich miejsce wartości domyślne – które zwykle nie są optymalne dla podanego problemu). W przypadku, gdy dopasowujemy poniższą funkcję do danych:

```
def funct(x, c, d, e):
    return c * np.sin(d * x + e)

xdata = t # czas
ydata = X2_t # dane (po usunięciu trendu)

Q = np.quantile(X2_t, 0.9)
p0 = [Q,  (2*np.pi) * (1/60), random.random()]
params, pcov = curve_fit(f=funct, xdata=xdata, ydata=ydata,
                         p0=p0, bounds=((Q, 0, 0), (10, np.inf, 1)))
```

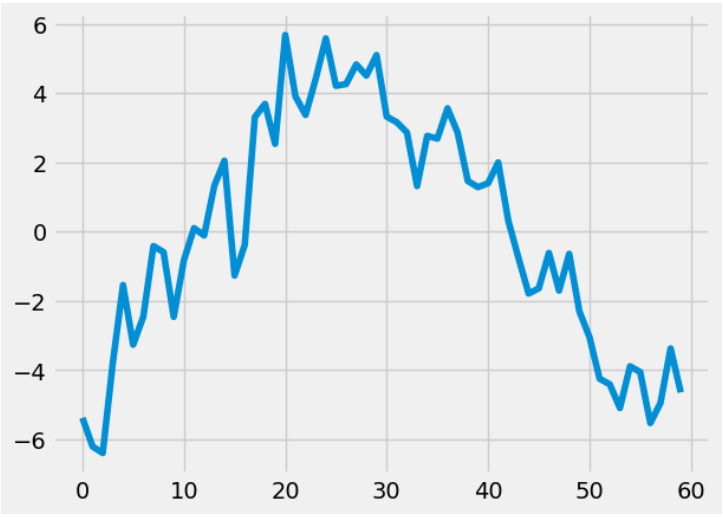
  to możemy wstępnie na podstawie wykresu oszacować wartości c, d oraz e.

- Parametr c możemy oszacować na podstawie amplitudy. W poniższym przykładzie wybrałam jako punkt początkowy dla c wartość Q, czyli kwantyla na poziomie 0.9 (ale mogą Państwo przetestować również inne kwantyle np. 0.85, 0.95, 0.99) lub np. wartość maksymalną (w celu sprawdzenia czy zmiana pozytywnie wpłynie na dopasowanie funkcji).
- Parametr d szacujemy bazując na tym jak długi jest okres funkcji. Na tej podstawie możemy obliczyć częstotliwość (która po pomnożeniu przez 2*pi zwróci nam pulsację – czyli nasz parametr d).

Przykładowo na zajęciach wygenerowaliśmy takie dane:



czerwoną linią zaznaczyłam dopasowanie zwrócone przez Pythona kiedy nie podamy mu optymalnych parametrów. Jak widać jest bardzo słabe. Możemy sobie teraz takie dane przedstawić na wykresie dla pewnego zakresu t, np.

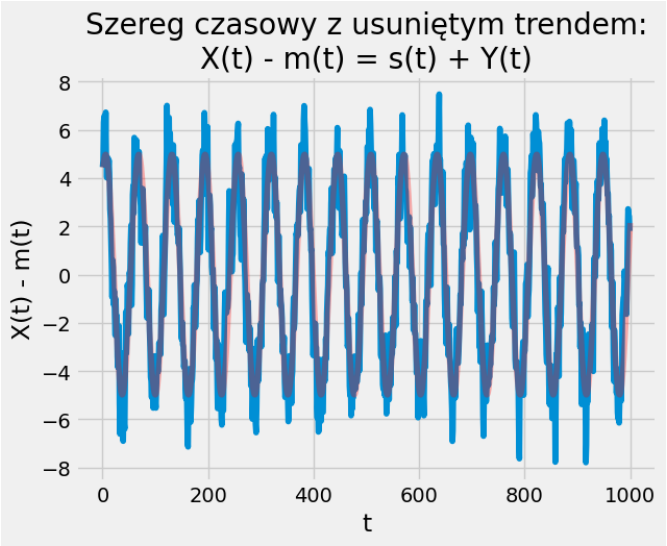


Na tym zbliżonym wykresie widzimy, że T/2 wynosi prawie 60 (dlatego w kodzie z zajęć używaliśmy tej wartości).

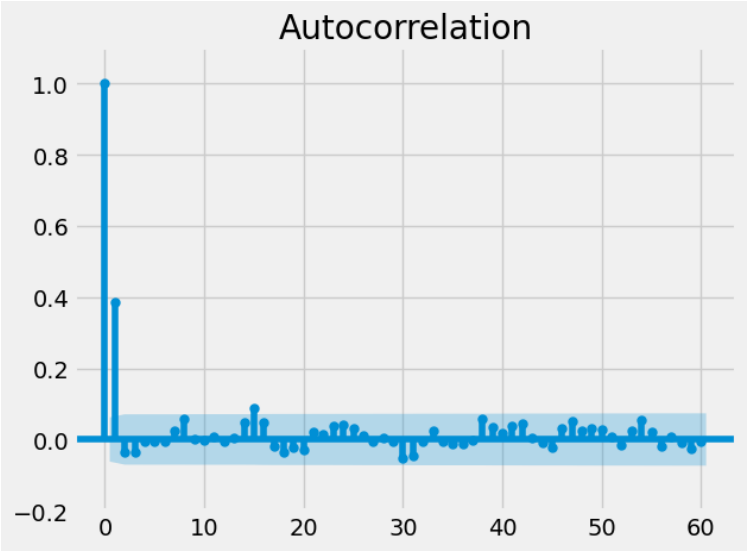
- Przesuniecie fazowe określiłam jako dowolną liczbę z zakresu (0,1), ponieważ algorytm dość dobrze radzi sobie z jego estymacją (jeśli spojrzą Państwo do kodu z pierwszego linka, to zauważycie, że w tamtym algorytmie ten parametr został ustawiony na 0, czyli zaczynamy od założenia, że nie mamy przesunięcia fazowego).

Kluczowe do poprawnego działania curve_fit jest podanie punktów początkowych p0. Ale możemy również poprawić jego skuteczność dodając bounds (czyli ograniczenia wartości parametrów). Podajemy je, jeśli mamy pewność, że nasz parametr nie przekroczy określonej wartości. Składnia jest taka, że podajemy dwie krotki: bounds = ((c_min, d_min, e_min), (c_max, d_max, e_max)). Ważne jest, że musimy podać wartości dla każdego parametru (czyli jeśli nie chcemy go ograniczać, to wystarczy wpisać jako ograniczenie numpy.inf lub -numpy.inf).

Po zastosowaniu wyżej wymienionych procedur otrzymaliśmy następujące dopasowanie.



Dane w zadaniu pochodziły z szeregu MA(1) z dodanym trendem i sezonowością. Po usunięciu trendu i sezonowości ACF wygląda następująco:



Widzimy, że zgadza się to z wiedzą z wykładu - dla szeregów MA ostatni lag dla ACF, który nie wpada w przedziały ufności określa rząd modelu.