Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Modelowanie i identyfikacja

Sprawozdanie z projektu II, zadanie 33

Adam Sokołowski

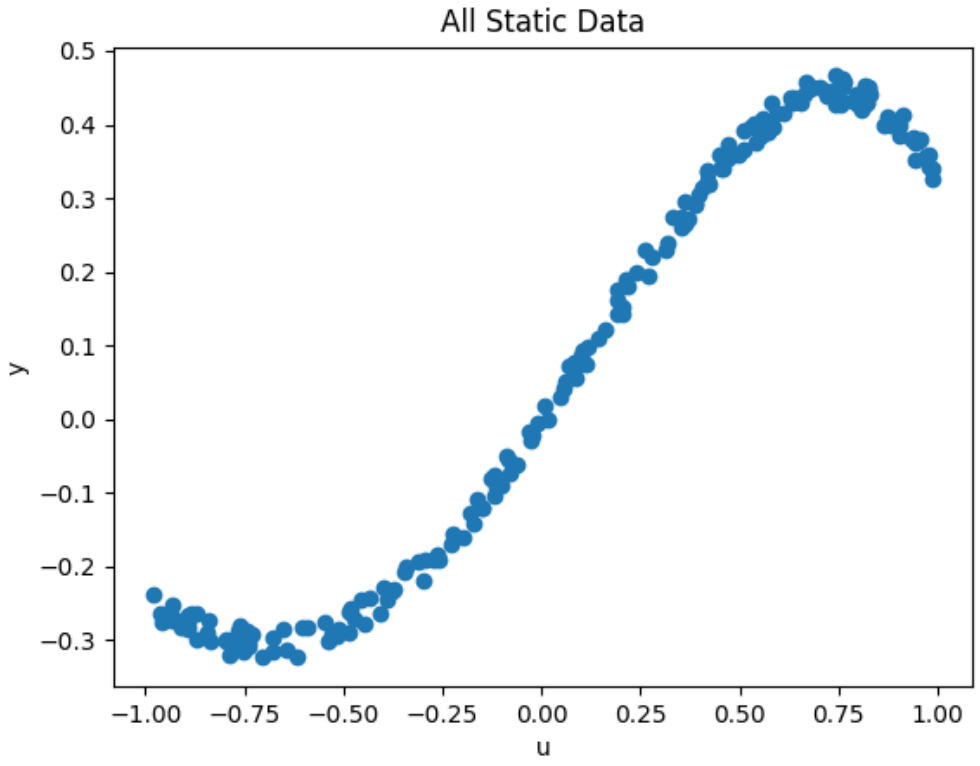
Warszawa, 2024

Spis treści

1. Identyfikacja modeli statycznych

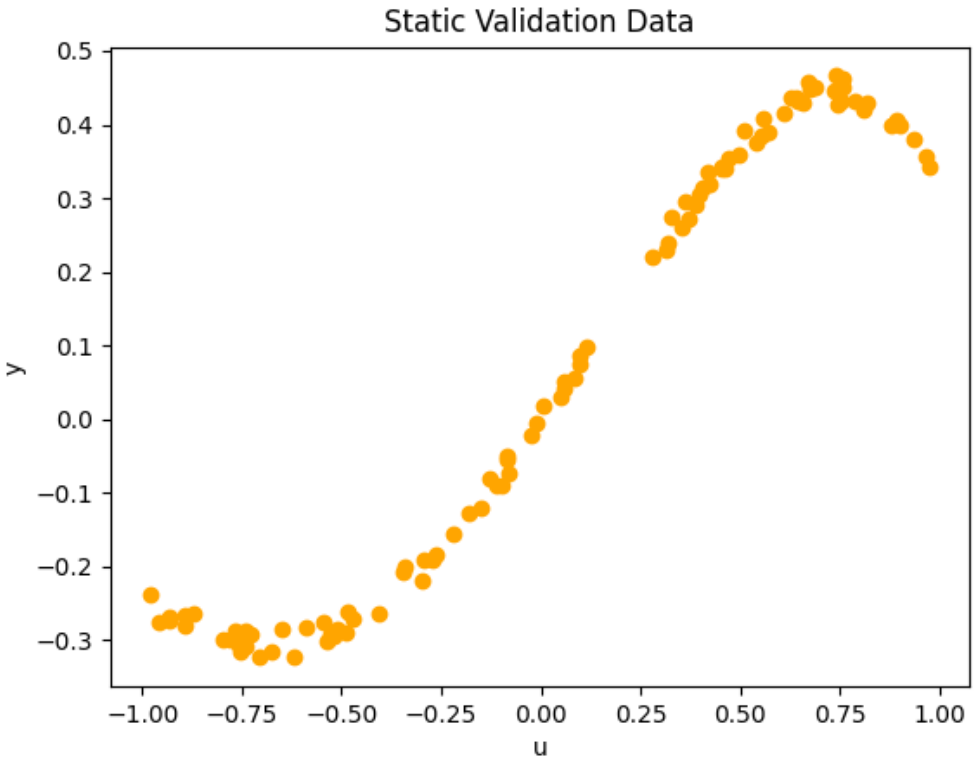
1.1. Podział danych

Dane podzielono na dwa zbiory: uczący i weryfikujący. Mają one tyle same elementów, a uzyskano je biorąc wyrazy ze zbiory wszystkich danych o indeksie parzystym do zbioru uczącego, a on nieparzystym do weryfikującego.



Rys. 1.1. Niepodzielone dane

Rys. 1.2. Dane uczące



Rys. 1.3. Dane weryfikujące

1.2. Wyznaczanie modelu statycznego liniowego

W celu wyznaczenie liniowego modelu statycznego rozwiązano zadanie najmniejszych kwadratów i policzono błąd kwadratowy dla otrzymanych wartości. Poniżej wzór obliczanego modelu statycznego liniowego.

$$y(u) = a_0 + a_1 u \tag{1.1}$$

1.2.1. Implementacja algorytmu w python

Obliczanie macierzy M.

```
M_ucz = np.column_stack((np.ones_like(u_ucz), u_ucz))
M_wer = np.column_stack((np.ones_like(u_wer), u_wer))
M_stat = np.column_stack((np.ones_like(u), u))
```

Obliczanie współczynników

```
wsp_ucz = np.linalg.lstsq(M_ucz, y_ucz, rcond=None)[0]
```

Obliczanie wyjścia modelu

```
y_ucz_hat = np.dot(M_ucz, wsp_ucz)
y_wer_hat = np.dot(M_wer, wsp_ucz)
y_stat = np.dot(M_stat, wsp_ucz)
```

Obliczanie błędu kwadratowego

```
error_ucz = np.sum((y_ucz - y_ucz_hat) ** 2)
error_wer = np.sum((y_wer - y_wer_hat) ** 2)
```

1.2.2. Otrzymane błędy modelu

Metodą najmniejszych kwadratów wyznaczono wartości współczynników, dla których sumaryczny błąd kwadratowy osiągnął wartości:

Dla zbioru uczącego

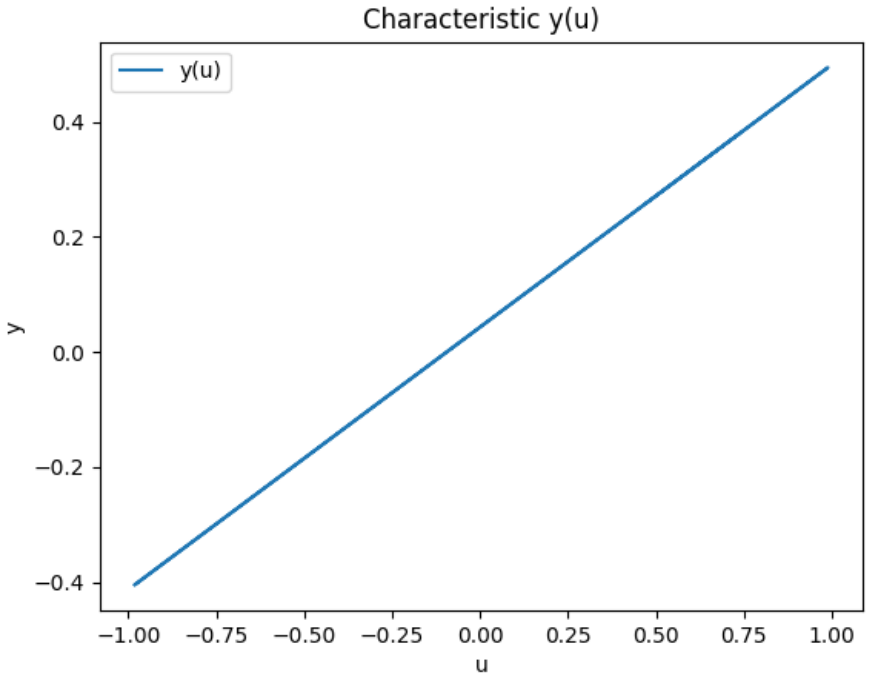
$$E_{ucz} = 0.600$$

Dla zbioru weryfikującego:

$$E_{wer} = 0.650$$

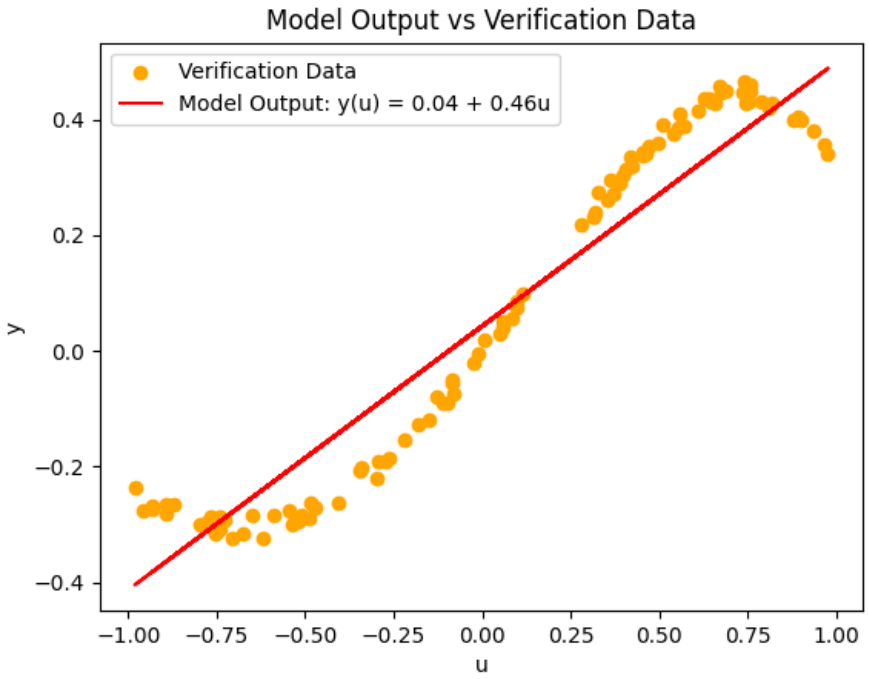
Jak widać wartość błędu dla danych weryfikujących jest większa. Jest to wynik, którego należałoby się spodziewać, gdyż model zwykle radzi sobie gorzej z danymi, które nie były użyte w modelowaniu. Niestety po samych wartościach błedu cieżko stwierdzić jak dobrym indentyfikatorem jest model statyczny liniowy.

1.2.3. Wykresy modelu na tle danych



Rys. 1.4. Charakterystyka statyczna

Rys. 1.5. Wyjście modelu na tle danych uczących



Rys. 1.6. Wyjście modelu na tle danych weryfikujących

Jak widać z powyższych rysunków modelu liniowy jest słabym identyfikatorem danego procesu. Można jednak wywnioskować, że metoda najmniejszych kwadratów działa dobrze gdyż otrzymana prosta wydaje się być najlepszym możliwym liniowym przybliżeniem tych danych.

1.3. Wyznaczanie modelu statycznego nieliniowego

Wykorzystywany model będzie modelem wielomianowych postaci

$$y(u) = a_0 + \sum_{i=1}^{N} a_i u^i \tag{1.2}$$

gdzie N jest stopniem wielomianu.
Skorzystano z metody najmniejszych kwadratów i błedu kwadratowego.

1.3.1. Implementacja algorytmu

Wybór stopni wielomianu

```
N_values = [1, 2, 3, 4, 5, 6]
```

Pętla obliczająca macierze M, współczynniki modelu i wyjścia dla kolejnych stopni wielomianu

```
for N in N_values:
    M_ucz = np.column_stack([u_ucz ** i for i in range(N + 1)])
    M_wer = np.column_stack([u_wer ** i for i in range(N + 1)])
    M_stat = np.column_stack([u ** i for i in range(N + 1)])

    wsp_ucz = np.linalg.lstsq(M_ucz, y_ucz, rcond=None)[0]

    y_ucz_hat = np.dot(M_ucz, wsp_ucz)
    y_wer_hat = np.dot(M_wer, wsp_ucz)
    y_stat = np.dot(M_stat, wsp_ucz)
```

Obliczanie błedu kwadratowego

```
error_ucz = np.sum((y_ucz - y_ucz_hat) ** 2)
error_wer = np.sum((y_wer - y_wer_hat) ** 2)
```

Powyższy algorytm w sposób iteracyjny tworzy modele w postaci wielomianów o kolejnych stopniach. Stworzono modele od pierwszego stopnia do szóstego.

1.3.2. Otrzymane błędy modeli

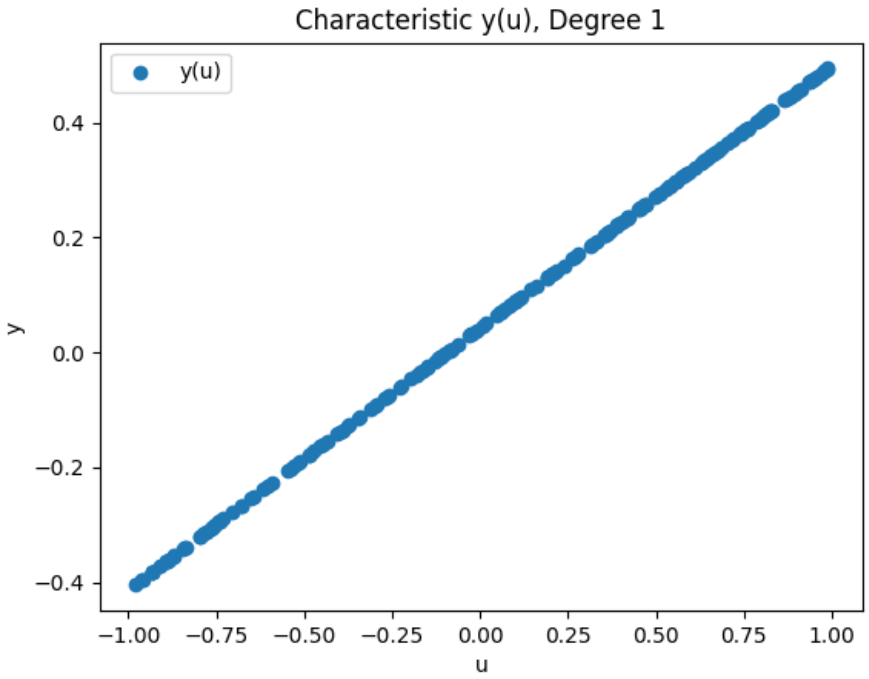
Stopień wielomianu	Błąd uczenia	Błąd weryfikacji
1	0.600011	0.649802
2	0.597620	0.635665
3	0.038560	0.036925
4	0.016299	0.013644
5	0.016146	0.014108
6	0.015800	0.014962

Tab. 1.1. Błedy identyfikacji modeli o róznych stopniach dla danych uczących i weryfikujących

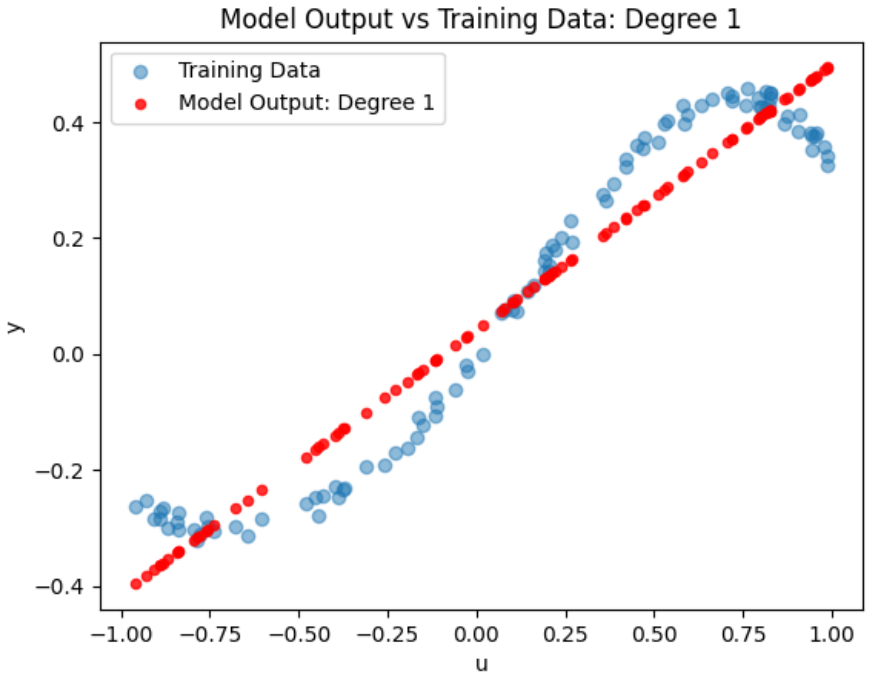
Należy zauważyć, że już dla wielomianu stopnia trzeciego jakość modelowania znacznie się poprawiła w porównaniu do modelu liniowego. Dla danych uczących model poprawia się dla coraz większego stopnia wielomianu. Należy pamiętać jednak, że wiąże się to również z dłuższym czasem obliczania modelu. Widać również, że błąd weryfikacyjny dla modelu stopnia szóstego jest wiekszy niż błąd modelu stopnia piątego. Zatem najlepszym modelem statycznym jest wielomian piątego stopnia.

1.3.3. Wykresy modeli statycznych

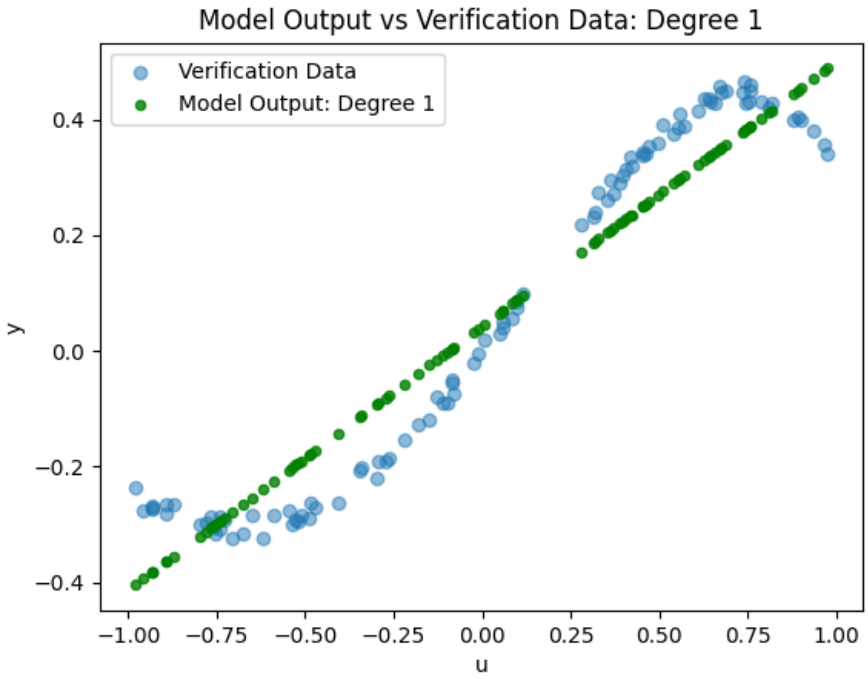
Model stopnia pierwszego:



Rys. 1.7. Charakterystyka statyczna, st. 1

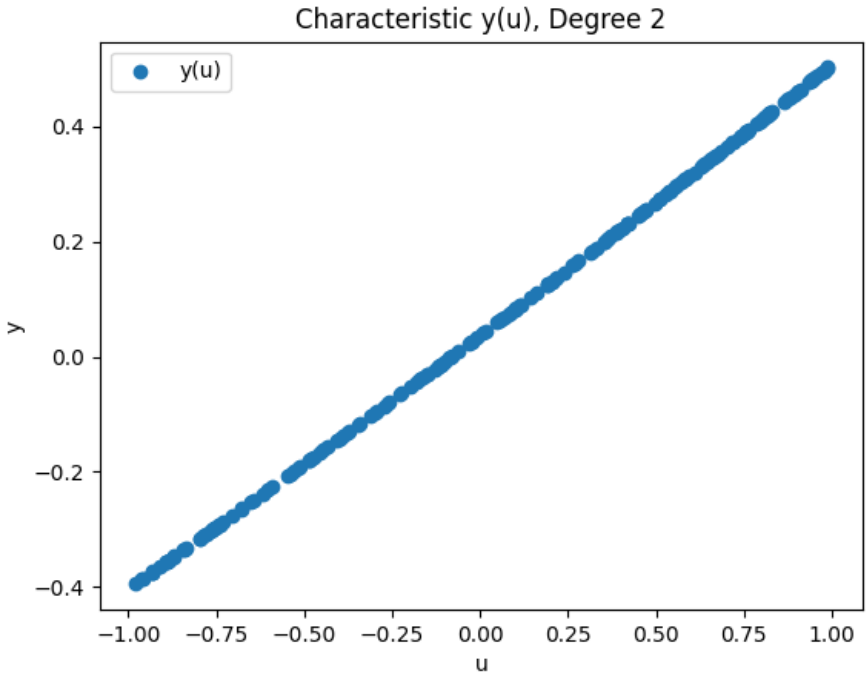


Rys. 1.8. Wyjście modelu na tle dancyh uczących, st. 1

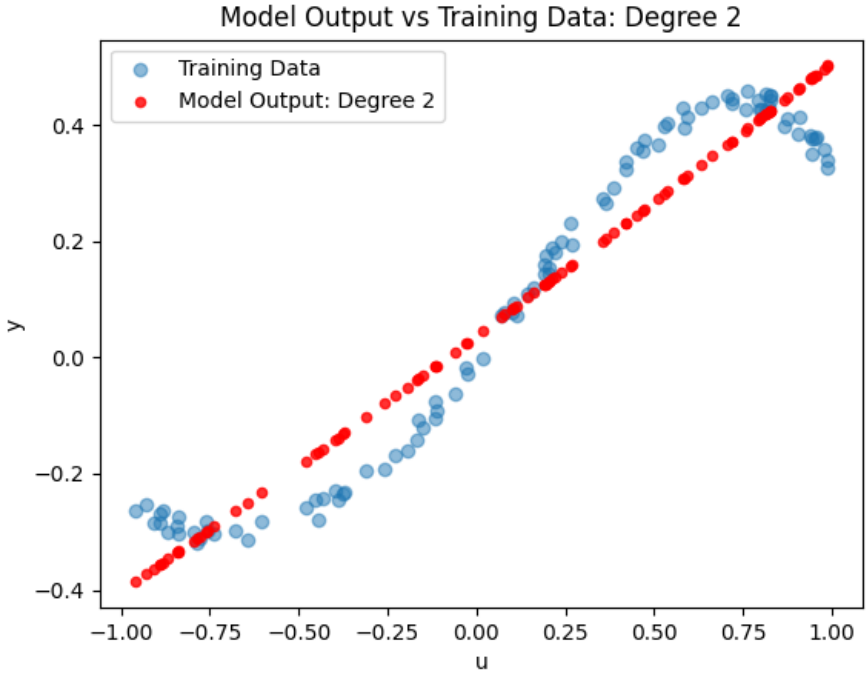


Rys. 1.9. Wyjście modelu na tle dancyh weryfikacyjnych, st. 1

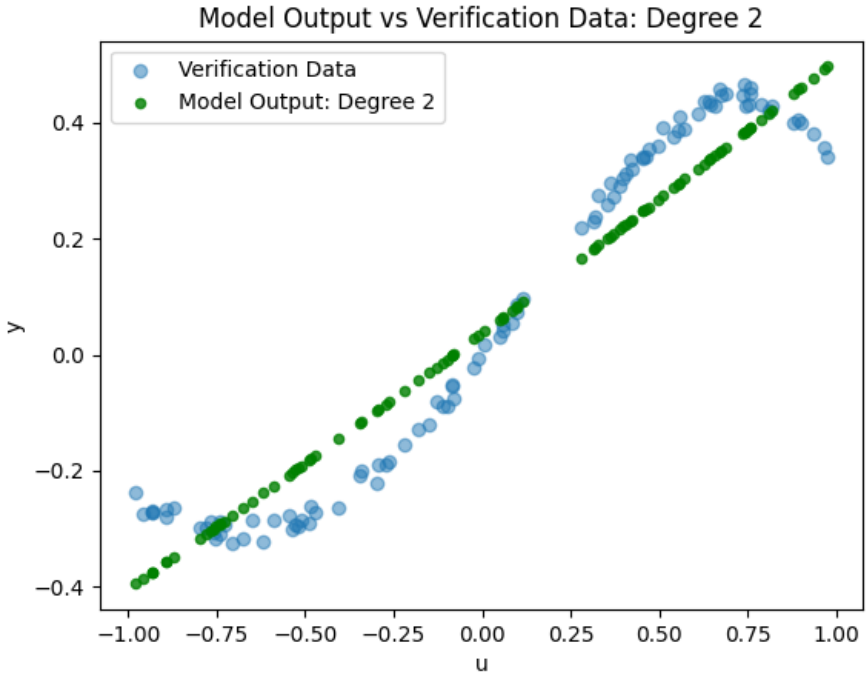
Model stopnia drugiego:



Rys. 1.10. Charakterystyka statyczna, st. 2

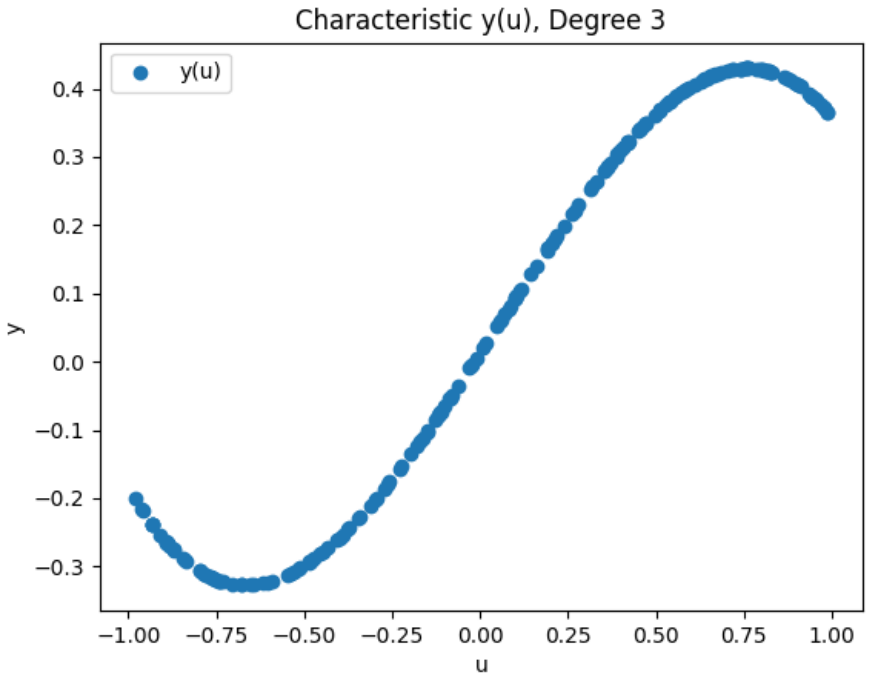


Rys. 1.11. Wyjście modelu na tle dancyh uczących, st. 2

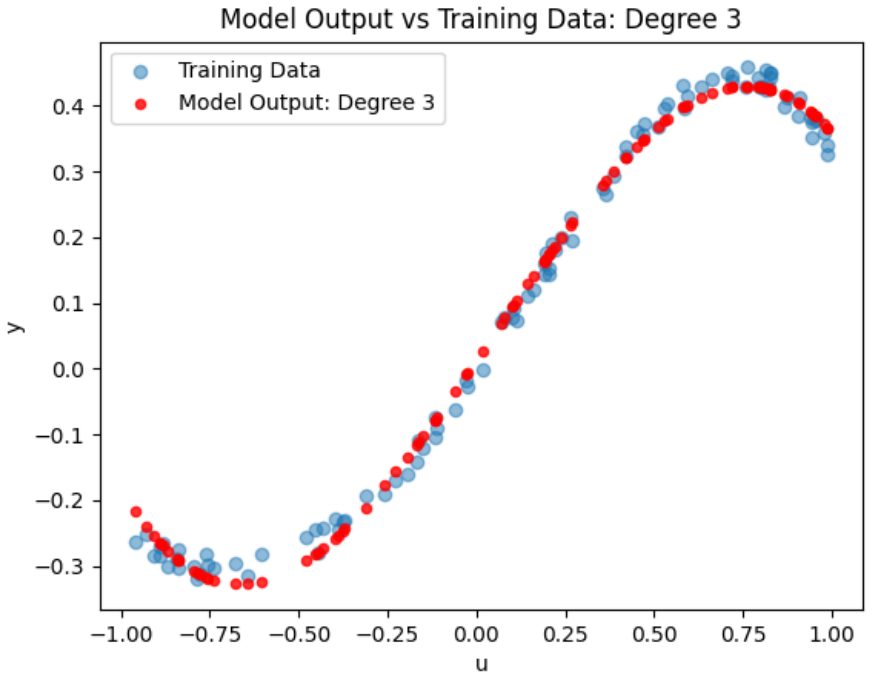


Rys. 1.12. Wyjście modelu na tle dancyh weryfikacyjnych, st. 2

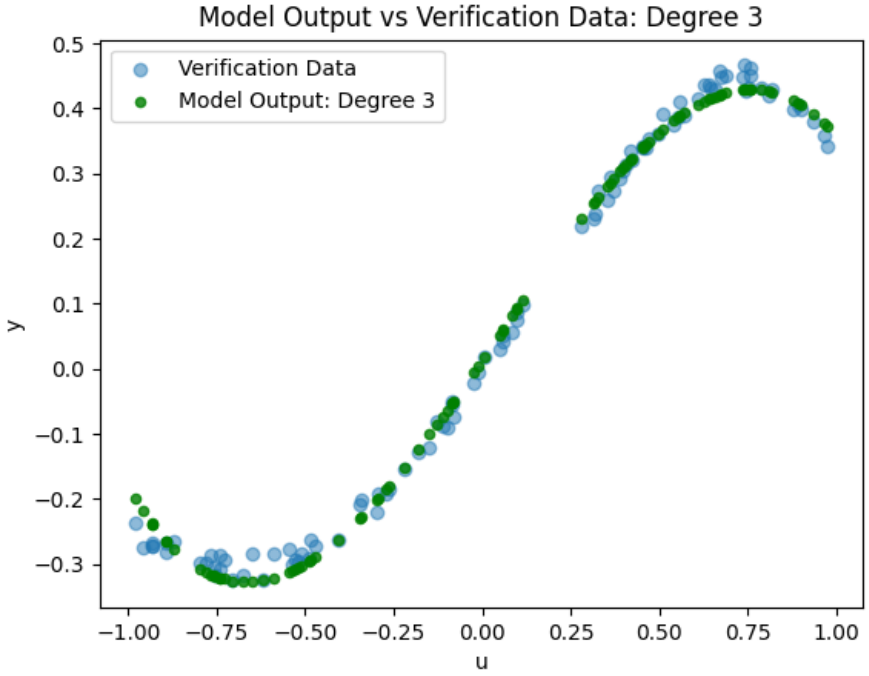
Model stopnia trzeciego:



Rys. 1.13. Charakterystyka statyczna, st. 3

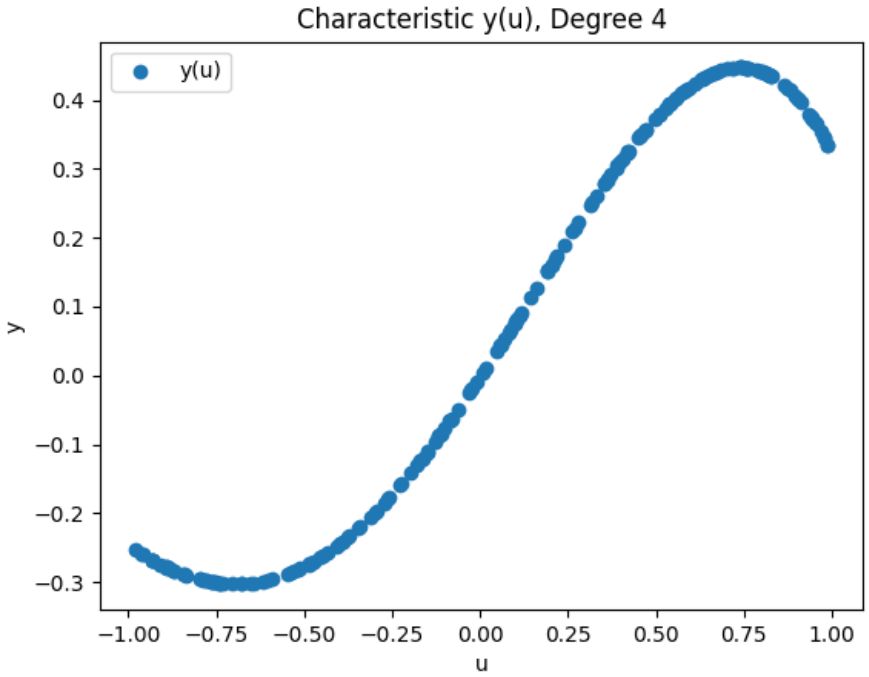


Rys. 1.14. Wyjście modelu na tle dancyh uczących, st. 3

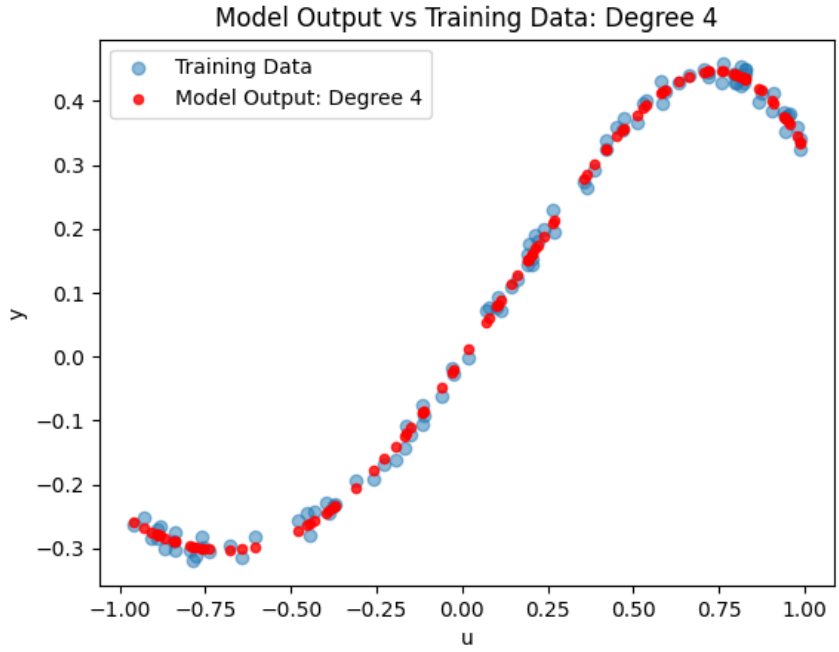


Rys. 1.15. Wyjście modelu na tle dancyh weryfikacyjnych, st. 3

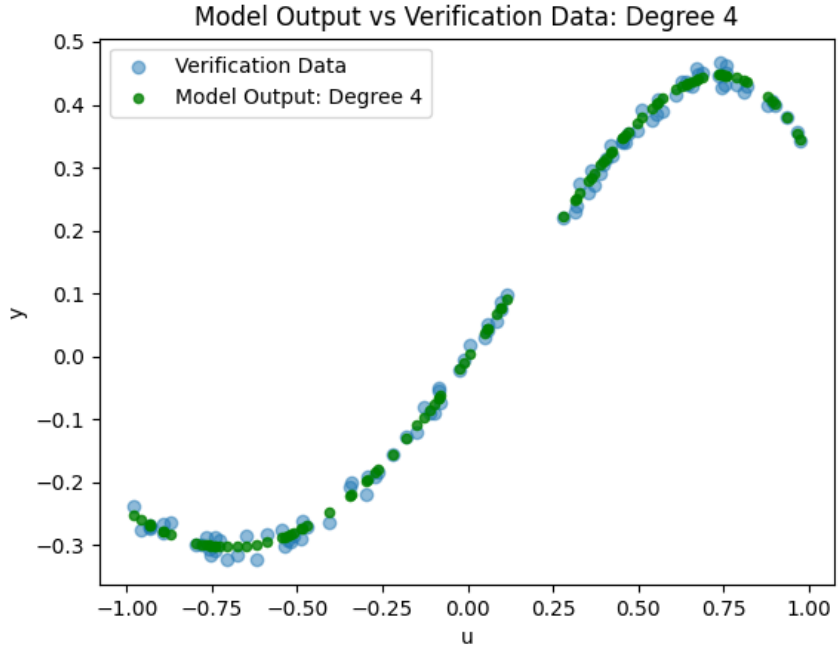
Model stopnia czwartego:



Rys. 1.16. Charakterystyka statyczna, st. 4

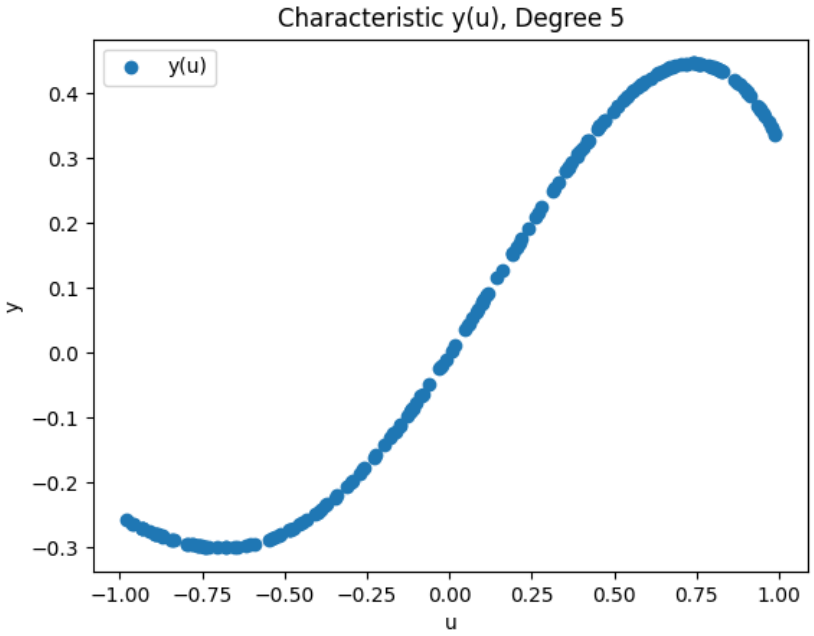


Rys. 1.17. Wyjście modelu na tle dancyh uczących, st. 4

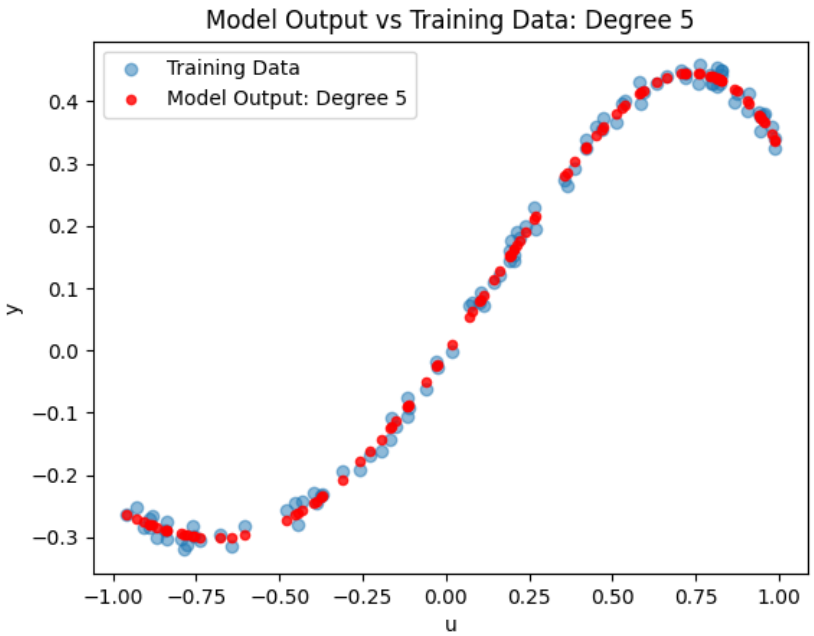


Rys. 1.18. Wyjście modelu na tle dancyh weryfikacyjnych, st. 4

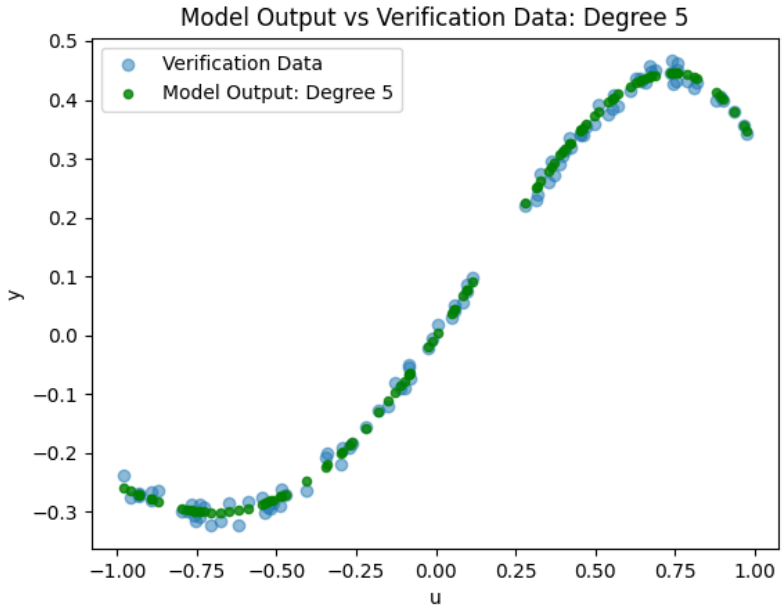
Model stopnia piątego:



Rys. 1.19. Charakterystyka statyczna, st. 5

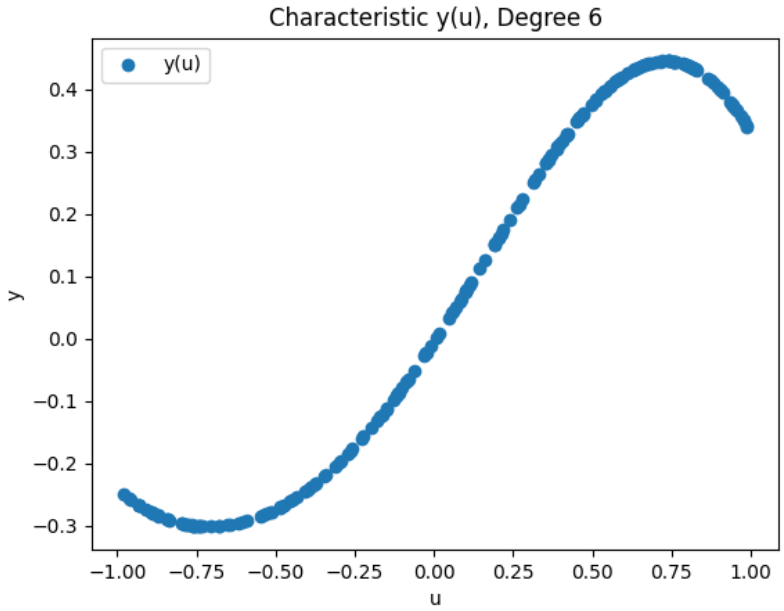


Rys. 1.20. Wyjście modelu na tle dancyh uczących, st. 5

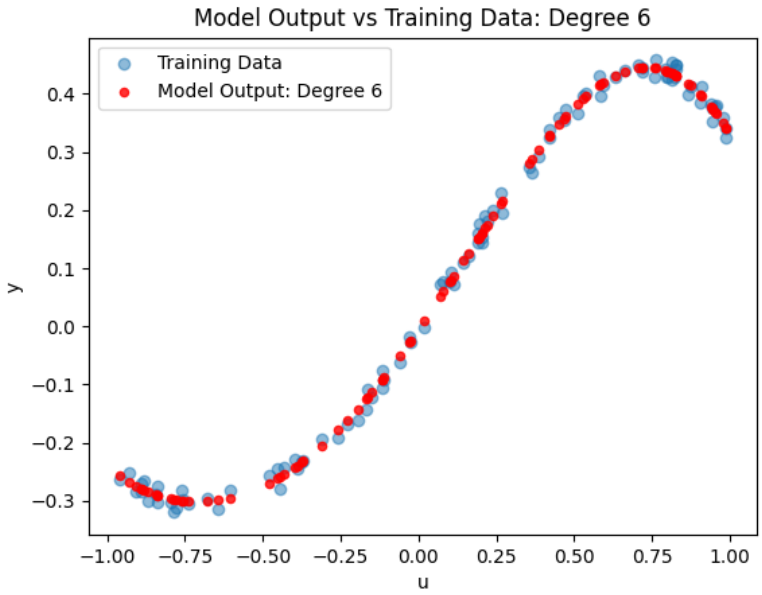


Rys. 1.21. Wyjście modelu na tle dancyh weryfikacyjnych, st. 5

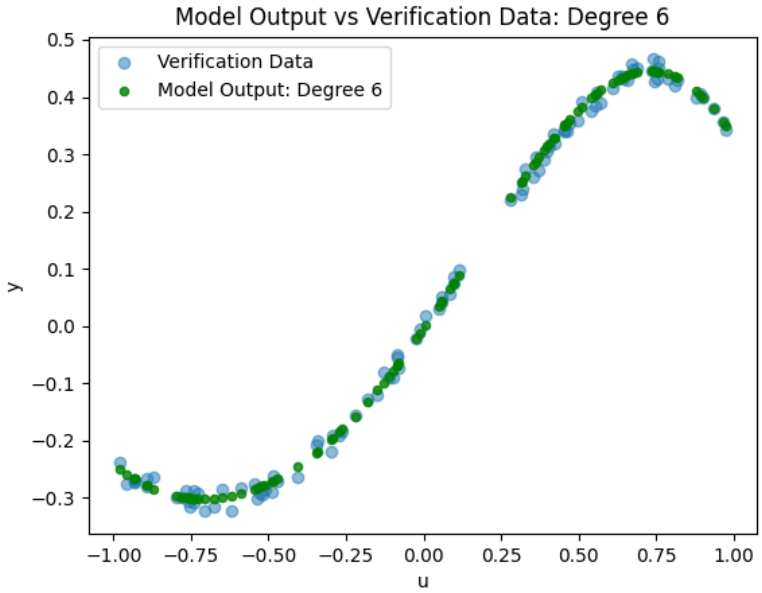
Model stopnia szóstego:



Rys. 1.22. Charakterystyka statyczna, st. 6



Rys. 1.23. Wyjście modelu na tle dancyh uczących, st. 6



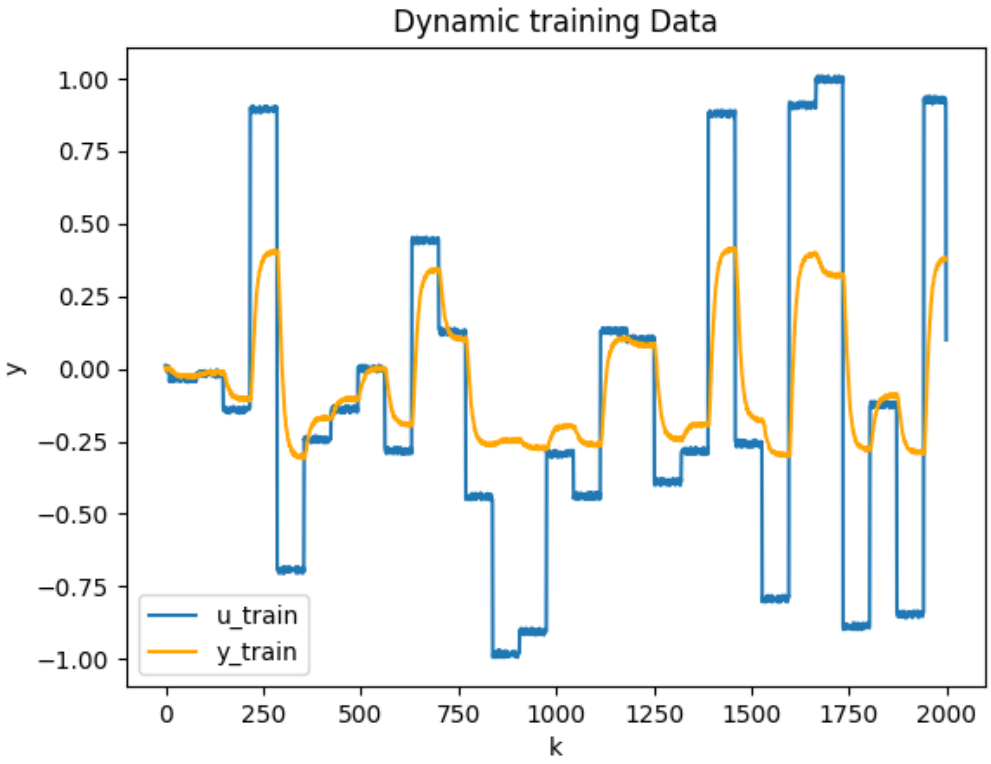
Rys. 1.24. Wyjście modelu na tle dancyh weryfikacyjnych, st. 6

Jak widać wyznaczone błędy pokrywają się z tym co można zaobserwować na rysunkach. Już dla modelu stopnia trzeciego widać, że bardzo dobrze identyfikuje on dane i dla kolejnych stopni wielomianu widać coraz mniejszą poprawę. Podsumowując najlepszym modelem do identyfikacji tego procesu jest wielomian stopnia piątego, ponieważ jego błąd dla zbioru weryfikującego jest najmniejszy, a złożoność obliczeniowa jeszcze nie jest bardzo duża.

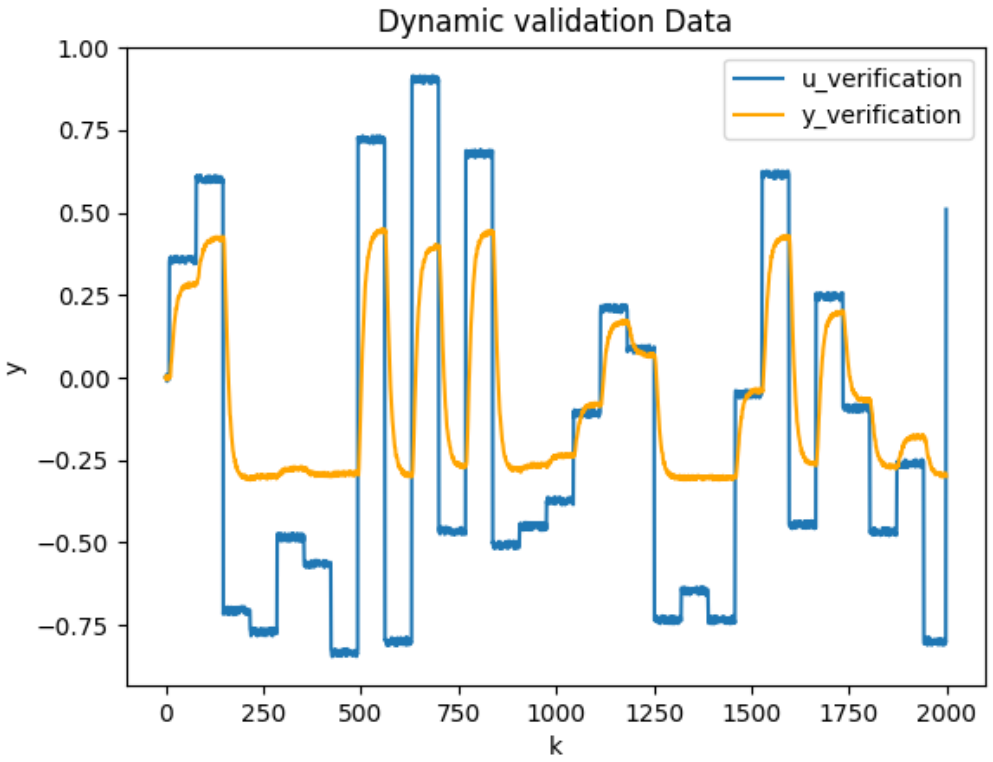
2. Identyfikacja modeli dynamicznych

2.1. Podział danych

Pobrano wcześniej przygotowane zbiory danych: uczący i weryfikujący.



Rys. 2.1. Dane uczące



Rys. 2.2. Dane weryfikujące

2.2. Wyznaczanie modeli dynamicznych rzędu pierwszego drugiego i trzeciego

Zadanie zrealizowano korzystając z metody najmniejszych kwadratów w trybie bez rekurencji i z rekurencją. Wyznaczane modele są postaci

$$y(k) = \sum_{i=1}^{N} b_i u(k-i) + \sum_{i=1}^{N} a_i y(k-1) \tag{2.1}$$

gdzie N to rząd.

2.2.1. Implementacja algorytmu w python

Wzory na kolejne rzędy wypisano ręcznie. Poniżej kod dla pierwszego rzędu.

```
    # wybór rzędu
    if n == 1:

        # inicjalizacja wektorów trzymających dane
        y_ucz_hat = []
        y_wer_hat = []
        y_ucz_hat_rek = []
        y_wer_hat_rek = []

        # dodanie do wyjść modelu tylu pierwszych próbek ile wynosi stopień
        y_ucz_hat.append(y_ucz[0])
        y_ucz_hat_rek.append(y_ucz[0])
        y_wer_hat.append(y_ucz[0])
        y_wer_hat_rek.append(y_ucz[0])

        # stworzenie macierzy M i policzenie współczynników modelu
        M_ucz = np.column_stack((u_ucz[:-1], y_ucz[:-1]))
        wsp_ucz = np.linalg.lstsq(M_ucz, y_ucz[1:], rcond=None)[0]

        # obliczanie kolejnych wyjść modelu
        for i in range(len(y_ucz)-n):
            y_ucz_hat.append(u_ucz[i]*wsp_ucz[0] + y_ucz[i]*wsp_ucz[1])
            y_wer_hat.append(u_wer[i]*wsp_ucz[0] + y_wer[i]*wsp_ucz[1])
            y_ucz_hat_rek.append(u_ucz[i]*wsp_ucz[0]
                                 + y_ucz_hat_rek[i]*wsp_ucz[1])
            y_wer_hat_rek.append(u_wer[i]*wsp_ucz[0]
                                 + y_wer_hat_rek[i]*wsp_ucz[1])

    # obliczanie i wypisywanie błedu
    error_ucz = np.sum((y_ucz - y_ucz_hat) ** 2)
    error_wer = np.sum((y_wer - y_wer_hat) ** 2)
    error_ucz_rek = np.sum((y_ucz - y_ucz_hat_rek) ** 2)
    error_wer_rek = np.sum((y_wer - y_wer_hat_rek) ** 2)
```

2.2.2. Otrzymane błędy modelu

Poniższa tabela przedstawia wyliczone błędy kwadratowe dla modelu dla danych uczących oraz weryfikujących w wariancie modelu z rekurencją i bez rekurencji.

Rząd modelu	Błędy bez rekurencji		Błędy z rekurencją	
	Ucznie	Weryfikacja	Ucznie	Weryfikacja
1	0.11417177603289931	0.13855993137490835	12.31461547788733	16.400415390515057
2	0.09770875876449303	0.11161317069434919	10.836014360595343	14.669134202652549
3	0.0801926862040162	0.08650869659689536	9.486345062848919	13.164506344344247

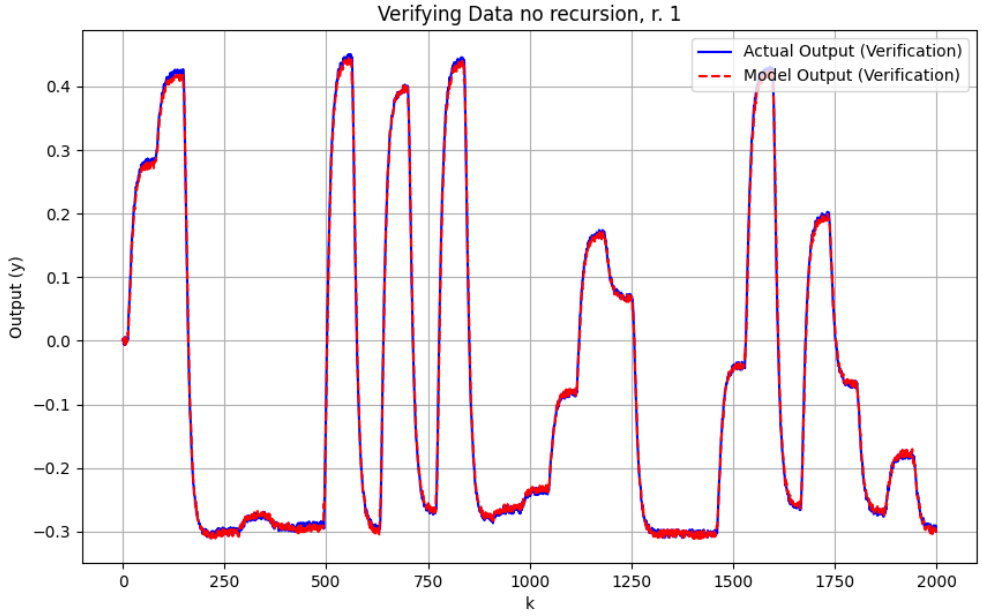
Tab. 2.1. Otrzymane błędy modelu

Najlepiej sprawdza się model o rzędzie dynamiki 3 w trybie bez rekurencji dla danych uczących i weryfikujących. W trybie z rekurencją również najlepiej sprawdza się rząd dynamiki 3.

2.2.3. Wykresy modeli dynamicznych

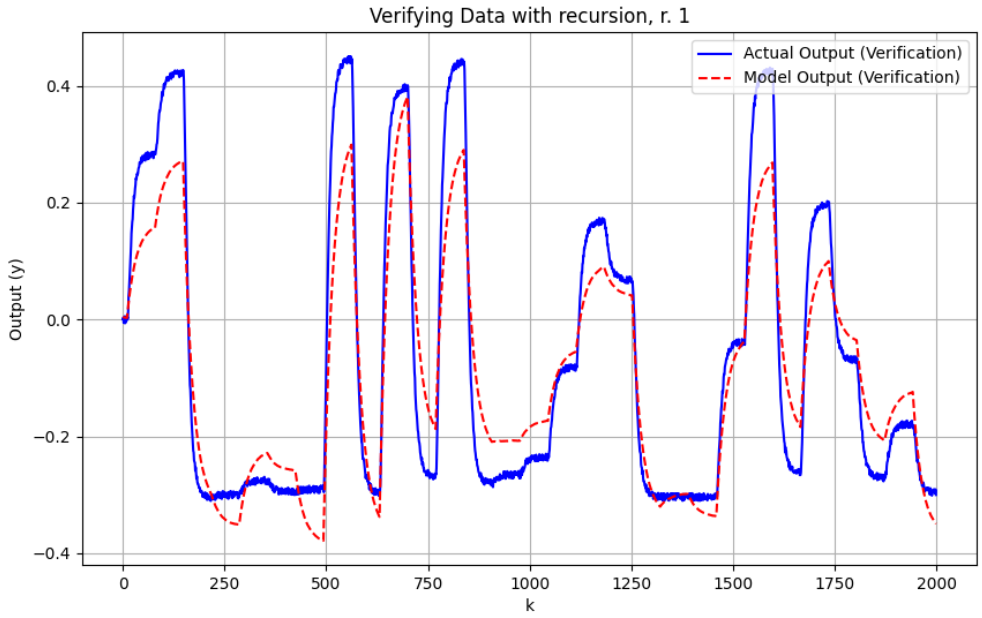
Rząd 1:

Rys. 2.3. Wyjście modelu na tle danych uczących, rząd 1, bez rekurencji



Rys. 2.4. Wyjście modelu na tle danych weryfikujących, rząd 1, bez rekurencji

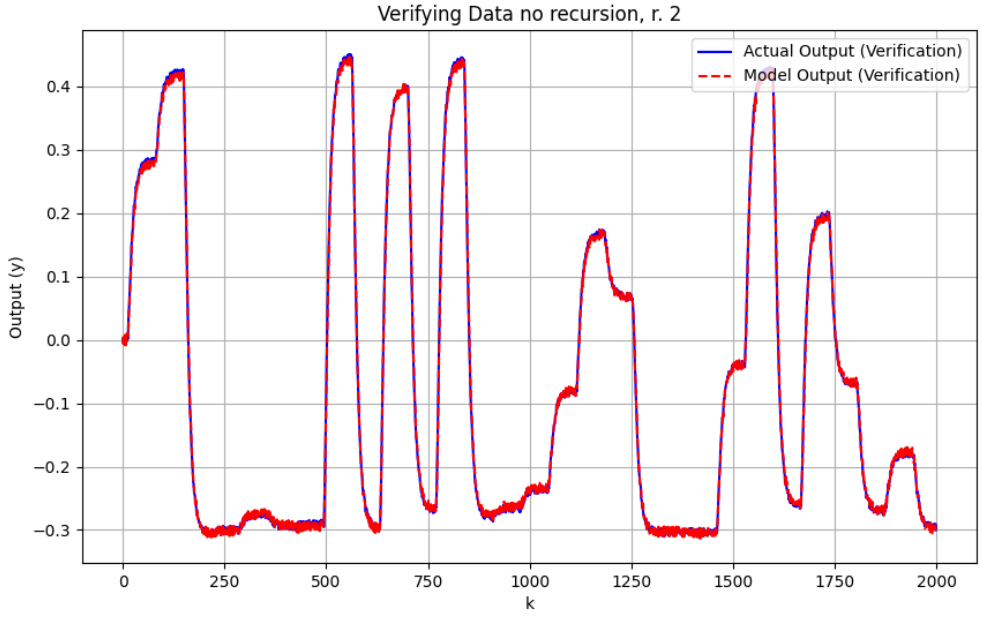
Rys. 2.5. Wyjście modelu na tle danych uczących, rząd 1, z rekurencją



Rys. 2.6. Wyjście modelu na tle danych weryfikujących, rząd 1, z rekurencją

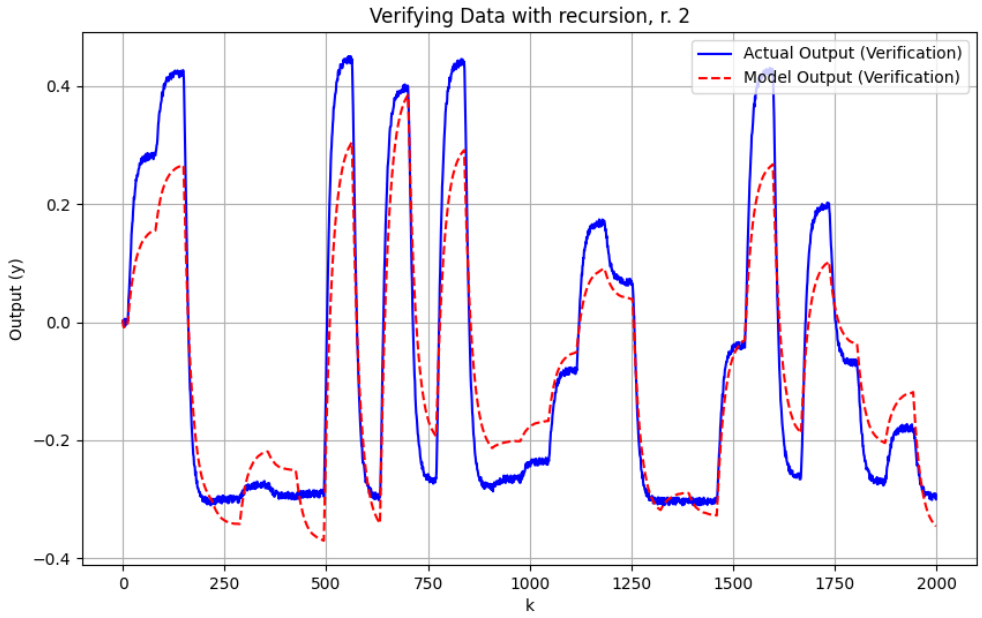
Rząd 2:

Rys. 2.7. Wyjście modelu na tle danych uczących, rząd 2, bez rekurencji



Rys. 2.8. Wyjście modelu na tle danych weryfikujących, rząd 2, bez rekurencji

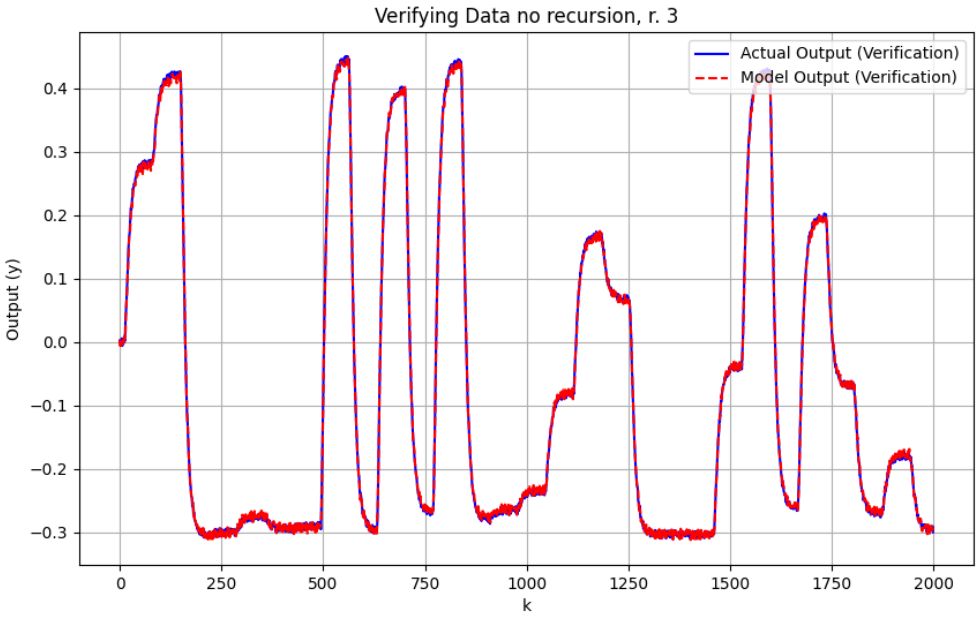
Rys. 2.9. Wyjście modelu na tle danych uczących, rząd 2, z rekurencją



Rys. 2.10. Wyjście modelu na tle danych weryfikujących, rząd 2, z rekurencją

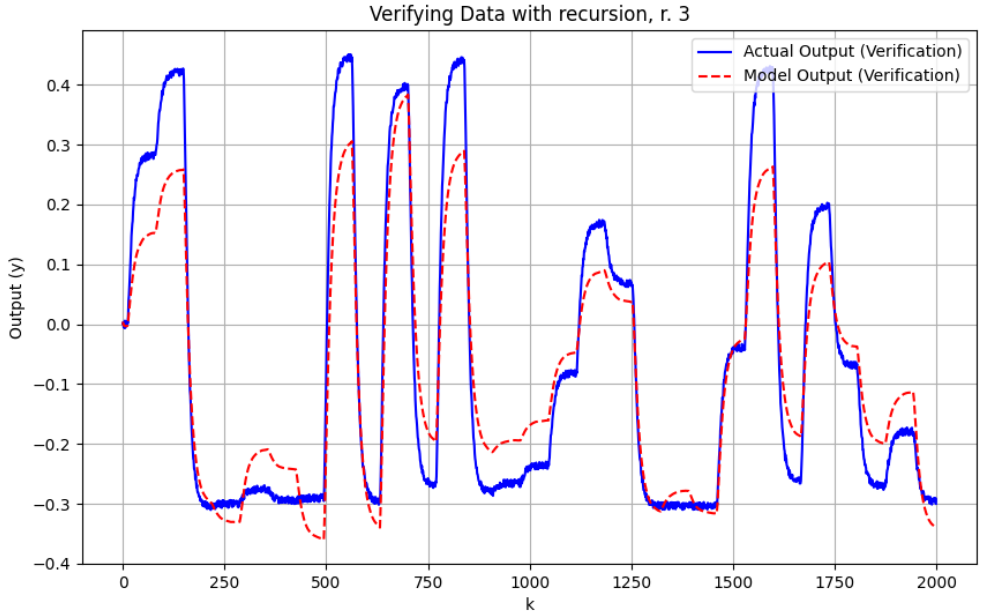
Rząd 3:

Rys. 2.11. Wyjście modelu na tle danych uczących, rząd 3, bez rekurencji



Rys. 2.12. Wyjście modelu na tle danych weryfikujących, rząd 3, bez rekurencji

Rys. 2.13. Wyjście modelu na tle danych uczących, rząd 3, z rekurencją



Rys. 2.14. Wyjście modelu na tle danych weryfikujących, rząd 3, z rekurencją

Zgodnie z wyznaczonymi błędami, najlepiej sprawdzają się modele bez rekurencji. Są to na tyle dobre odwzorowania, że ciężko zauważyć różnicę między wykresem modelu a faktycznego procesu dynamicznego. Z punktu widzenia dokładności, najlepszym modelem rekurencyjnym jest model trzeciego rzędu.

2.3. Wyznaczanie szeregów wielomianowych dynamicznych modeli nieliniowych

Metodą najmniejszych kwadratów wyznaczono szeregi wielomianowych dynamicznych modeli nieli niowych. Rozważono modele o różnym rzędzie dynamiki i strukturze nieliniowości.

2.3.1. Implementacja algorytmu

W tym zadaniu stworzono algorytm który pozwala na wyznaczenie szeregów wielomianowych dynamicznych modeli nieliniowych o dowolnym stopni dynamiki i strukturze nieliniowości.

```
def zad2_c(u_ucz, u_wer, y_ucz, y_wer, n_dyn, n_poly):
    # inicjalizacja kontenerów
    y_ucz_hat = []
    y_wer_hat = []
    y_ucz_hat_rek = []
    y_wer_hat_rek = []

    # dodanie pierwszych wyrazów wyjśc modelu
    for i in range(n_dyn):
        y_ucz_hat.append(y_ucz[i])
        y_wer_hat.append(y_ucz[i])
        y_ucz_hat_rek.append(y_ucz[i])
        y_wer_hat_rek.append(y_ucz[i])

    # tworzenie macierzy M i współczynników modelu
    columns = []
    for i in range(1, n_dyn+1):
        for j in range(1, n_poly+1):
            columns.append(u_ucz[n_dyn-i:-i]**j)
            columns.append(y_ucz[n_dyn-i:-i]**j)
    M_ucz = np.column_stack(tuple(columns))
    wsp_ucz = np.linalg.lstsq(M_ucz, y_ucz[n_dyn:], rcond=None)[0]

    #obliczanie kolejnych wyjść modelu
    for i in range(len(y_ucz)-n_dyn):
        yk = 0
        yk_wer = 0
        yk_rek = 0
        yk_wer_rek = 0
        counter = 0
        for j in range(1, n_dyn+1):
            for k in range(1, n_poly+1):
                yk += wsp_ucz[counter]*u_ucz[i+n_dyn-j]**k
                        + wsp_ucz[counter+1]*y_ucz[i+n_dyn-j]**k
                yk_wer += wsp_ucz[counter]*u_wer[i+n_dyn-j]**k
                        + wsp_ucz[counter+1]*y_wer[i+n_dyn-j]**k
                yk_rek += wsp_ucz[counter]*u_ucz[i+n_dyn-j]**k
                        + wsp_ucz[counter+1]*y_ucz_hat_rek[i+n_dyn-j]**k
                yk_wer_rek += wsp_ucz[counter]*u_wer[i+n_dyn-j]**k
                        + wsp_ucz[counter+1]*y_wer_hat_rek[i+n_dyn-j]**k
                counter += 2

        y_ucz_hat.append(yk)
        y_wer_hat.append(yk_wer)
        y_ucz_hat_rek.append(yk_rek)
        y_wer_hat_rek.append(yk_wer_rek)
```

```
    # obliczanie i wypisywanie błedu
    error_ucz = np.sum((y_ucz - y_ucz_hat) ** 2)
    error_wer = np.sum((y_wer - y_wer_hat) ** 2)
    error_ucz_rek = np.sum((y_ucz - y_ucz_hat_rek) ** 2)
    error_wer_rek = np.sum((y_wer - y_wer_hat_rek) ** 2)

    print(error_ucz, error_wer)
    print(error_ucz_rek, error_wer_rek)
```

2.3.2. Otrzymane błędy modelu

Poniższa tabela przedstawia wyliczone błędy kwadratowe dla modelu dla danych uczących oraz weryfikujących w wariancie modelu z rekurencją i bez rekurencji dal róznych rzędów i stopni wielomianu.

Rząd modelu	Stopień wielomianu	Błędy bez rekurencji		Błędy z rekurencją	
		Uczenie	Weryfikacja	Uczenie	Weryfikacja
1	1	0.1142	0.1386	12.3146	16.4004
1	2	0.1080	0.1451	11.3313	14.9039
1	3	0.0829	0.0944	1.1431	1.4123
2	1	0.0977	0.1116	10.8360	14.6691
2	2	0.0917	0.1129	9.3337	13.1134
2	3	0.0614	0.0661	0.5020	0.6829
3	1	0.0802	0.0865	9.4863	13.1622
3	2	0.0769	0.0888	8.2287	11.7060
3	3	0.0506	0.0530	0.3232	0.5132

Tab. 2.2. Data table for model order, polynomial degree, and corresponding errors

Jak widać z powyższej tabeli nie udało się znaleźć takiego zestawy parametrów dla którego błędy byłyby optymalne. Widać jednak, że im większy stopień dynamiki i stopień wielomianu tym lepsza dokładność. Stopień wielomianu ma większy wpływ na dokładość od rzędu modelu.

Przeprowadzono dodatkowy eksperyment w celu znalezienia najlepszego rzędu dynamiki i stopnia wielomianu w kontekscie dokładności modelu.

```
best_error_rek = float("inf")
best_rek = []
best_error = float("inf")
best = []
for i in range(1,30):
    for j in range(1,30):
        error_ucz, error_wer, error_ucz_rek, error_wer_rek =
                zad2_c(u_ucz, u_wer, y_ucz, y_wer, i, j)
        if error_wer < best_error:
            best_error = error_wer
            best = (i,j)
        if error_wer_rek < best_error_rek:
            best_error_rek = error_wer_rek
            best_rek = (i,j)
print(best)
print(best_rek)
```

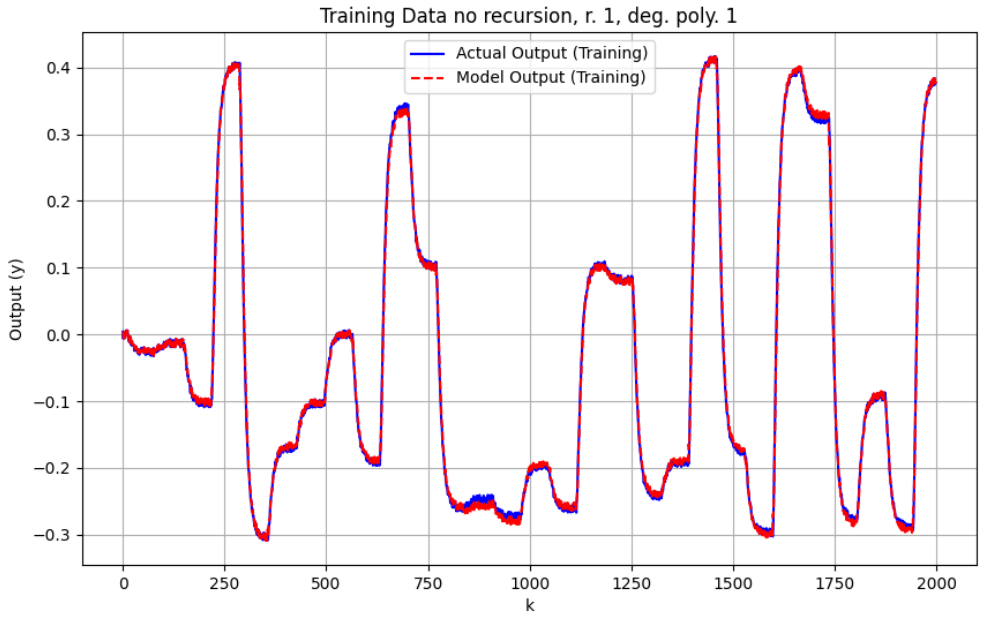
Tym sposobem znaleziono najlepszy zestaw parametrów i dla trybu rekurencyjnego był to rząd dynamiki równy 13, a stopień wielomianu równy 5. Błędy dla tego modelu w trybie rekurencyjnym wyniosły:

$$E_{ucz} = 0.028800660801364175,\ E_{wer}0.032276774182479484$$

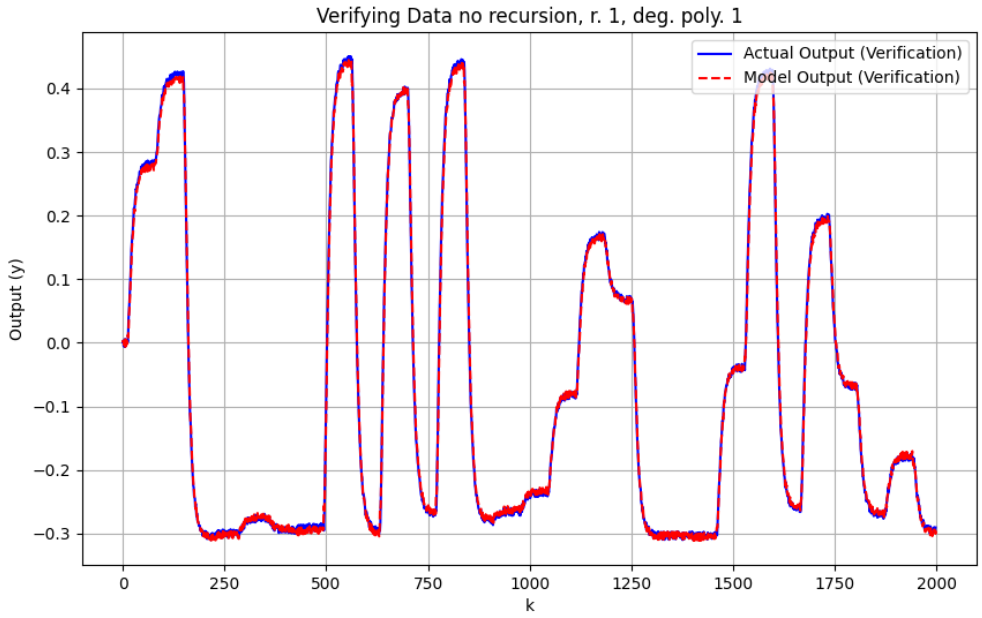
Można z tego wywnioskować, że stopień wielomianu ma większy wpływ na dokładność, jednak jeśli będzie za duży to pogorszy jakość modelu.

2.3.3. Wykresy modeli dynamicznych

Poniżej przedstawiono wykresy modeli dynamicznych dla rzędów 1 i 2, oraz stopni wielomianu 1 i 2.

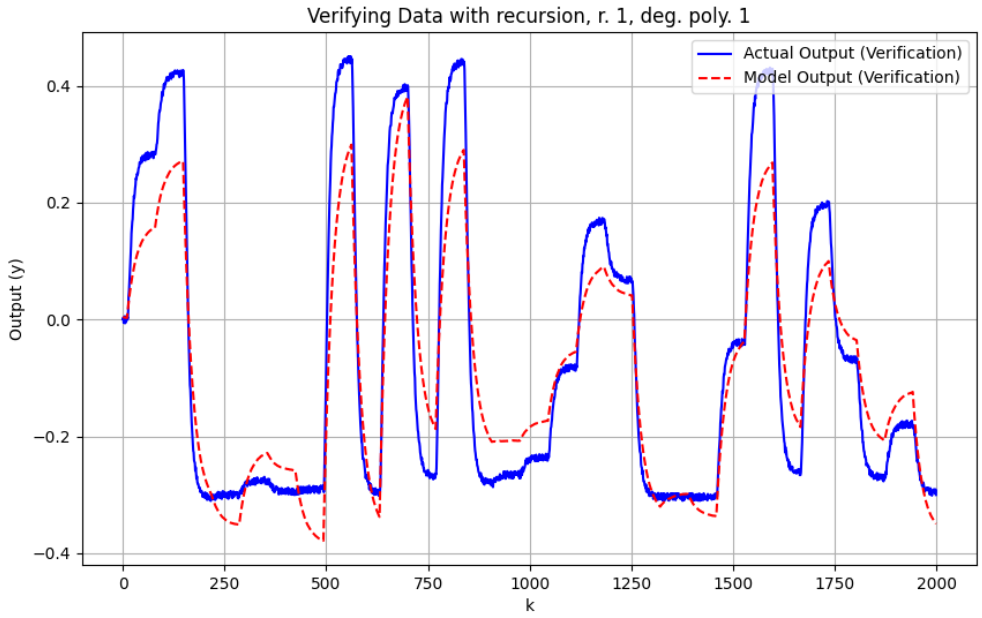


Rys. 2.15. Wyjście modelu na tle danych uczących, rząd 1, st. wielo. 1, bez rekurencji

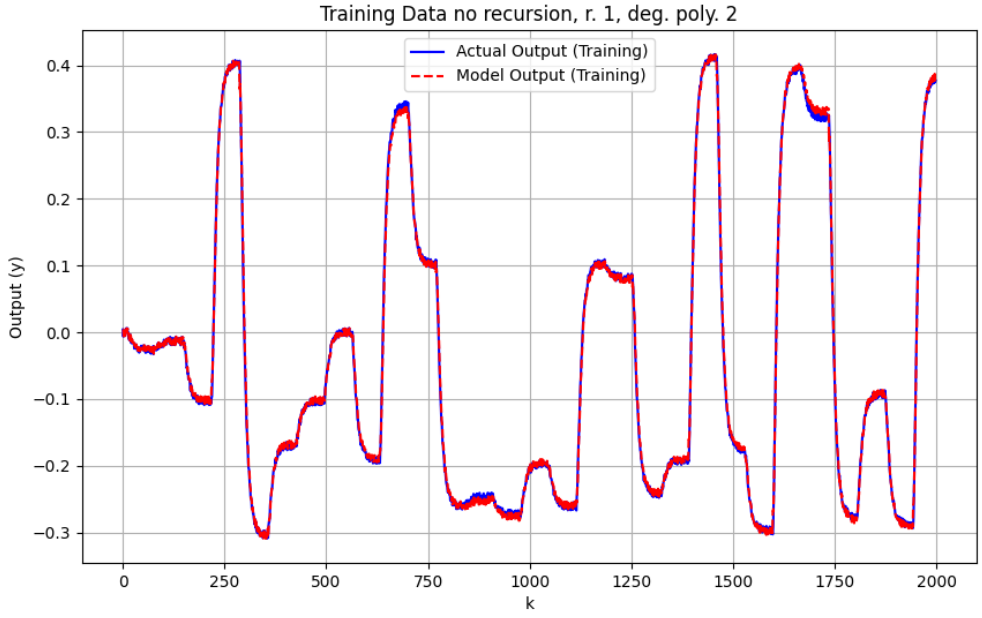


Rys. 2.16. Wyjście modelu na tle danych weryfikujących,rząd 1, st. wielo. 1, bez rekurencji

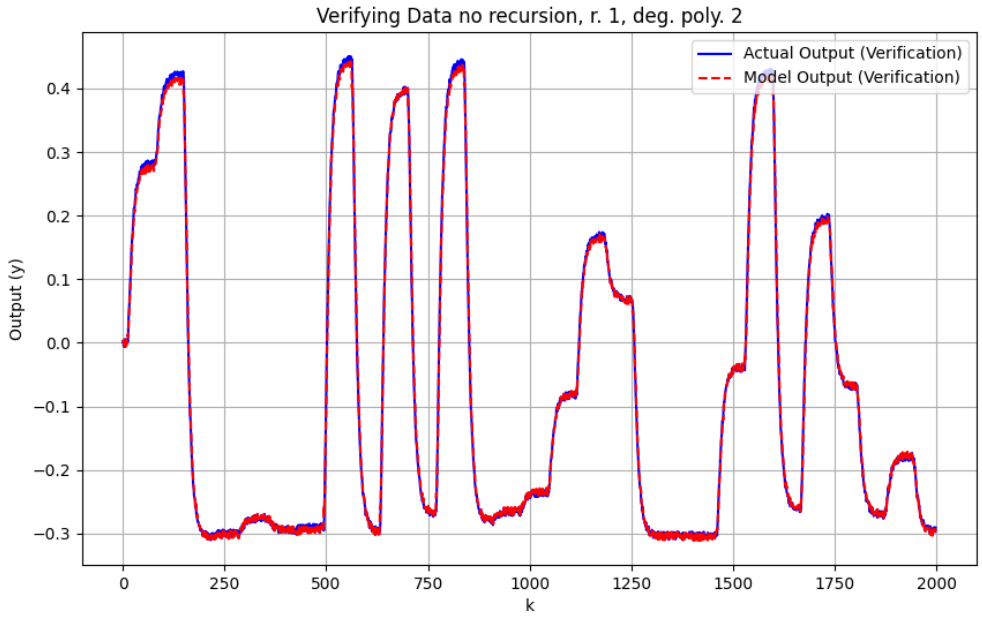
Rys. 2.17. Wyjście modelu na tle danych uczących, rząd 1, st. wielo. 1, z rekurencją



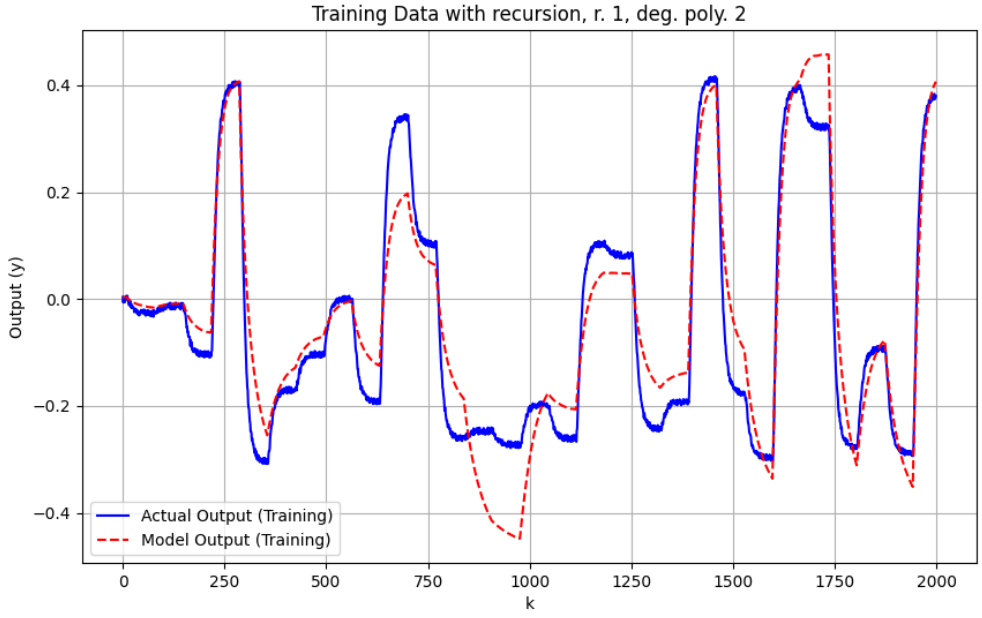
Rys. 2.18. Wyjście modelu na tle danych weryfikujących, rząd 1, st. wielo. 1, z rekurencją



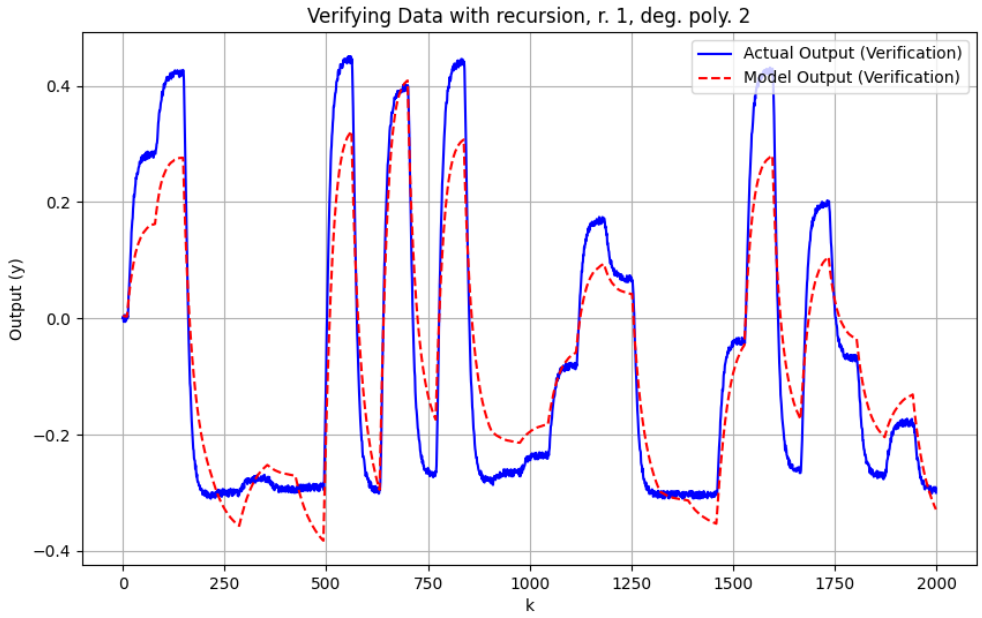
Rys. 2.19. Wyjście modelu na tle danych uczących, rząd 1, st. wielo. 2, bez rekurencji



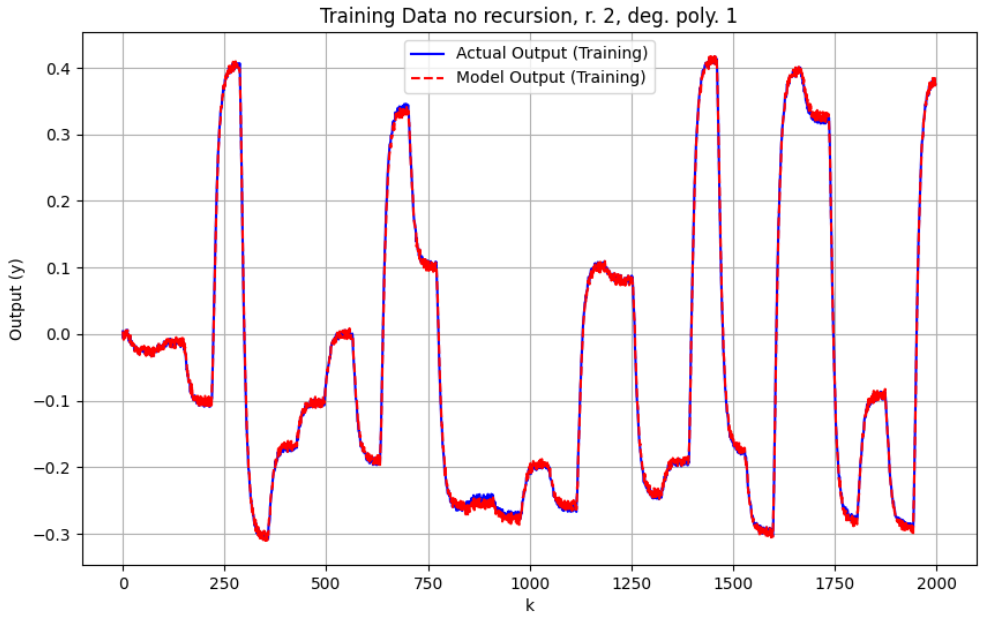
Rys. 2.20. Wyjście modelu na tle danych weryfikujących, rząd 1, st. wielo. 2, bez rekurencji



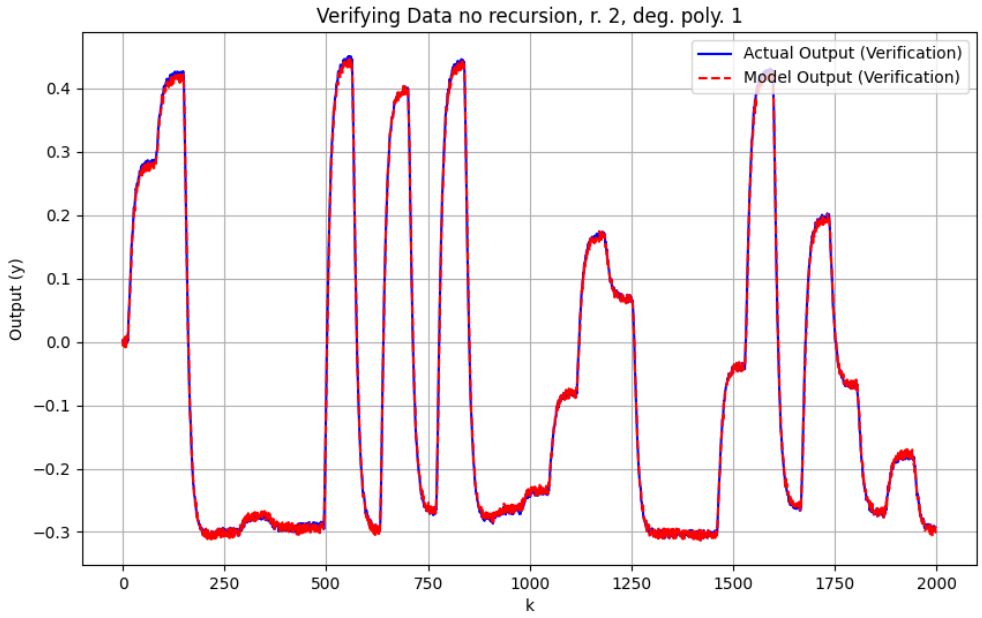
Rys. 2.21. Wyjście modelu na tle danych uczących, rząd 1, st. wielo. 2, z rekurencją



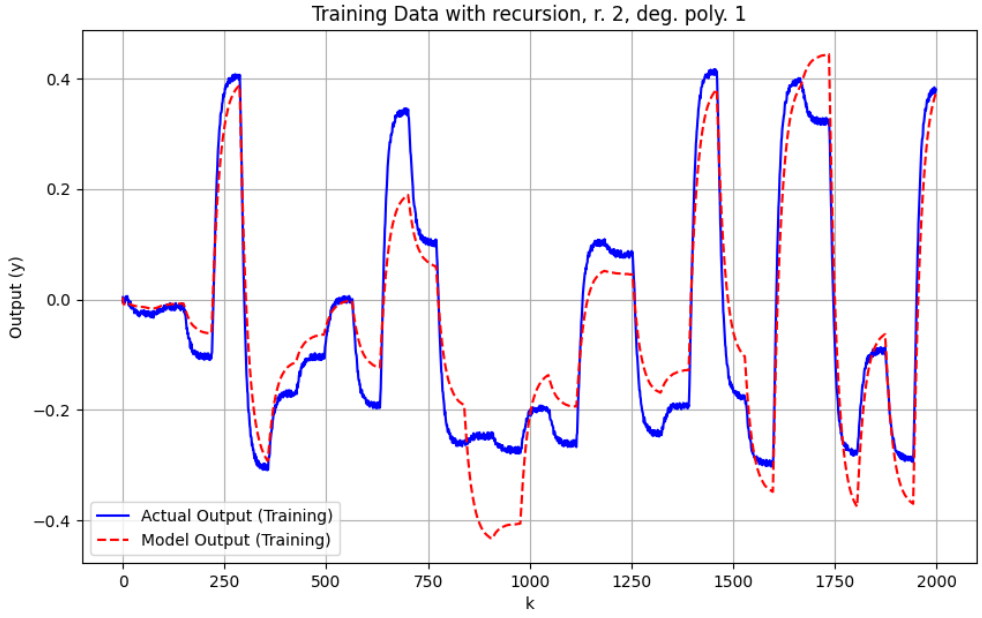
Rys. 2.22. Wyjście modelu na tle danych weryfikujących, rząd 1, st. wielo. 2, z rekurencją



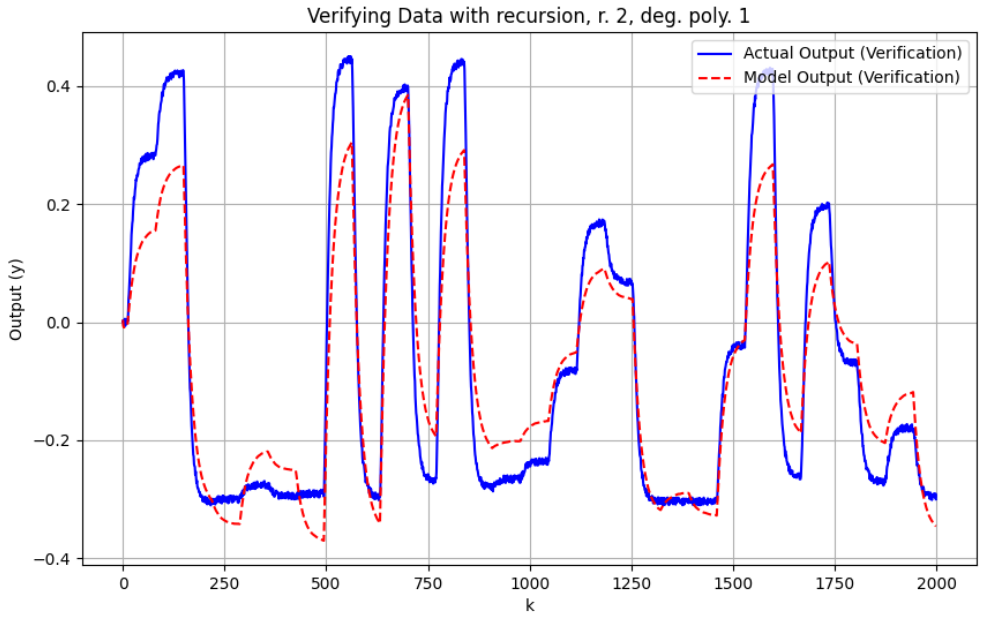
Rys. 2.23. Wyjście modelu na tle danych uczących, rząd 2, st. wielo. 1, bez rekurencji



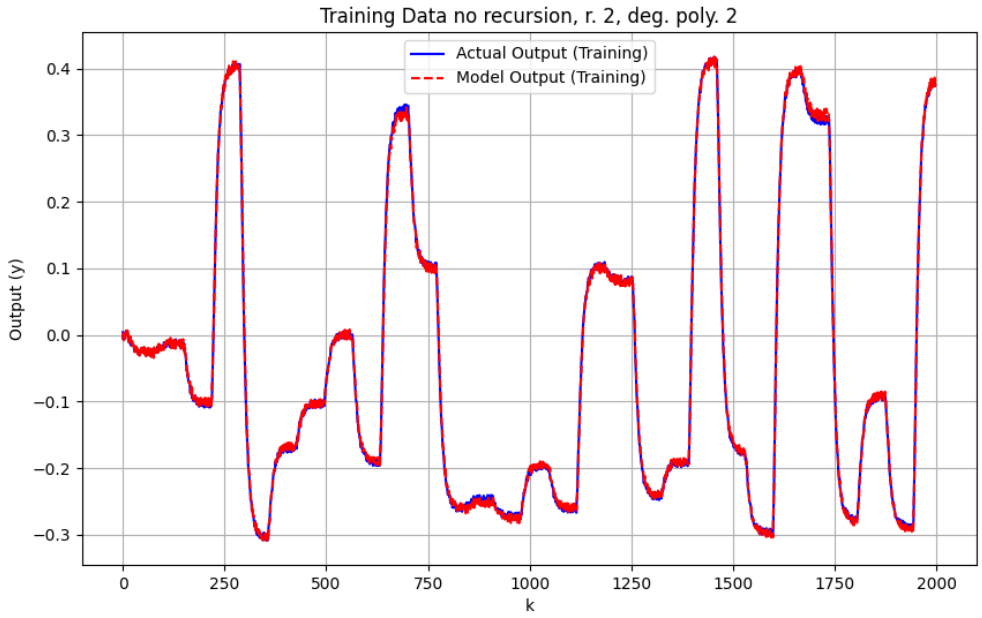
Rys. 2.24. Wyjście modelu na tle danych weryfikujących, rząd 2, st. wielo. 1, bez rekurencji



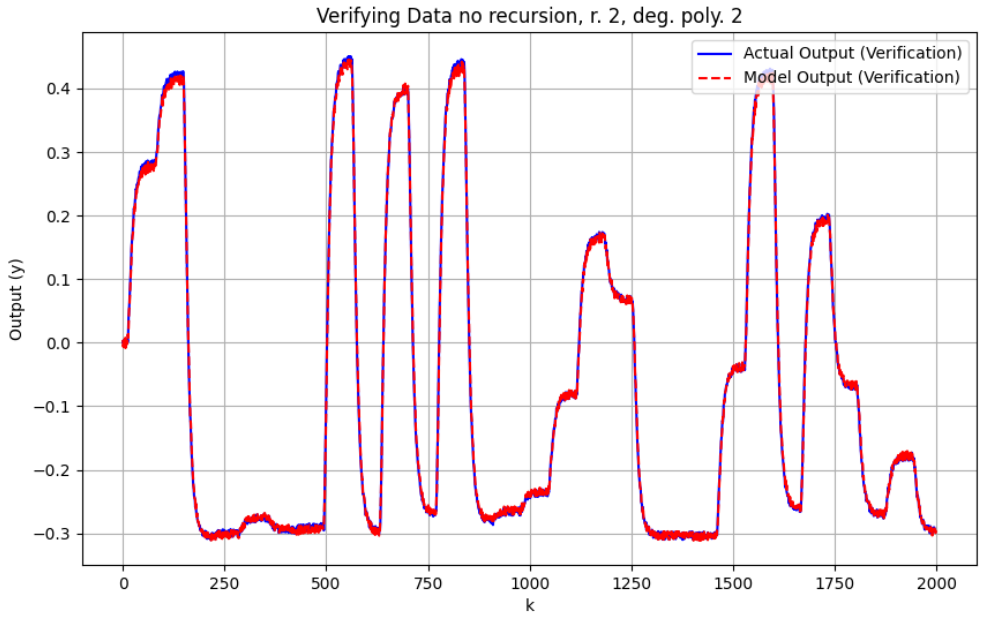
Rys. 2.25. Wyjście modelu na tle danych uczących, rząd 2, st. wielo. 1, z rekurencją



Rys. 2.26. Wyjście modelu na tle danych weryfikujących, rząd 2, st. wielo. 1, z rekurencją

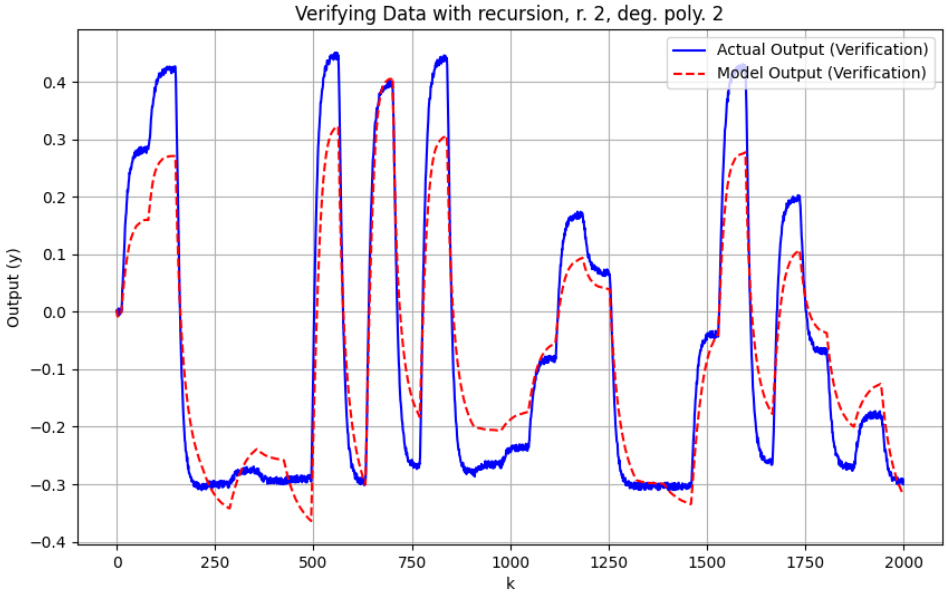


Rys. 2.27. Wyjście modelu na tle danych uczących, rząd 2, st. wielo. 2, bez rekurencji



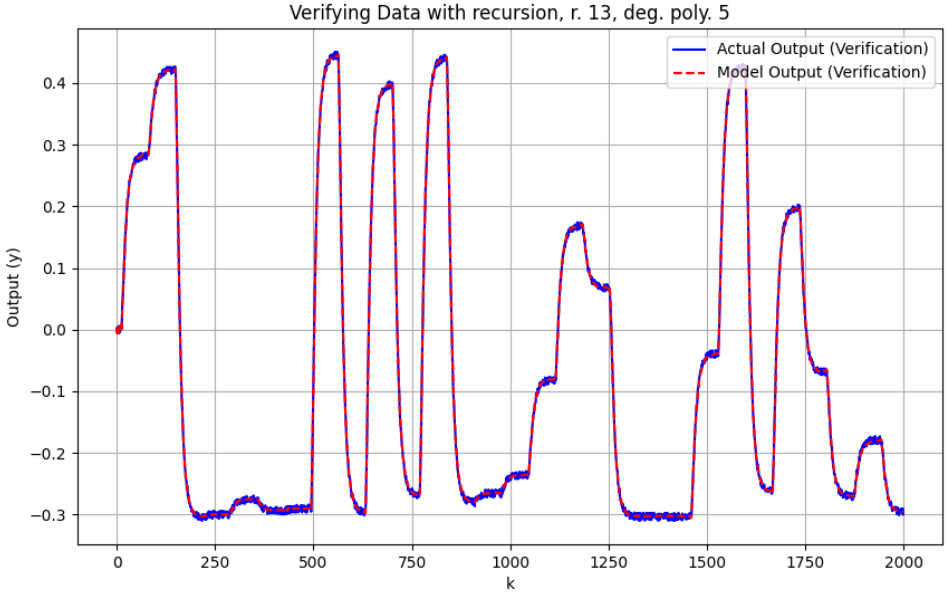
Rys. 2.28. Wyjście modelu na tle danych weryfikujących, rząd 2, st. wielo. 2, bez rekurencji

Rys. 2.29. Wyjście modelu na tle danych uczących, rząd 2, st. wielo. 2, z rekurencją



Rys. 2.30. Wyjście modelu na tle danych weryfikujących, rząd 2, st. wielo. 2, z rekurencją

Błąd dla wyjść modelu bez rekurencji jest bardzo mały już dla modeli o małych rzędach i stopniach, wyjścia modelu prawie pokrywają się z danymi rzeczywistymi. Dla modelu z rekurencją jest gorzej, jednak widać, że wcześniejsze obserwacje się potwierdzają i wraz ze zwiększaniem parametrów zwiększa się tez dokładność. Poniżej wykres wyjścia modelu dla rzędu 13 i stopnia wielomianu 5 na tle danych weryfikacyjnych.

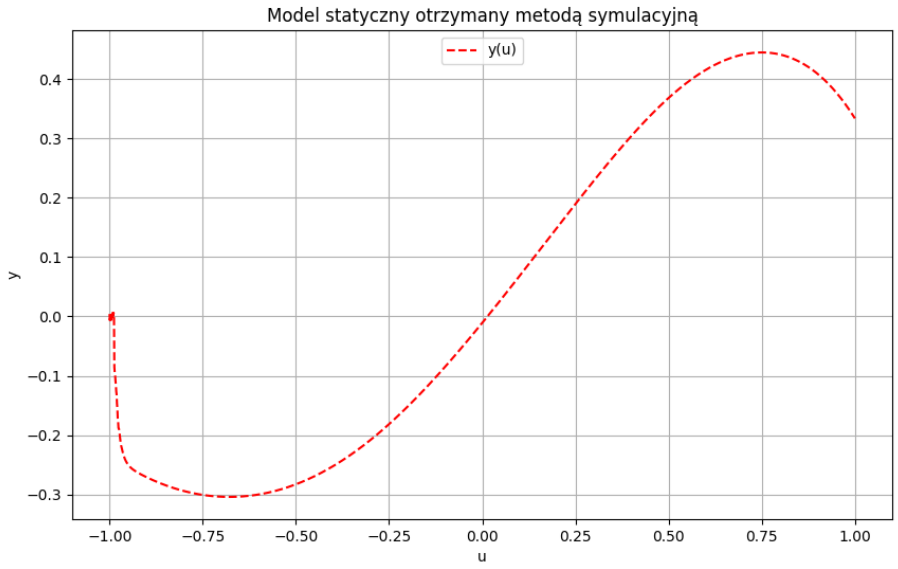


Rys. 2.31. Wyjście modelu na tle danych weryfikujących, rząd 13, st. wielo. 5, z rekurencją

Gołym okiem pratycznie nie da się zauważyć różnicy wyjścia modelu i danych rzeczywistych. Jest to najlepszy model.

2.4. Charakterystyka statyczna

Charakterystyke statyczną wyznaczono dla następującego modelu: rząd dynamiki 13, stopień wielomianu 5.



Rys. 2.32. Charakterystyka statyczna

Otrzymana charakterystyka statyczna odpowiada danym statycznym z zadania 1, jedynie na samym początku charakterystyki wartości wynoszą 0, wynika to najprawdopodobniej z braku danych na tym przedziale.

2.4.1. Implementacja algorytmu

```
y_wer_hat_rek = []
u_wer = np.linspace(-1, 1, 2000)
for i in range(n_dyn):
    y_wer_hat_rek.append(y_ucz[i])
columns = []
for i in range(1, n_dyn+1):
    for j in range(1, n_poly+1):
        columns.append(u_ucz[n_dyn-i:-i]**j)
        columns.append(y_ucz[n_dyn-i:-i]**j)
M_ucz = np.column_stack(tuple(columns))
wsp_ucz = np.linalg.lstsq(M_ucz, y_ucz[n_dyn:], rcond=None)[0]
for i in range(len(y_ucz)-n_dyn):
    yk_wer_rek = 0
    counter = 0
    for j in range(1, n_dyn+1):
        for k in range(1, n_poly+1):
            yk_wer_rek += wsp_ucz[counter]*u_wer[i+n_dyn-j]**k +
                    wsp_ucz[counter+1]*y_wer_hat_rek[i+n_dyn-j]**k
            counter += 2
    y_wer_hat_rek.append(yk_wer_rek)
```

Zamiast y weryfikacyjnych modelowi podano 2000 wartości od -1 do 1, bo taki był przedział wartości sygnału u.

3. Zadanie dodatkowe

Do wykonania zadanie skorzystano z biblioteki Keras. Jako nieliniową funkcję aktywacji wybrano Leaky Relu, gdyż dawała najlepsze rezultaty (dla tgh wartości kończyły się na 1, a dla ReLu nie było ujemnych wartości). Wybrano rządy dynamiki równy 13 na podstawie poprzednich doświadczeń. Przetestowano dla jednego neurona w warstwie ukrytej oraz piętnastu.

3.1. Implementacja algorytmu

```
model = Sequential()
if recursive:
  model.add(LSTM(neurony,input_shape=(rzad-1,2),activation='leaky_relu'))
else:
  model.add(Dense(neurony,input_dim=2*(rzad-1),activation='leaky_relu'))
model.add(Dense(1,activation='leaky_relu'))

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(u, y, epochs=100)

predictions_ucz = model.predict(u)
u_wer, y_wer = create_data(rzad, False, recursive)
predictions_wer = model.predict(u_wer)

predictions_ucz = predictions_ucz.flatten()
predictions_wer = predictions_wer.flatten()

mse_ucz = mean_squared_error(y, predictions_ucz)
mse_wer = mean_squared_error(y_wer, predictions_wer)
```

3.2. Błędy modeli

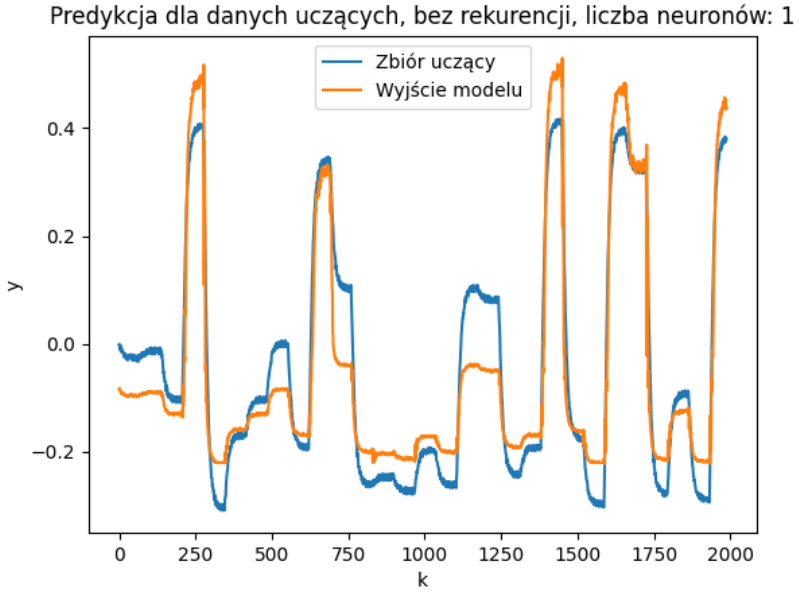
Obliczano błąd średnio kwadratowy, wyniki w tabeli poniżej.

Model	Błąd uczenia	Błąd weryfikacji
1 neuron bez rekurencji	0.0049288222768150295	0.007430235142535047
1 neuron z rekurencją	0.00040339790835600505	0.0007669741127963158
15 neuronów bez rekurencji	5.0423620851449225e-05	6.496136137764511e-05
15 neuronów z rekurencją	6.010847332119231e-05	0.00019428729665904045

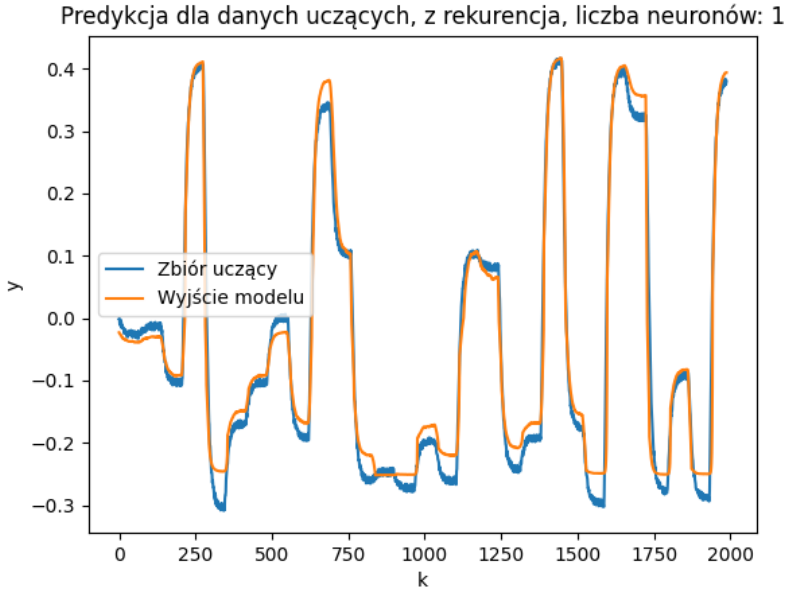
Tab. 3.1. Porównanie błędów uczenia i weryfikacji dla modeli z 1 i 15 neuronami

Jak widać po zwiększeniu liczby neuronów w warstwie ukrytej nastąpiła znaczna poprawa błędu. Ponadto model z rekurencją okazuje się być lepszy w tym przypadku. Jest to dziwne, ponieważ w poprzednich doświadczeniach było inaczej. Możliwe, że jest to spowodowane tym, że wagi w sieci neuronowej na początku są losowane.

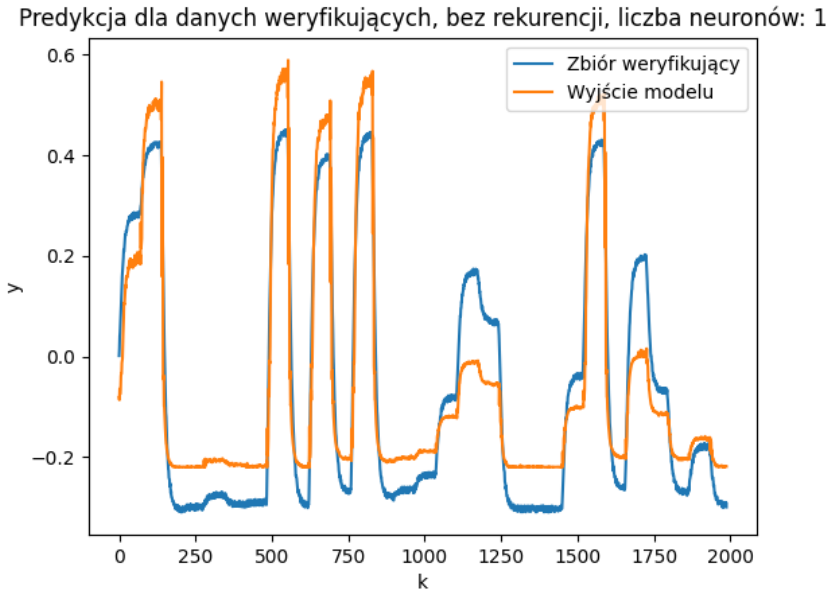
3.3. Wykresy



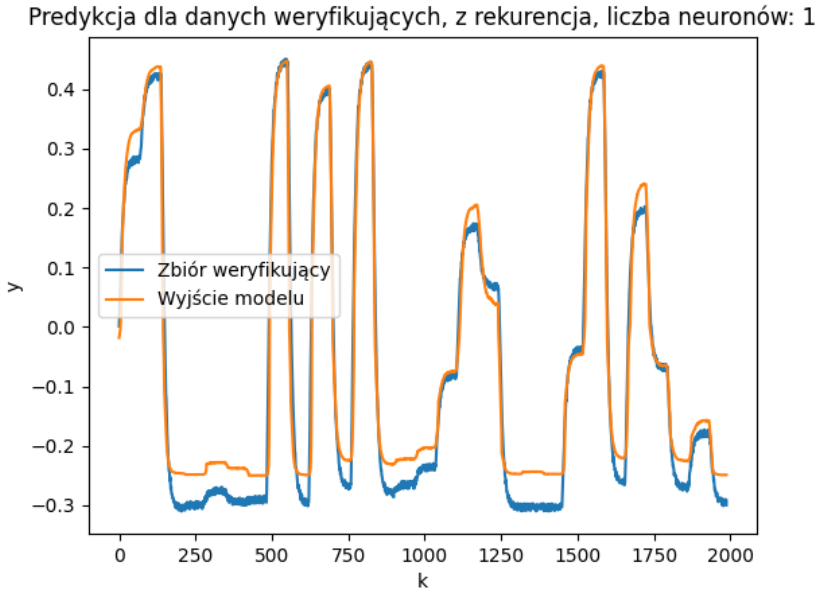
Rys. 3.1. Przewidywania modelu, 1 neuron, bez rekurencji, na tle uczących



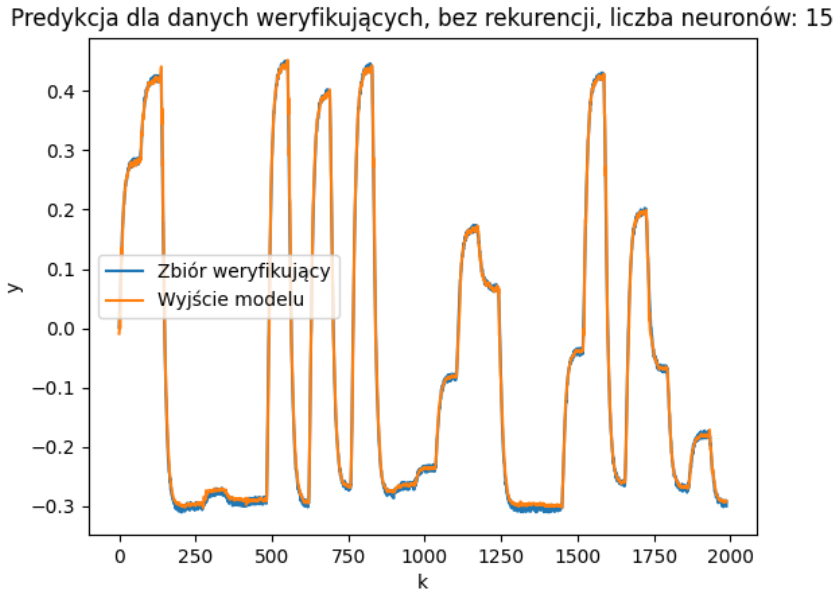
Rys. 3.2. Przewidywania modelu, 1 neuron, z rekurencją, na tle uczących



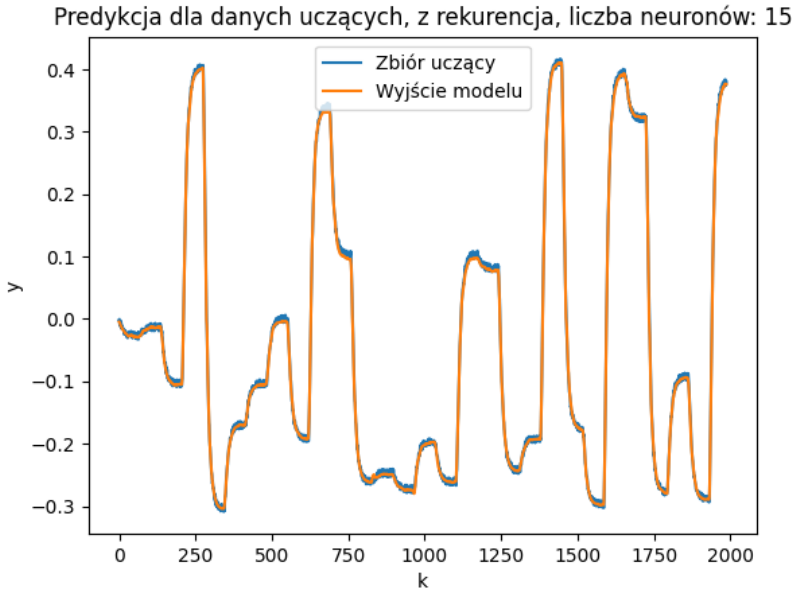
Rys. 3.3. Przewidywania modelu, 1 neuron, bez rekurencji na tle weryfikujących



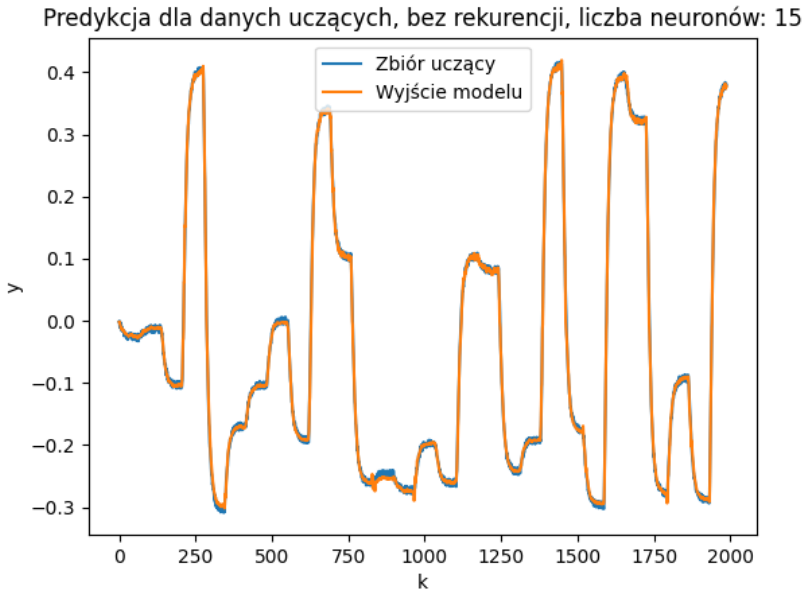
Rys. 3.4. Przewidywania modelu, 1 neuron, z rekurencją, na tle weryfikujących



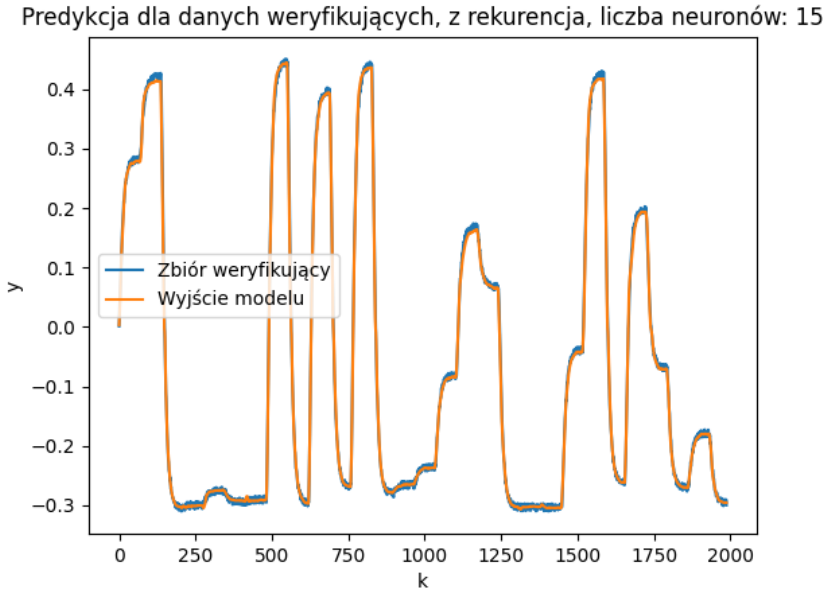
Rys. 3.5. Przewidywania modelu, 15 neuronów, bez rekurencji, na tle uczących



Rys. 3.6. Przewidywania modelu, 15 neuronów, z rekurencją, na tle uczących



Rys. 3.7. Przewidywania modelu, 15 neuronów, bez rekurencji na tle weryfikujących



Rys. 3.8. Przewidywania modelu, 15 neuronów, z rekurencją, na tle weryfikujących

Widać, że po zmianie liczby neuronów nastąpiła bardzo duża poprawa jakości identyfikacji. Dla jednego neurona wyjście modelu prawie nie pokrywało się z faktycznymi danymi, a już dla piętnastu idealnie dopasowywuje się do prawdziwych danych. Po rysunkach ciężko stwierdzić czy model z rekurencją czy bez jest lepszy.