**Wydział Elektroniki i Technik Informacyjnych**
**Politechnika Warszawska**

# MODI

## Sprawozdanie z projektu nr 2

**Jan Szymczak**

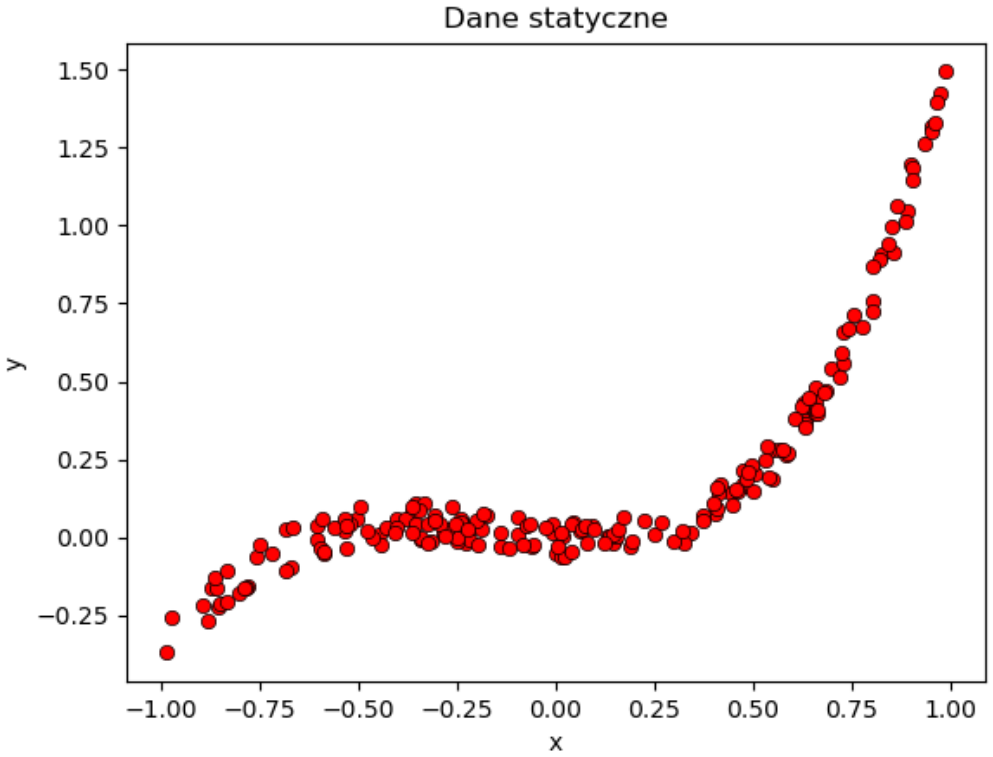
**Warszawa, 2024**

# Spis treści

# 1. Identyfikacja modeli statycznych

## 1.1. Wizualizacja oraz podział danych

Tak przedstawiają się omawiane dane statyczne przed podziałem:

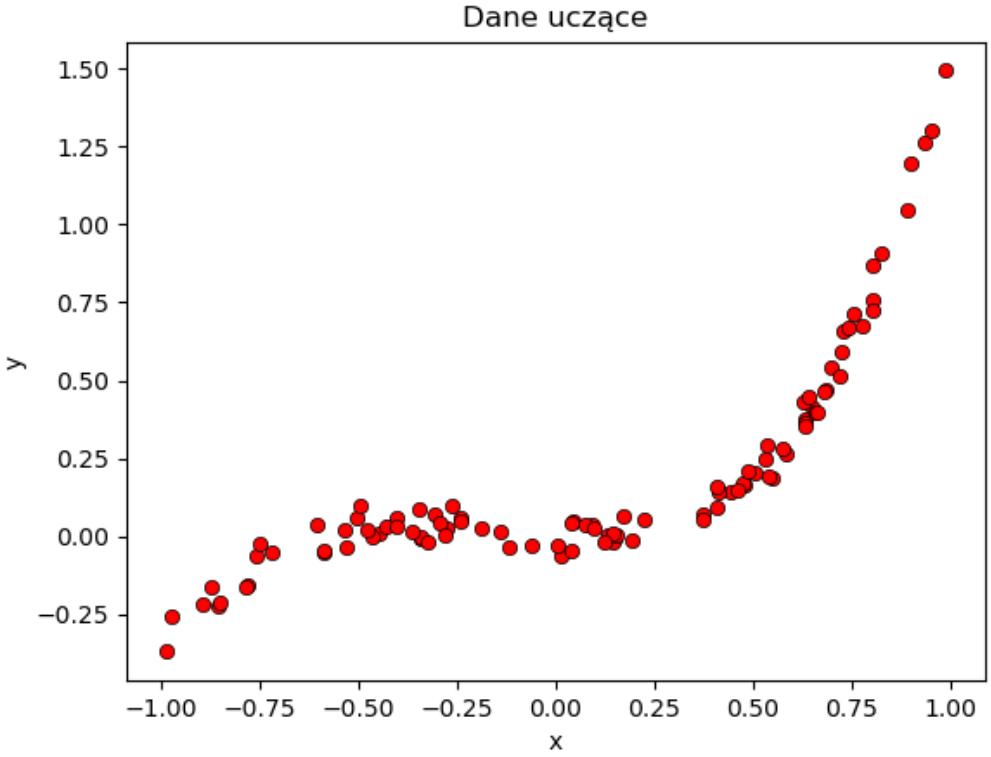



Rys. 1.1. Dane statyczne

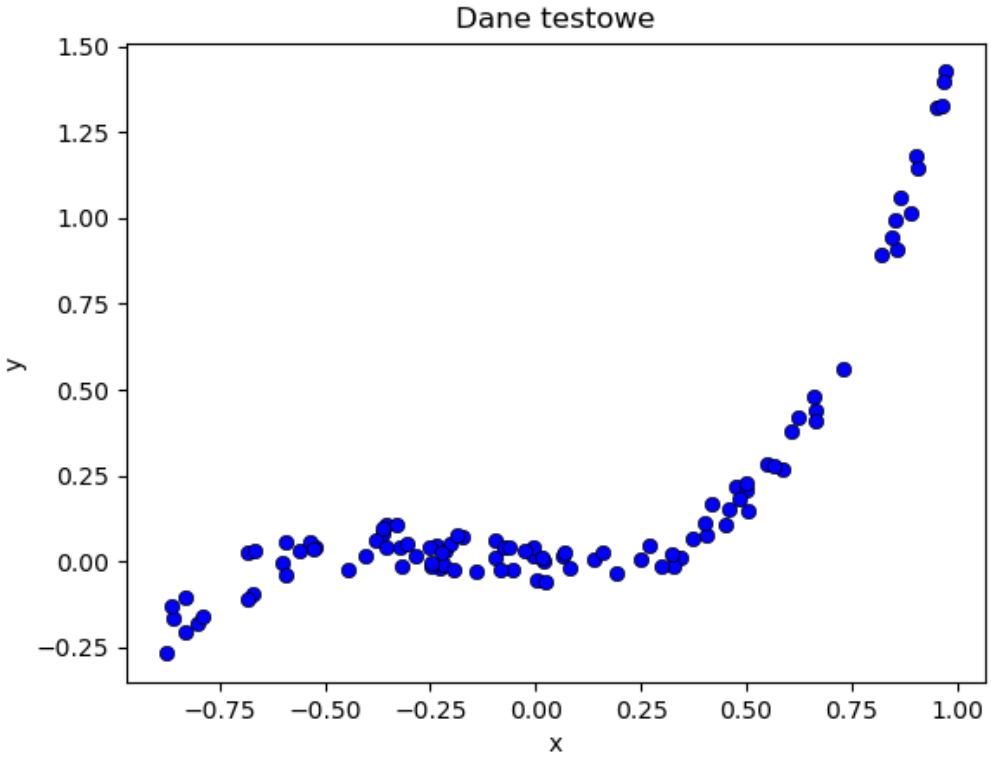
Aby podzielić dane na zbiór uczący oraz testujący, do zbioru testującego została dodana co druga próbka. Reszta próbek przydzielona została do zbioru trenującego. Zostało to przeprowadzone za pomocą funkcji *split_static_data()* a kluczowe są tu linijki:

```
    train = data[::2]
    test = data[1::2]
```

Tak przedstawiają się dane po podziale:




Rys. 1.2. Zbiór uczący




Rys. 1.3. Zbiór testujący

## 1.2. Statyczny model liniowy

Statyczny model liniowy i każdy kolejny model został wyznaczony metodą najmniejszych kwadratów, czyli zgodnie ze wzorem:

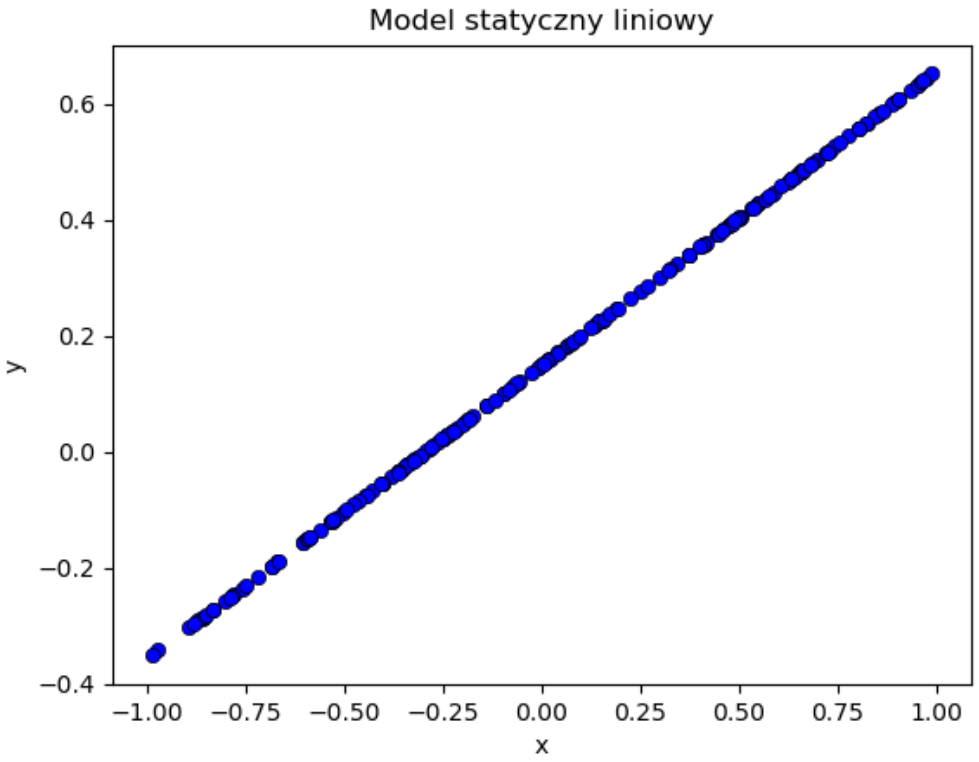
$$w = (M^T M)^{-1} M^T \cdot Y$$

lub: $M \backslash Y$, i to ten sposób został wykorzystany w kodzie programu: W programie zostało to wykonane w następujący sposób:

```
M = [np.ones(len(train)), train[:, 0]]
M = np.array(M).T
w = np.linalg.lstsq(M, Y, rcond=None)[0]
```

Pierwsze dwie linijki to stworzenie macierzy M, gdzie pierwszą kolumnę stanowią jedynki, a drugą stanowią argumenty (x) danych uczących (pierwsza kolumna tych danych). Ostatnia linijka to wyliczenie współczynników przy pomocy "lewego dzielenia", które zostało przeprowadzone z pomocą biblioteki *numpy* (w projekcie wykorzystana została jedynie ta biblioteka do obliczeń, a do wizualizacji danych biblioteka pyplot z matplotlib) i komendy *lstsq*. Aby obliczyć wyjście modelu wykorzystana została komenda *np.polyval(w[::-1], data[:, 0])*.

Tak prezentuje się liniowa charakterystyka statyczna y(u):

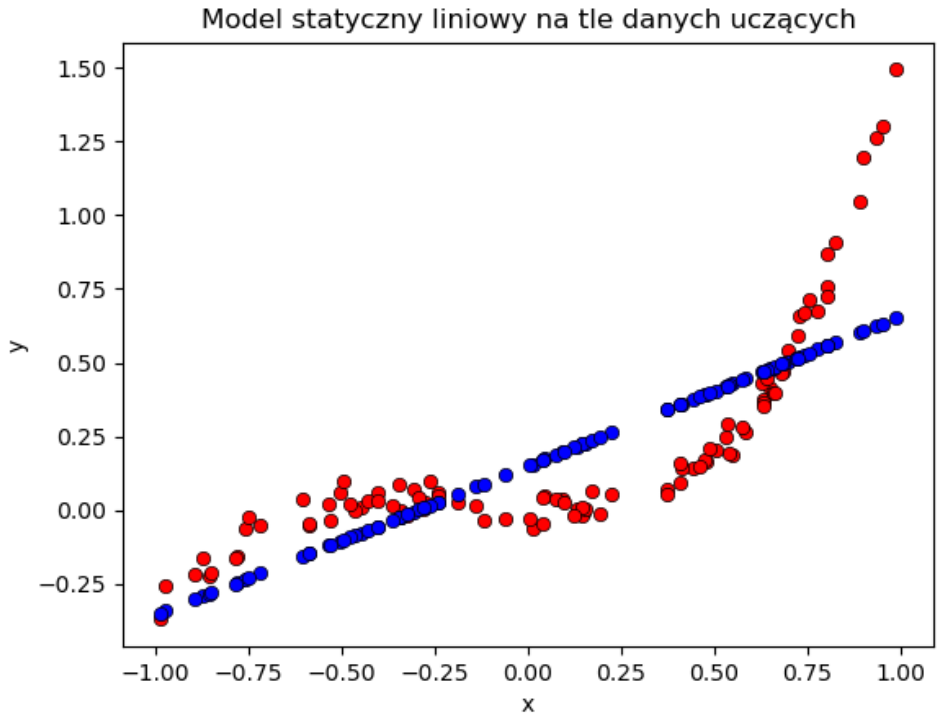



Rys. 1.4. Liniowa charakterystyka statyczna

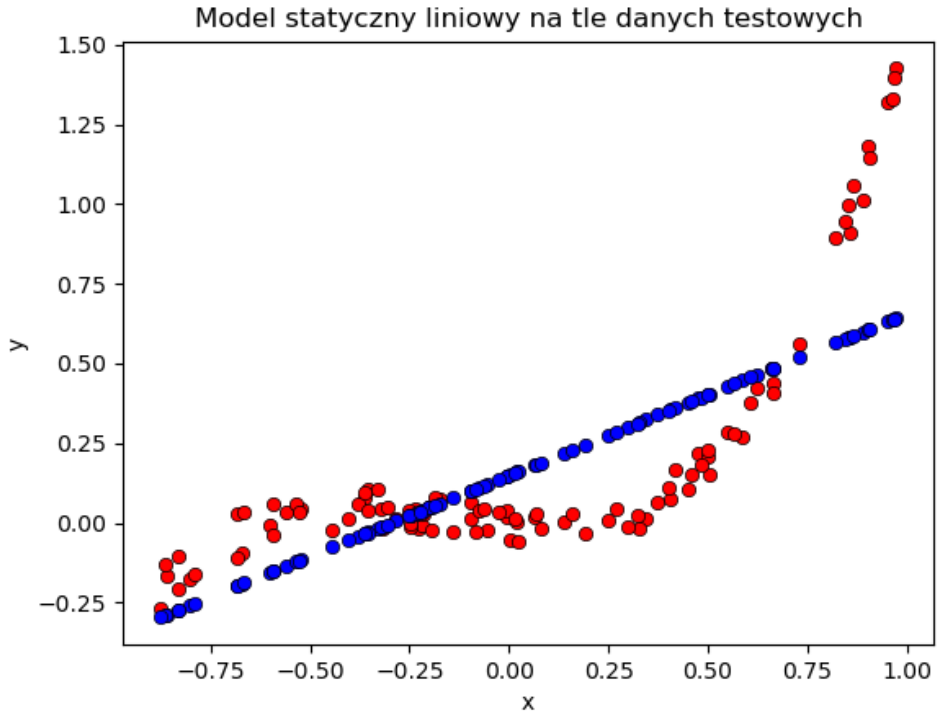
Uzyskane błędy:

— Dla danych uczących: 4.485
— Dla danych testowych: 5.874

Charakterystyka statyczna na tle kolejno danych uczących i testowych:




Rys. 1.5. Liniowa charakterystyka statyczna na tle danych uczących




Rys. 1.6. Liniowa charakterystyka statyczna na tle danych testowych

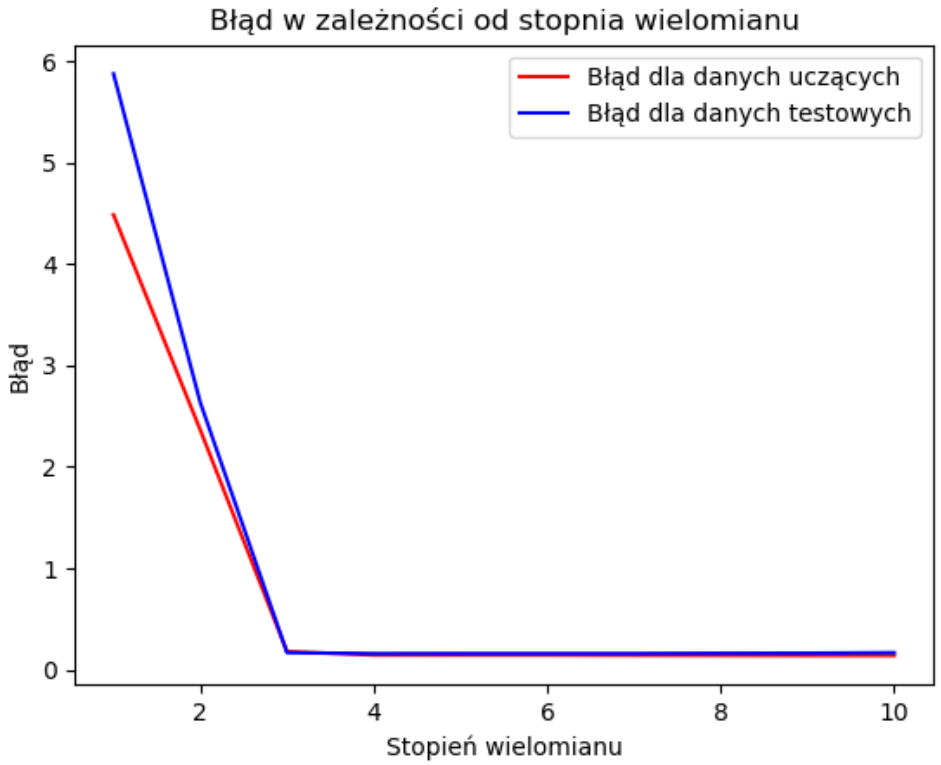
Jak widać po wykresach oraz wielkości błędów, przybliżenie wielomianem pierwszego stopnia jest niewystarczające, ponieważ charakter danych jest nieliniowy.

## 1.3. Statyczny model nieliniowy

Napisany program przyjmuje jako argument stopień wielomianu, który może być dowolną dodatnią liczbą całkowitą. Jedyną różnicą względem programu obliczającego charakterystykę liniową jest sposób tworzenia macierzy M, który teraz został uogólniony (oczywiście umożliwia także stworzenie charakterystyki liniowej). Tworzenie macierzy M przedstawia się następująco:

```
M = []
for i in range(len(train[:, 0])):
    row = []
    row.append(1)
    for j in range(1, N + 1):
        row.append(train[i, 0] ** j)
    M.append(row)
M = np.array(M)
```

Ponownie, pierwszą kolumnę macierzy M stanowią jedynki, jednak kolejne kolumny to pierwsza kolumna danych trenujących, z wartościami podniesionymi do kolejnych potęg naturalnych, począwszy od jedynki, aż do argumentu N, czyli stopnia wielomianu. Współczynniki w są wyliczane w ten sam sposób.
W celu znalezienia optymalnego stopnia wielomianu, przeprowadzono modelowanie dla kolejnych całkowitych stopni wielomianu z przedziału $\langle 1, 10\rangle$. Tak przedstawiają się wykresy błędów dla danych uczących i testowych względem stopni wielomianu:



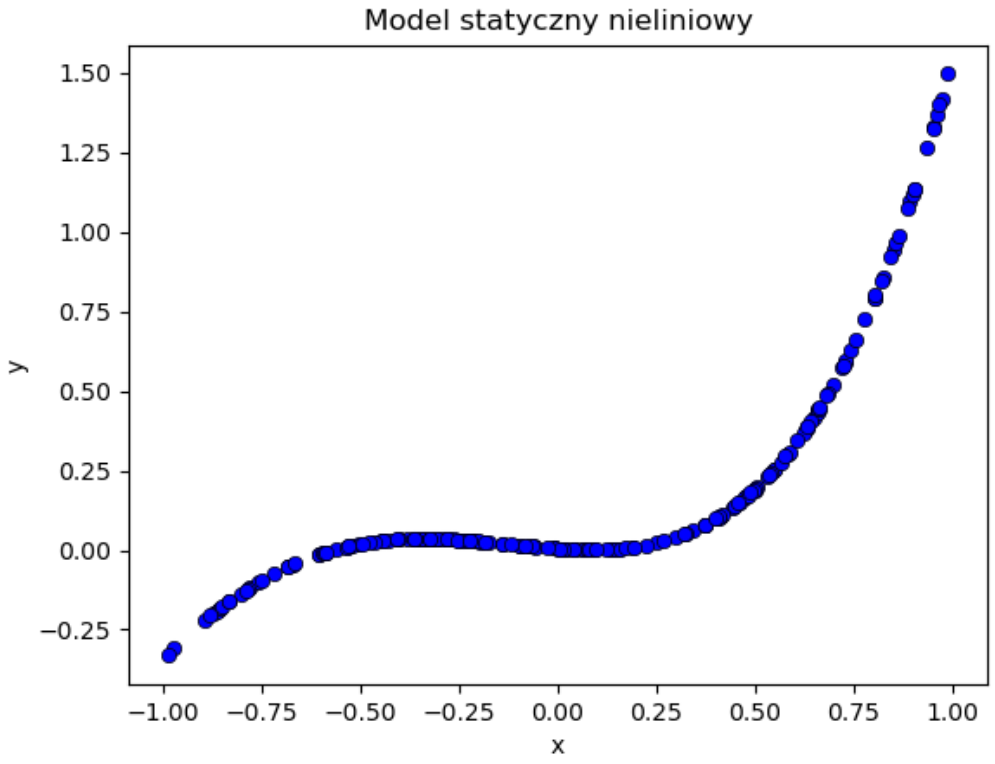

Rys. 1.7. Obliczone błędy względem stopnia wielomianu charakterystyki statycznej

Z wykresu dokładne błędy są niemożliwe do odczytania, więc wyniki zostały przedstawione w tabeli:

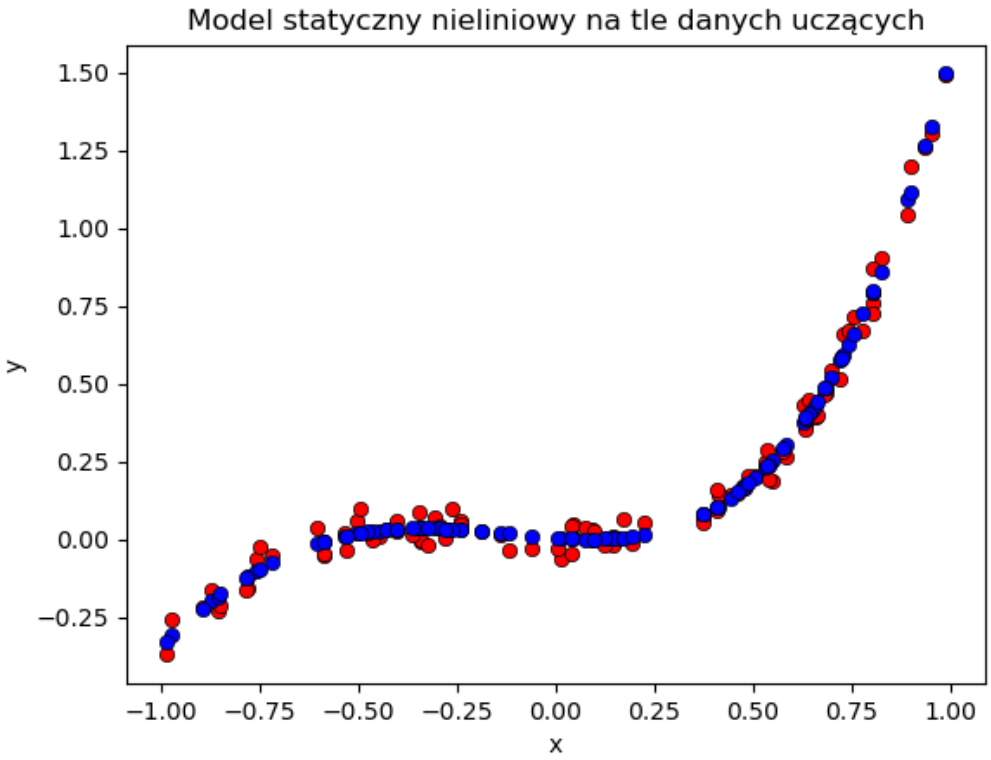
| Stopień wielomianu | 1 | 2 | 3 | **4** | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Dane uczące | 4.485 | 2.365 | 0.180 | **0.146** | 0.146 | 0.146 | 0.144 | 0.143 | 0.142 | 0.142 |
| Dane testowe | 5.874 | 2.636 | 0.171 | **0.161** | 0.162 | 0.162 | 0.162 | 0.164 | 0.165 | 0.170 |

Tab. 1.1. Błędy dla danych uczących i testowych dla wielomianów stopnia od 1 do 10

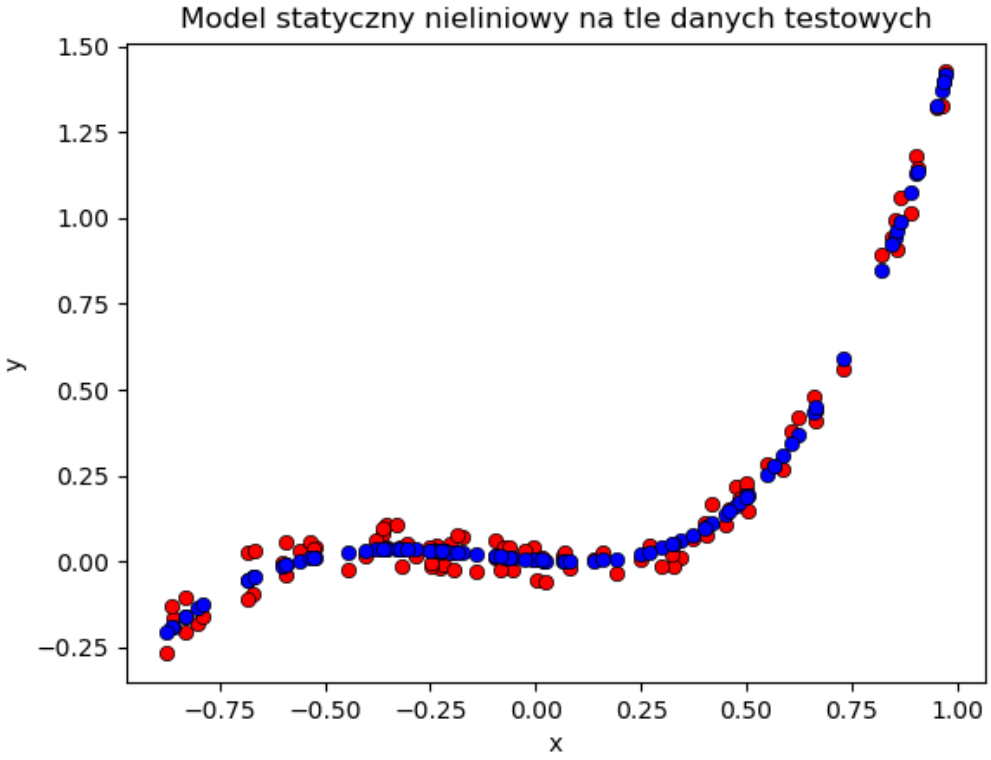
Najmniejszy błąd dla danych testowych został uzyskany dla wielomianu stopnia 4, dlatego ta kolumna została wyróżniona w tabeli. Widać, że wraz ze wzrostem stopnia wielomianu błąd dla danych uczących maleje. Z początku, błąd dla danych testowych także maleje, jednak począwszy od wielomianu stopnia 4 zaczyna rosnąć, jest to spowodowane przez tzw. "overfitting" - charakterystyka zbyt bardzo dopasowuje się do danych trenujących, przez co staje się coraz mniej uniwersalna dla danych spoza zbioru uczącego i dla nich generuje większy błąd. Najlepszym modelem statycznym jest zatem model stopnia czwartego. Poniżej przedstawiona została właśnie charakterystyka statyczna tego modelu oraz jego wyjście na obu zbiorach danych:




Rys. 1.8. Charakterystyka statyczna dla N=4

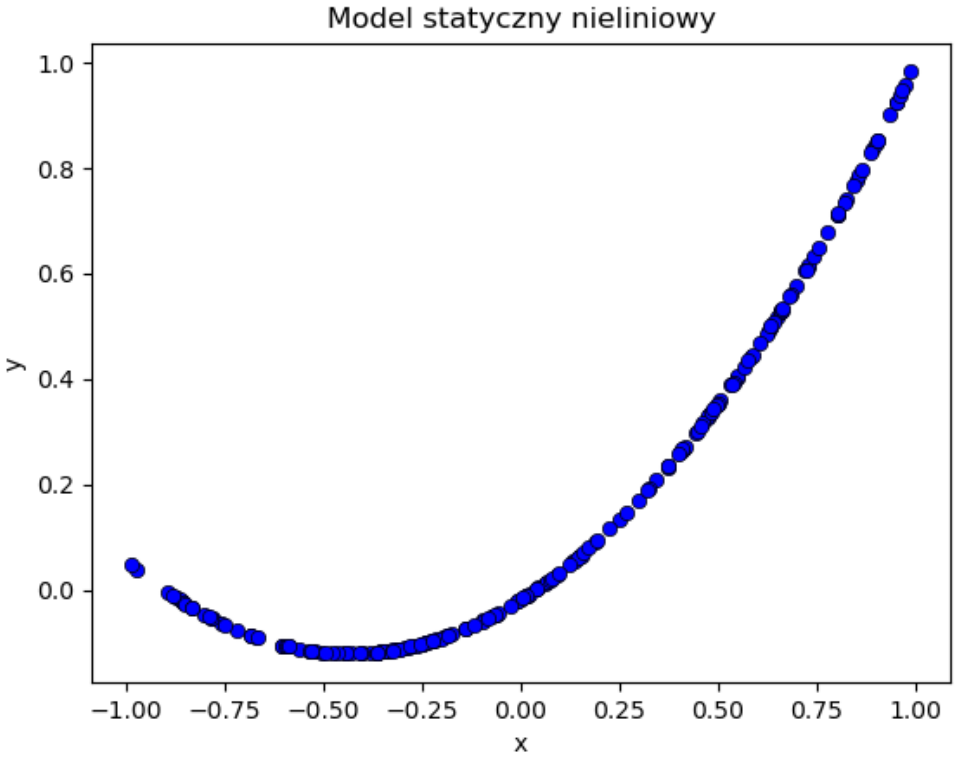



Rys. 1.9. Charakterystyka statyczna dla N=4 na tle danych uczących

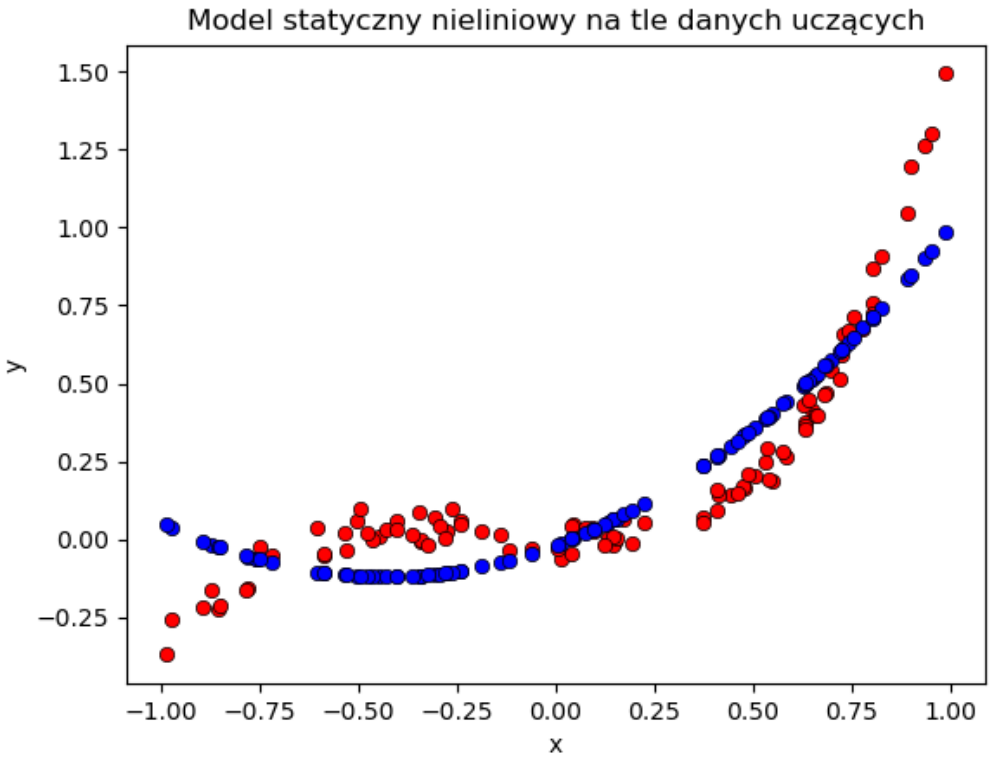



Rys. 1.10. Charakterystyka statyczna dla N=4 tle danych testowych

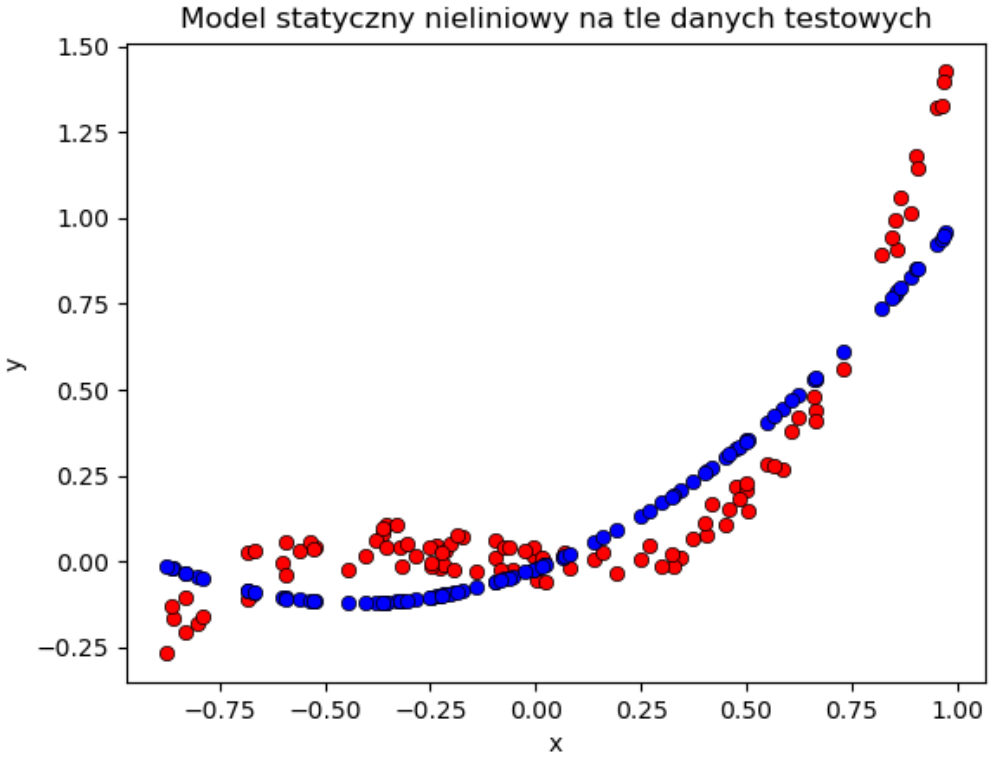
Z uwagi, że testy zostały wykonane do wielomianu stopnia 10, w celu uniknięcia powielania i tak podobnych wykresów, poniżej zostaną przedstawione odpowiadające wykresy jedynie dla wielomianów stopnia 2, 5 i 10, nie dla wszystkich, jak zostało to podane w poleceniu. Ukaże to w mojej ocenie wystarczająco zmiany, jakie zachodzą pomiędzy różnymi stopniami wielomianów. Napisany program oczywiście umożliwia wizualizacje charakterystyki dowolnego stopnia. Tak naprawdę, analizując charakter danych, widać, że posiada on dwa "wygięcia", więc teoretycznie wielomian stopnia 3 już powinien zadowalająco przybliżać dane - i tak się rzeczywiście dzieje, patrząc na wartości błędów w tabeli 1.1. Od stopnia 4, różnice w wykresach charakterystyk są na dobrą sprawę niezauważalne, podobnie ich błędy. Dopiero dla naprawdę wysokich stopni, np. wielomianu stopnia 100, widać nienaturalne i niepożądane (zauważalny "overfitting") dopasowywanie się charakterystyki do danych trenujących, co skutkuje dużym wzrostem błędu dla danych testujących. Poniżej wykresy dla wspomnianych różnych stopni wielomianów:



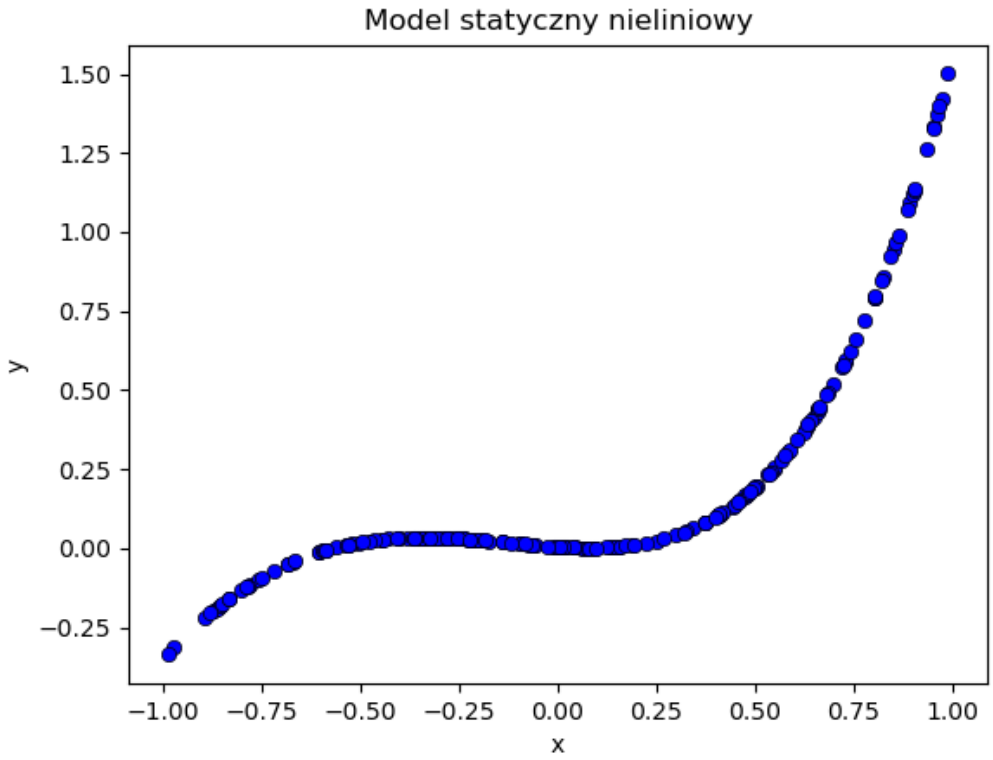

Rys. 1.11. Charakterystyka statyczna dla N=2



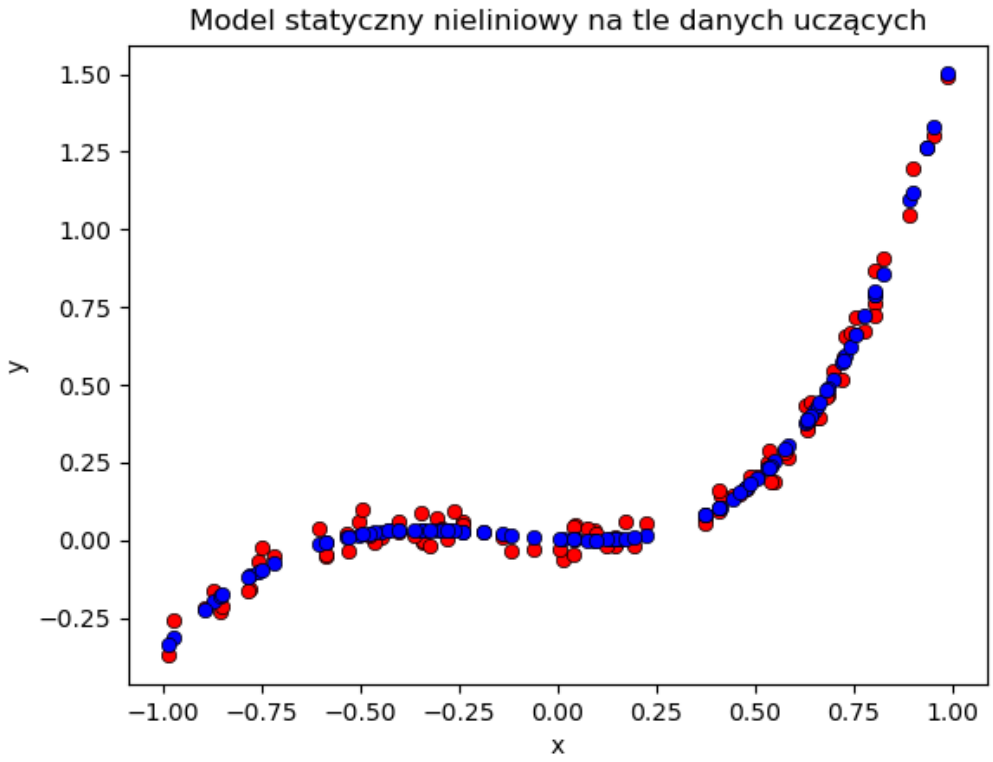

Rys. 1.12. Charakterystyka statyczna dla N=2 na tle danych uczących



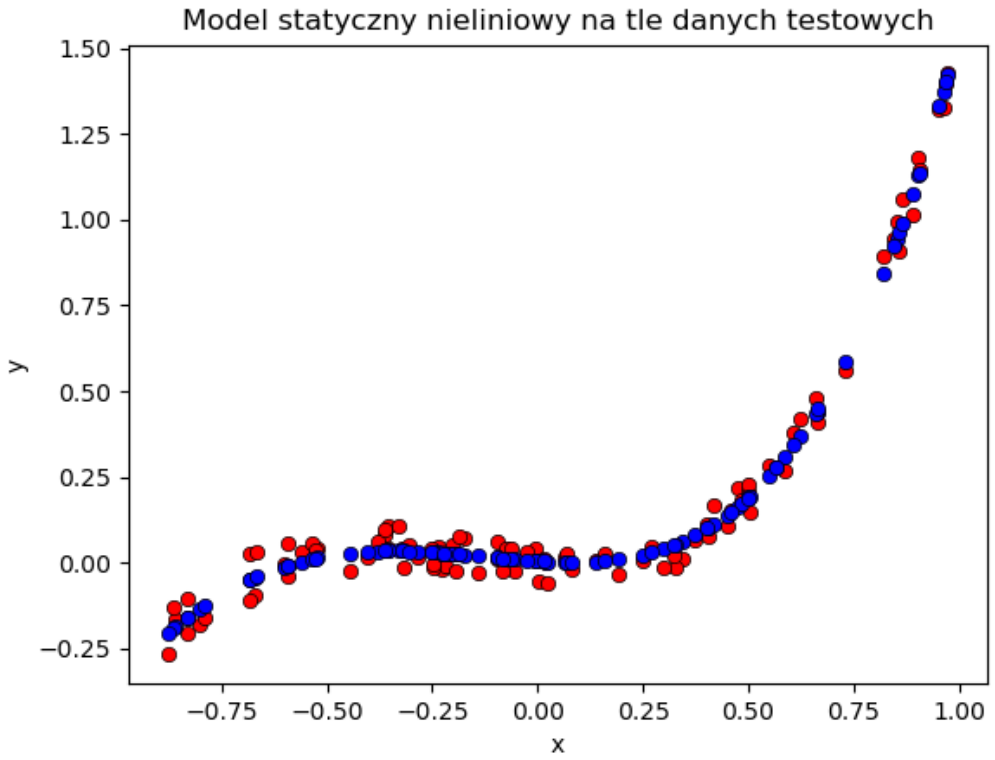

Rys. 1.13. Charakterystyka statyczna dla N=2 tle danych testowych



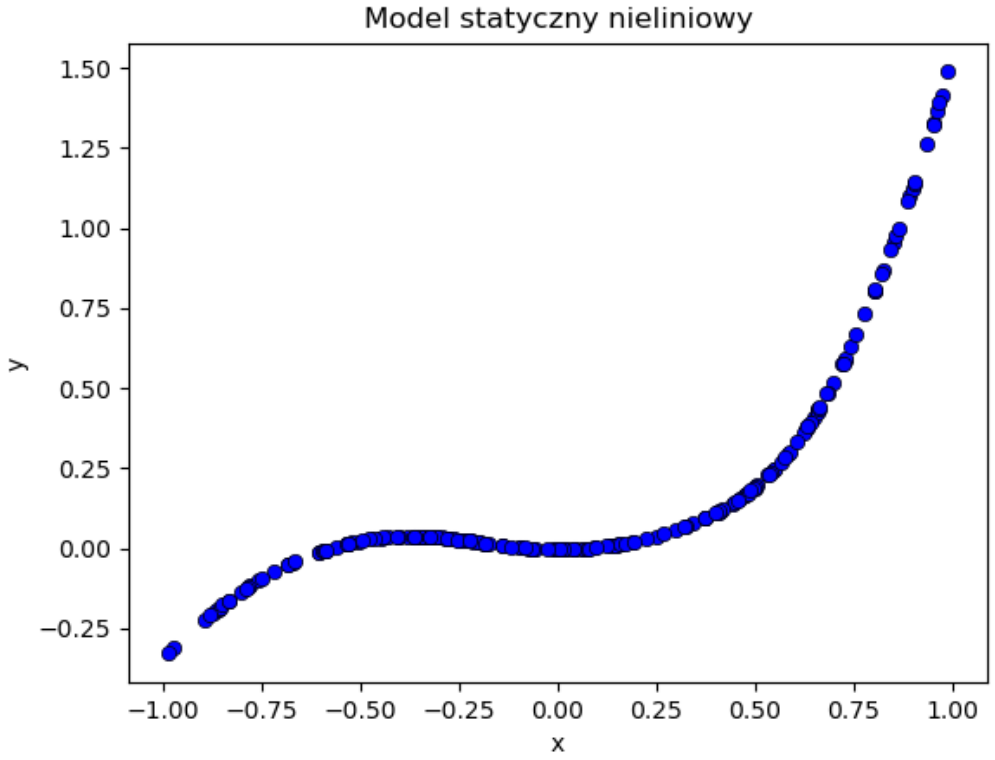

Rys. 1.14. Charakterystyka statyczna dla N=5



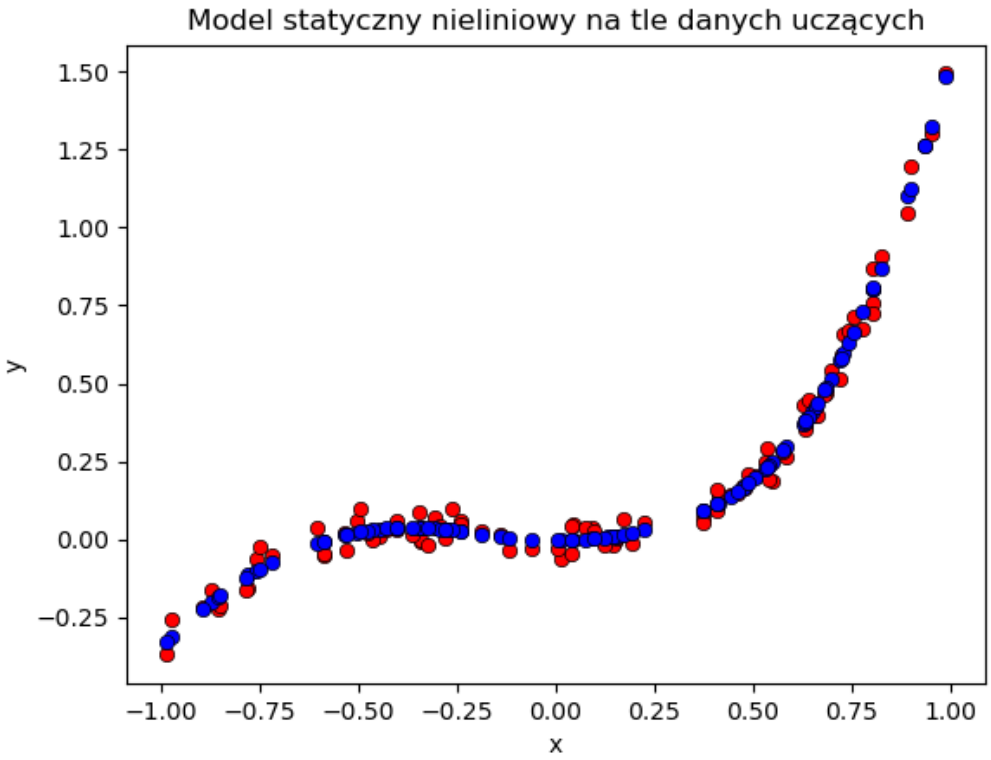

Rys. 1.15. Charakterystyka statyczna dla N=5 na tle danych uczących



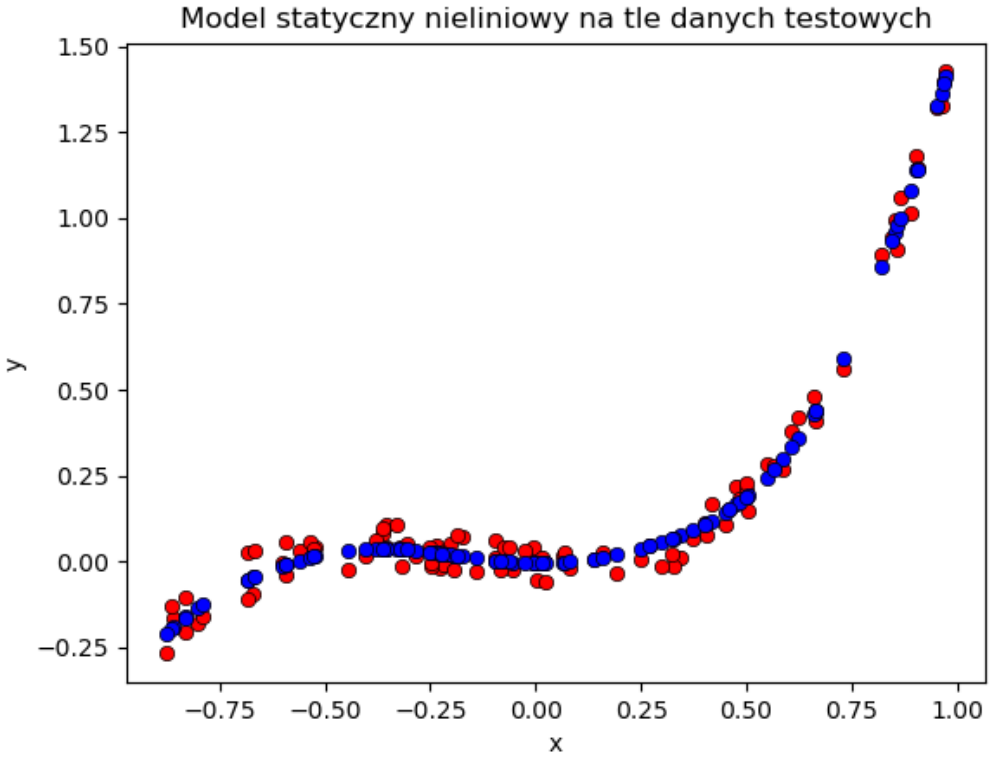

Rys. 1.16. Charakterystyka statyczna dla N=5 tle danych testowych




Rys. 1.17. Charakterystyka statyczna dla N=10




Rys. 1.18. Charakterystyka statyczna dla N=10 na tle danych uczących

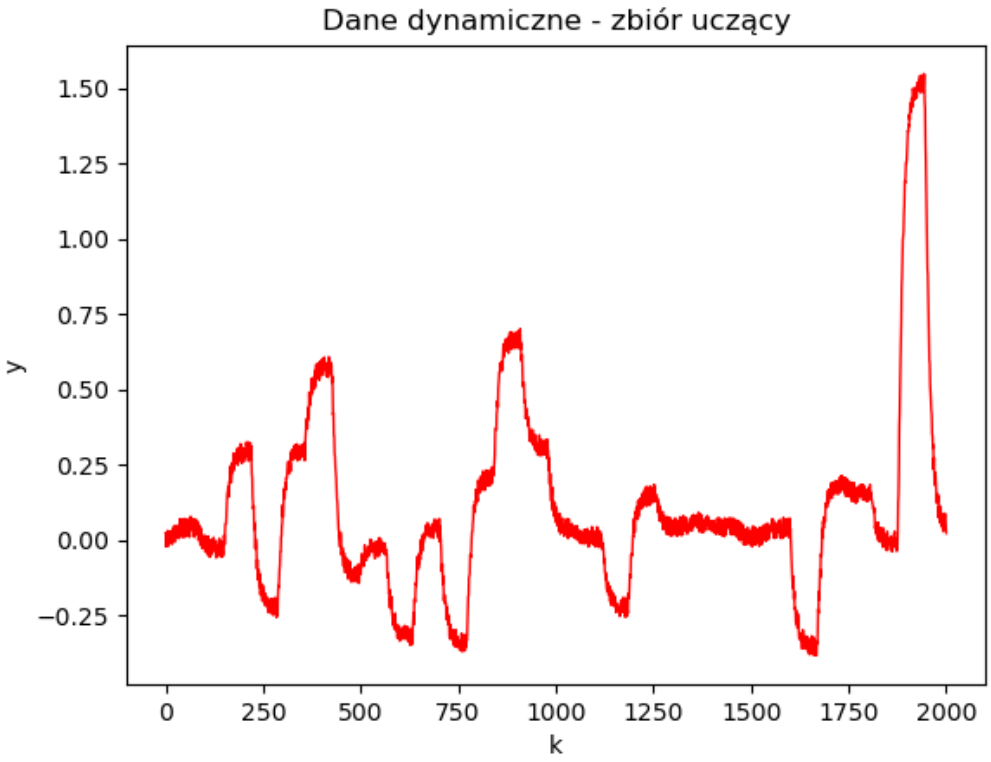



Rys. 1.19. Charakterystyka statyczna dla N=10 tle danych testowych

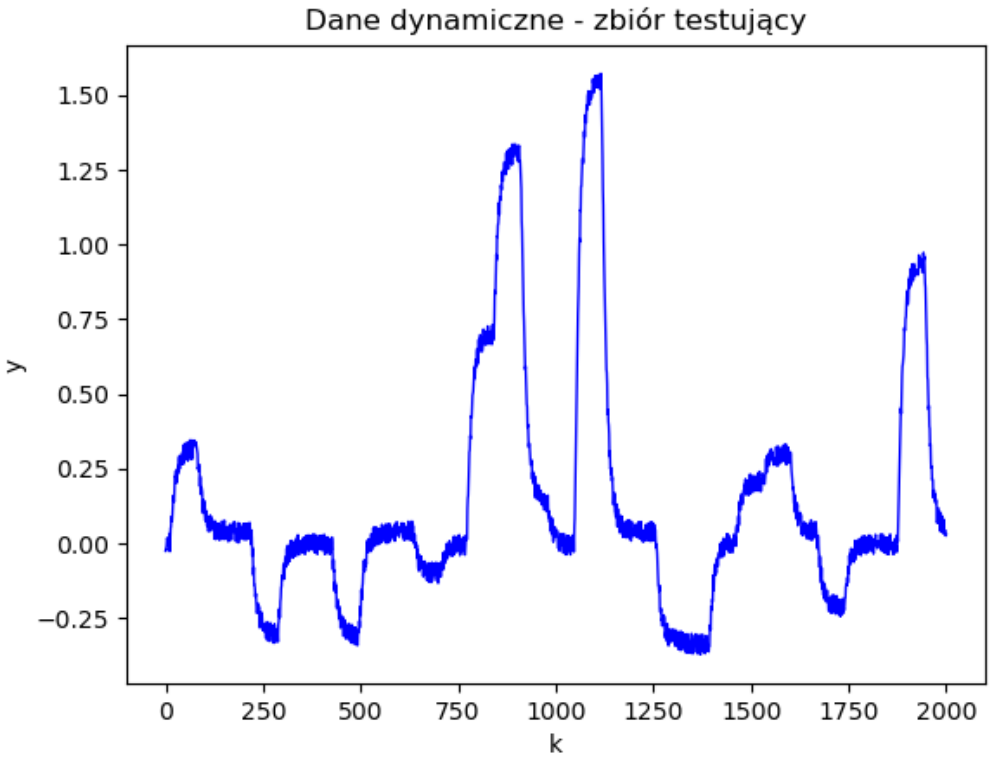
# 2. Identyfikacja modeli dynamicznych

## 2.1. Wizualizacja danych

Tak prezentuje się kolejno zbiór uczący i weryfikujący:




Rys. 2.1. Zbiór uczący




Rys. 2.2. Zbiór testujący

## 2.2. Modele dynamiczne liniowe

Ponownie, do wyznaczenia modeli została wykorzystana metoda najmniejszych kwadratów. Kod programu odpowiadający za to przedstawia się tak:

```
Y = Y[N:]
M = []
for i in range(N-1, len(train_data[:, 0])-1):
    row = []
    for j in range(N):
        row.append(train_data[i - j, 0])
        row.append(train_data[i - j, 1])
    M.append(row)
M = np.array(M)
w = np.linalg.lstsq(M, Y, rcond=None)[0]
```

W tym przypadku N oznacza rząd dynamiki modelu. Napierw zostaje stworzona macierz M w następujący sposób:

$$M = \begin{bmatrix} x_{N-1} & y_{N-1} & x_{N-2} & y_{N-2} & \cdots & x_0 & y_0 \\ x_N & y_N & x_{N-1} & y_{N-1} & \cdots & x_1 & y_1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_n & y_n & x_{n-1} & y_{n-1} & \cdots & x_{n-N+1} & y_{n-N+1} \end{bmatrix}$$

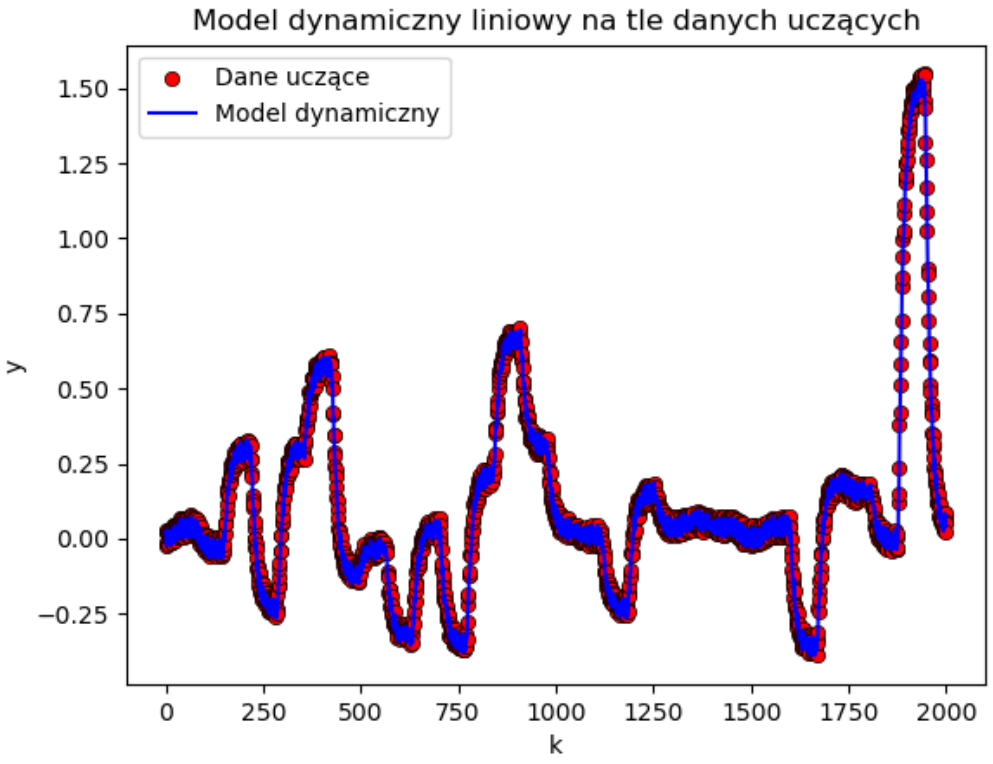
gdzie N to rząd dynamiki a n oznacza ilość wszystkich próbek. Wektor współczynników w jest liczony w taki sam sposób. Wcześniej, oczywiście należy ograniczyć także liczbę wierszów w ko-

lumnie wyjścia danych, uzyskiwane jest to w pierwszej ukazanej linijce kodu. Obliczenie wyjścia modelu dla danych uczących wraz z rekurencją zachodzi następująco:

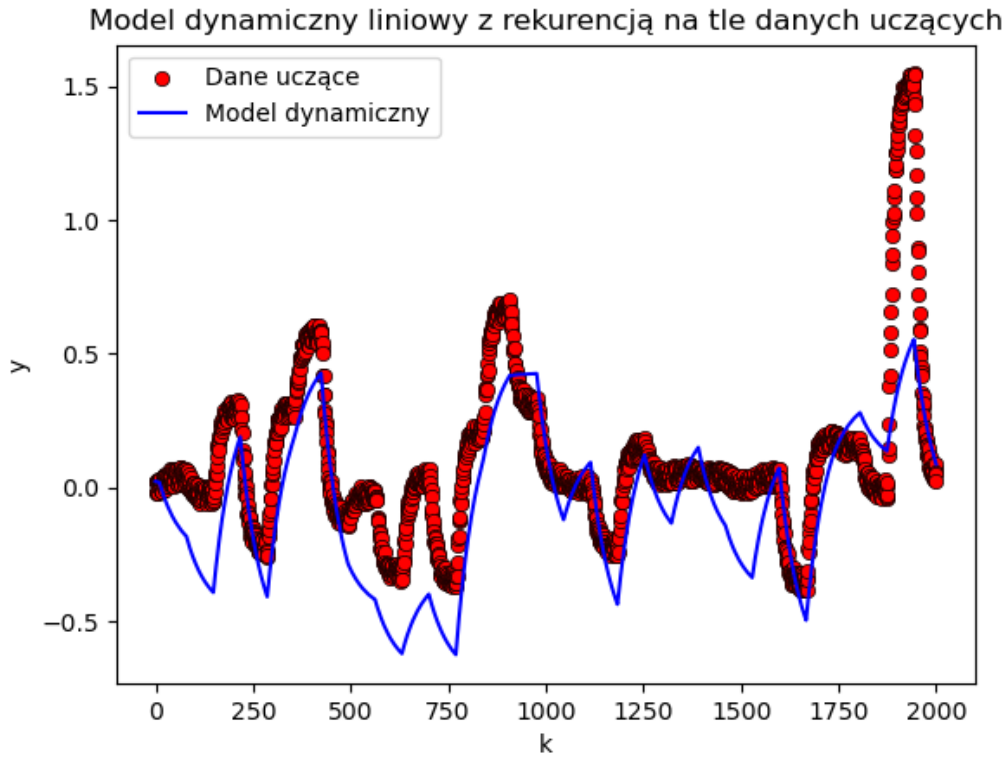
```
ymod_train = []
# Inicjalizacja modelu
ymod_train[:N] = train_data[:N, 1]
# wyjście modelu dla danych uczących
for i in range(N-1, len(train_data[:, 0])-1):
    temp = 0
    for j in range(N):
        temp +=
            w[j * 2] * train_data[i - j, 0] +
            w[j * 2 + 1] * ymod_train[i - j]
    ymod_train.append(temp)
```

Dla danych testowych zmienia się jedynie nazwa zbioru z *train_data* na *test_data*. W przypadku modelu bez rekurencji nie zachodzi oczywiście inicjalizacja modelu oraz przy obliczaniu wyjścia nie zostało wykorzystane wyjście modelu *ymod*, tylko kolumna y odpowiednich danych.

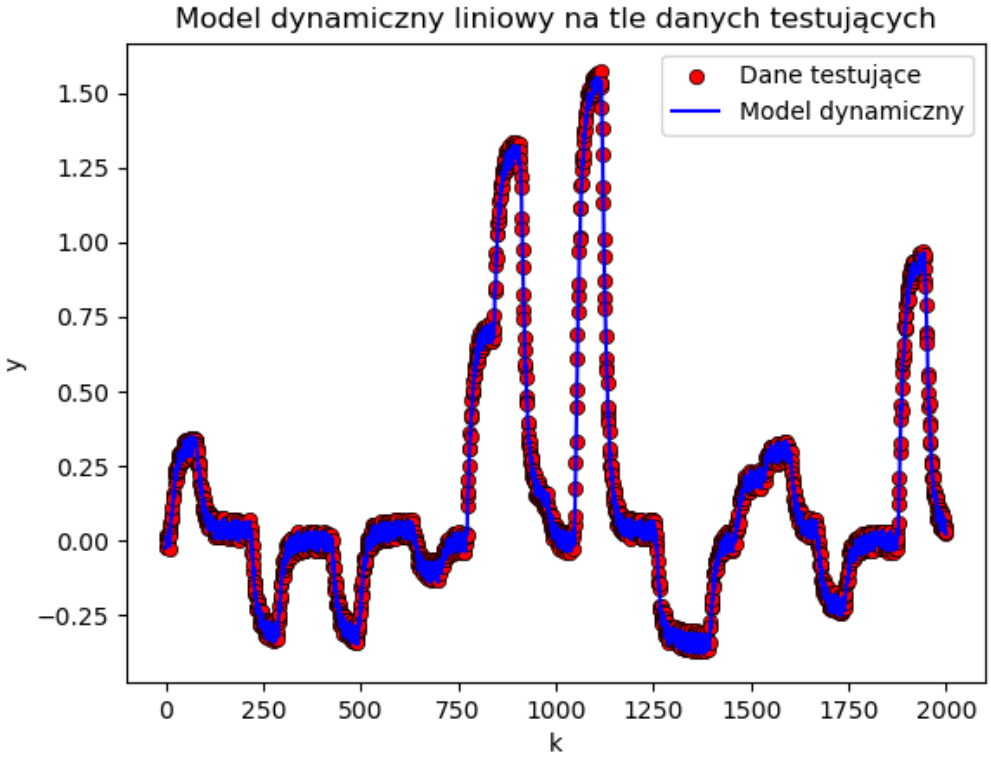
Poniżej znajdują się wyjścia modelu dla dynamiki kolejnych rzędów dynamiki na tle obu zbiorów danych bez rekursji oraz z rekurencją:



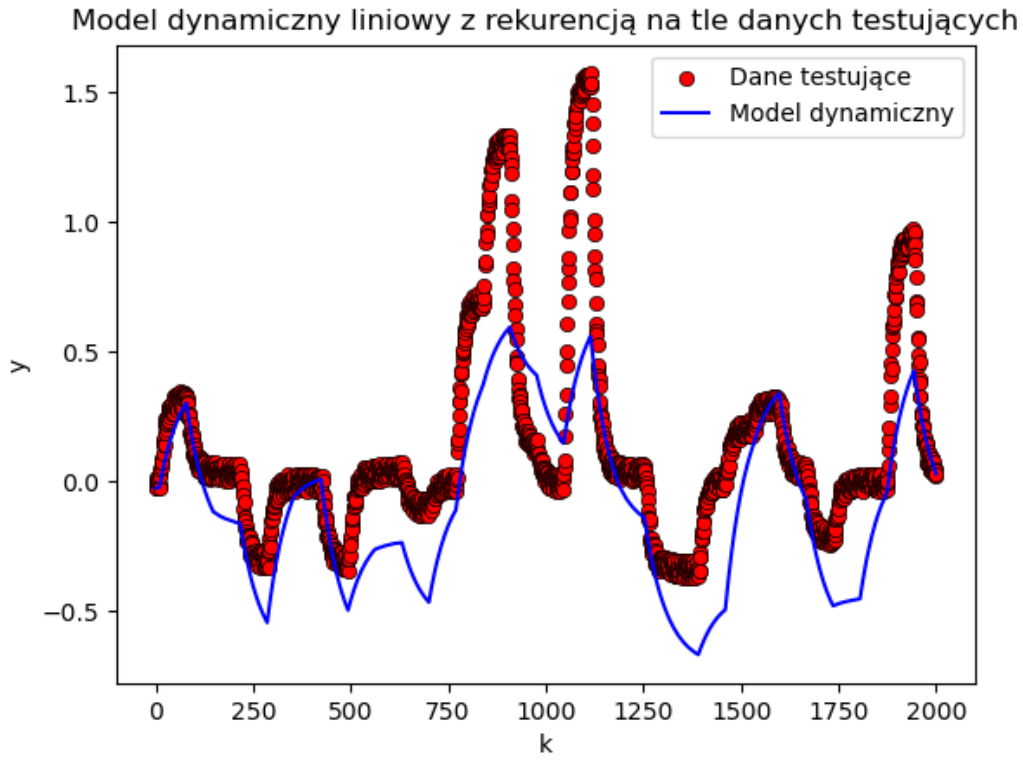

Rys. 2.3. Wyjście modelu bez rekurencji dla N=1 na tle danych uczących



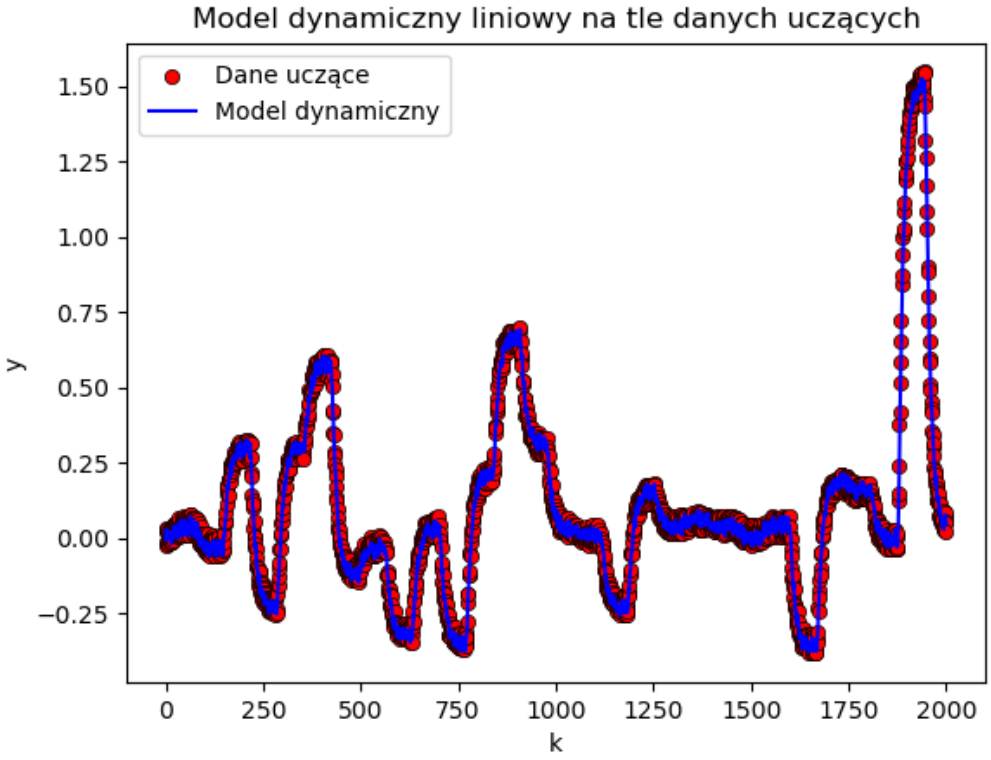

Rys. 2.4. Wyjście modelu z rekurencją dla N=1 na tle danych uczących



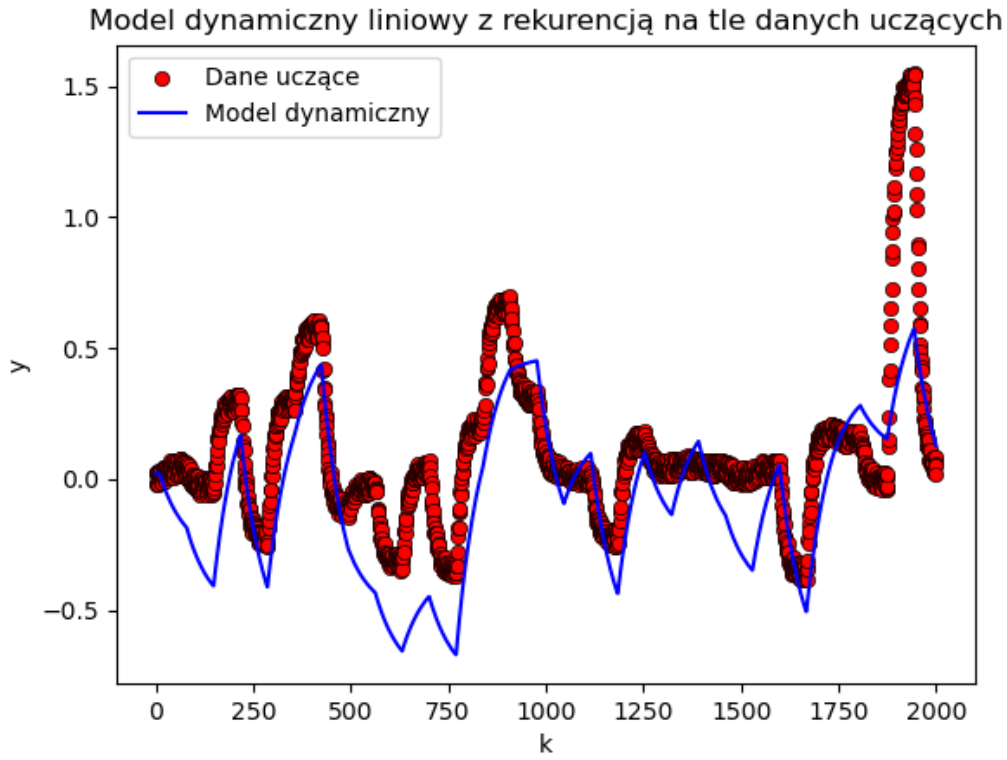

Rys. 2.5. Wyjście modelu bez rekurencji dla N=1 na tle danych testowych



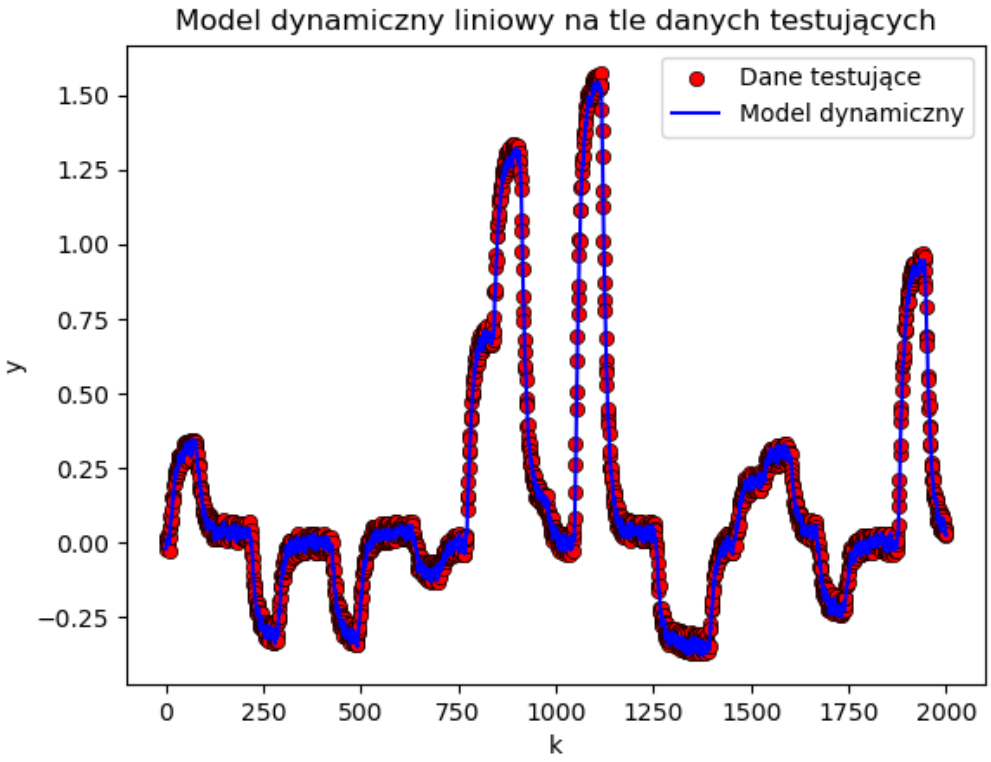

Rys. 2.6. Wyjście modelu z rekurencją dla N=1 na tle danych testowych



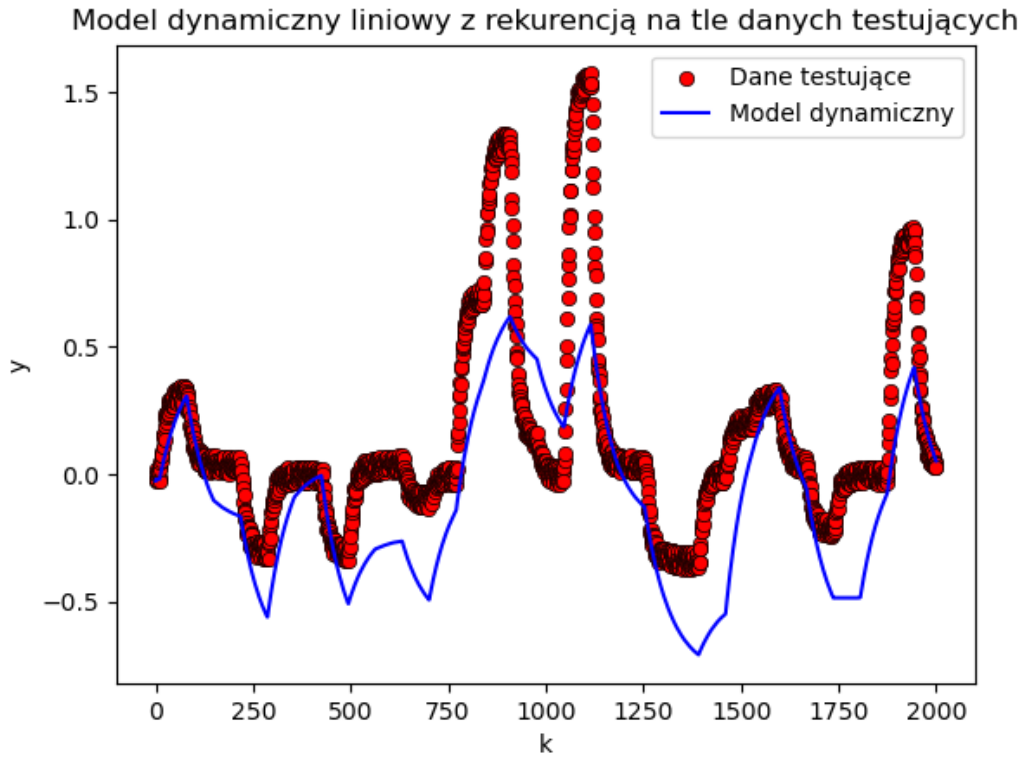

Rys. 2.7. Wyjście modelu bez rekurencji dla N=2 na tle danych uczących



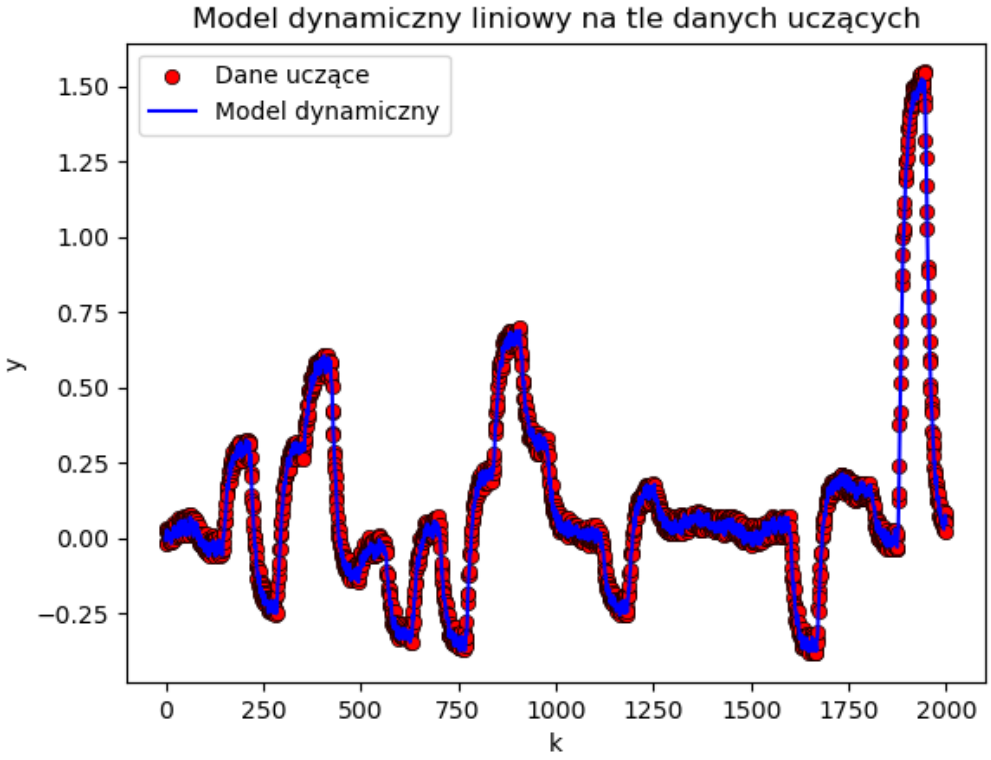

Rys. 2.8. Wyjście modelu z rekurencją dla N=2 na tle danych uczących



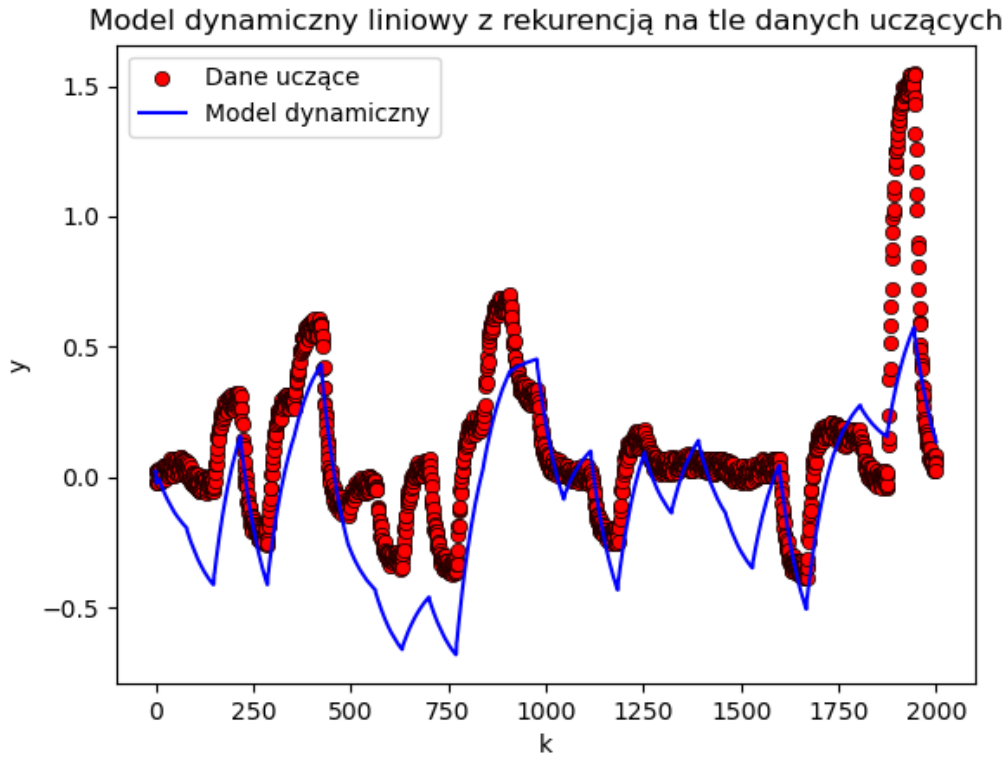

Rys. 2.9. Wyjście modelu bez rekurencji dla N=2 na tle danych testowych



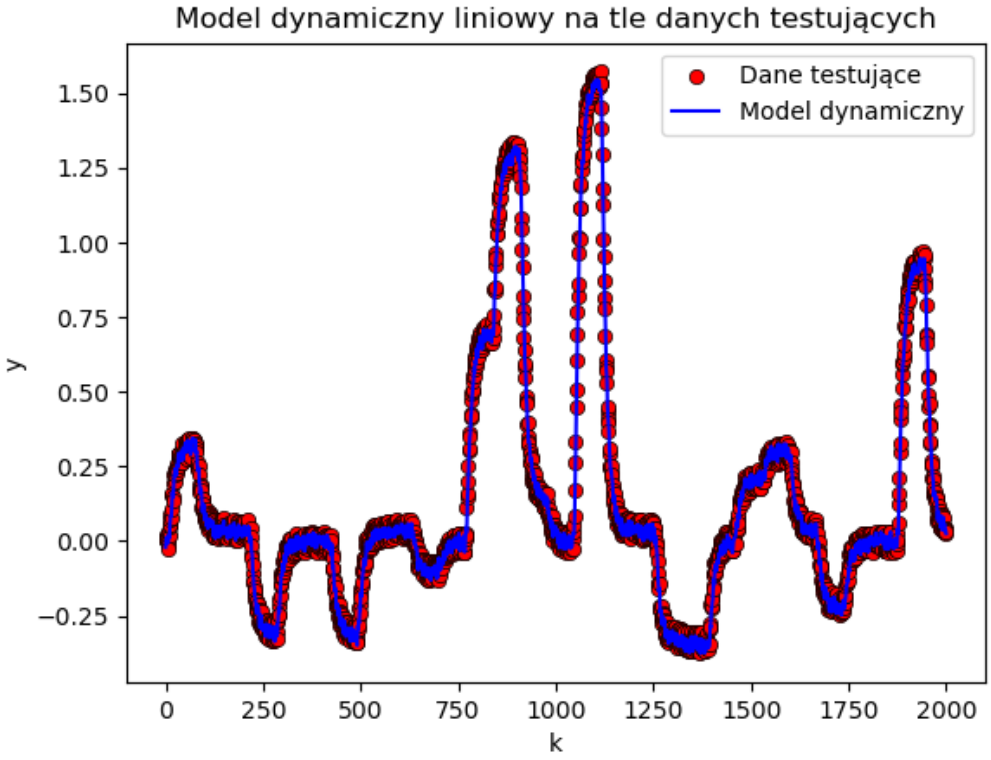

Rys. 2.10. Wyjście modelu z rekurencją dla N=2 na tle danych testowych



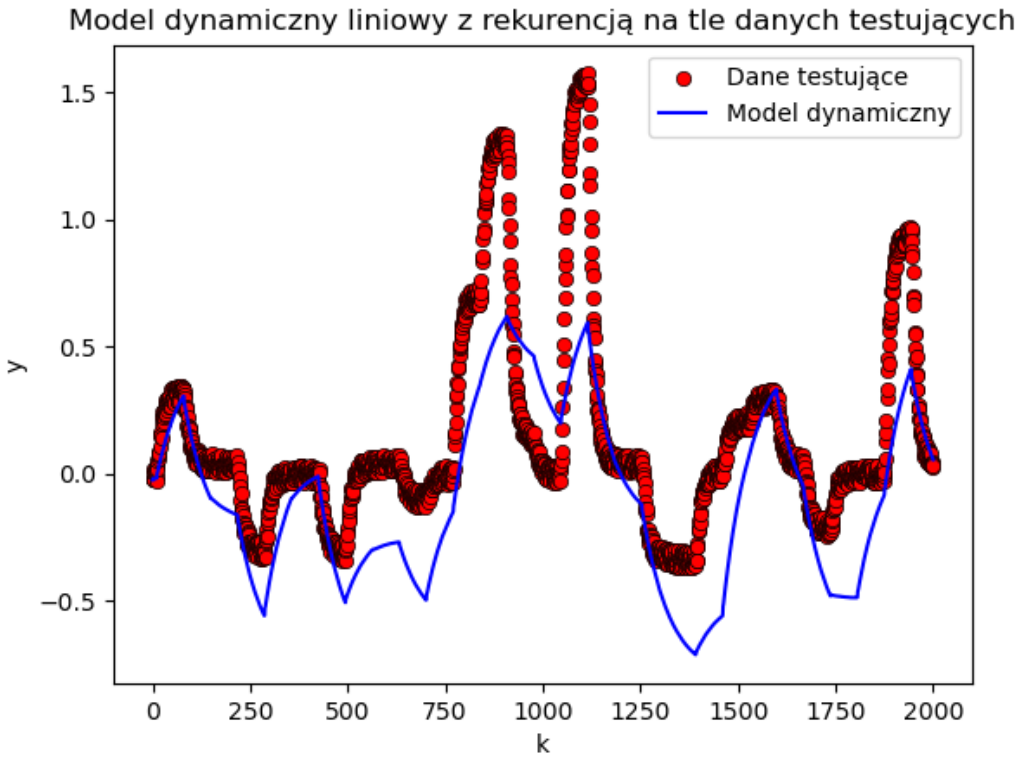

Rys. 2.11. Wyjście modelu bez rekurencji dla N=3 na tle danych uczących




Rys. 2.12. Wyjście modelu z rekurencją dla N=3 na tle danych uczących




Rys. 2.13. Wyjście modelu bez rekurencji dla N=3 na tle danych testowych




Rys. 2.14. Wyjście modelu z rekurencją dla N=3 na tle danych testowych

Wyniki uzyskanych błędów przedstawione w tabeli:

| **N** | **1** | | 2 | | 3 | |
|---|---|---|---|---|---|---|
| **Dane** | Uczące | Testujące | Uczące | Testujące | Uczące | Testujące |
| **Z rekurencją** | 1.790 | 1.957 | 1.600 | 1.815 | 1.591 | 1.821 |
| **Bez rekurencji** | 147.047 | **219.280** | 160.445 | 237.459 | 162.571 | 240.900 |

Tab. 2.1. Uzyskane błędy dla różnych rzędów dynamiki dla modelu bez i z rekurencją

Na czerwono zaznaczono najlepszy model i jego błąd względem modelu z rekurencją. Względem modelu bez rekurencji najlepszym modelem byłby model rzędu drugiego. Bez rekurencji błąd dla danych uczących maleje wraz ze wzrostem rzędu, dla danych testujących od rzędu drugiego zaczyna rosnąć. Ogólnie, każdy z przedstawionych modeli bez rekurencji działa poprawnie, błędy osiągane przez modele są małe. Z rekurencją natomiast modele na dobrą sprawę nie działają. Błędy są znaczące, widać także dużą rozbieżność na wykresach. Co ciekawe, wzrost rzędu dynamiki nawet nie poprawia wyników. Prawdopodobnie jest to spowodowane tym, że wyjście modelu jest tak bardzo różne od faktycznych danych, że wykorzystywanie go jeszcze więcej razy (przez wzrost rzędu dynamiki) jedynie powiększa uzyskiwany błąd. Dla modelu z rekurencją zdecydowanie potrzebny jest model nieliniowy.

## 2.3. Modele dynamiczne nieliniowe

Program modelujący posiada teraz dwa parametry: N - rząd dynamiki oraz K - stopień wielomianu. Tym razem implementacja metody najmniejszych kwadratów wygląda następująco:

```
Y = Y[N:]
M = []
for i in range(N-1, len(train_data[:, 0])-1):
    row = []
    for j in range(N):
        for k in range(1, K + 1):
            row.append(train_data[i - j, 0] ** k)
            row.append(train_data[i - j, 1] ** k)
    M.append(row)
M = np.array(M)
w = np.linalg.lstsq(M, Y, rcond=None)[0]
```

Macierz M ma zatem postać na podobieństwo macierzy M dla modeli liniowych, z tą różnicą, że kazda kolumna $x_i$ i $y_i$ jest podnoszona do kolejnych potęg naturalnych aż do stopnia K. Ponownie, w pierwszej linijce przedstawionego kodu ograniczamy wielkość wektora wartości y, a wektor w jest liczony w taki sam sposób. Obliczenie wyjścia modelu zostało zrealizowane następująco:

```
ymod_train = []
# Inicjalizacja modelu
ymod_train[:N] = train_data[:N, 1]
# Wyjście modelu dla danych uczących

for i in range(N-1, len(train_data[:, 0])-1):
    # Stworzenie wiersza macierzy na podobieństwo macierzy M,
    # ale z rekurencją, a więc z wyjściem modelu
    row = []
    for j in range(N):
        for k in range(1, K + 1):
            row.append(train_data[i - j, 0] ** k)
            row.append(ymod_train[i - j] ** k)
    # Obliczenie wyjścia modelu dla danych uczących
    temp = 0
    for i in range(len(row)):
        temp += w[i] * row[i]
    ymod_train.append(temp)
```

przy czym jest to liczenie wyjścia modelu z rekurencją dla danych uczących (dla innych danych naturalnie zmienia się wykorzystywana tablica danych oraz w modelu bez rekurencji wykorzystujemy dane y zamiast wyjścia modelu ymod). Na początku tworzony jest wiersz na podobieństwo wierszy macierzy M, z tą różnicą, że w miejscu danych y pojawia się wyjście modelu ymod. Wiersz ten jest wykorzystywany tylko do liczenia wyjścia, jest tworzony dla ułatwienia obliczeń z indeksami odpowiednich współczynników w.

W celu przetestowania modelów oraz szukania optymalnego rozwiązania, stworzone zostały wszystkie kombinacje modeli z rzędem dynamiki z przedziału $\langle 1, 8\rangle$ oraz stopnia wielomianu z przedziału $\langle 1, 6\rangle$. Wyniki przedstawione są w poniższych tabelach:

| **N** | | 1 | | | | | | 2 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **K** | | 1 | 2 | 3 | 4 | 5 | 6 | 1 | 2 | 3 | 4 | 5 | 6 |
| **Dane** | **Typ** | / | / | / | / | / | / | / | / | / | / | / | / |
| **Ucz.** | **Bez rek.** | 1.79 | 1.70 | 1.54 | 1.54 | 1.54 | 1.54 | 1.60 | 1.45 | 1.17 | 1.17 | 1.17 | 1.16 |
| | **Z rek.** | 147.05 | 54.85 | 2.28 | 2.39 | 2.39 | 2.32 | 160.44 | 55.45 | 1.38 | 1.18 | 1.15 | 1.13 |
| **Test.** | **Bez rek.** | 1.96 | 1.83 | 1.65 | 1.65 | 1.65 | 1.65 | 1.81 | 1.60 | 1.29 | 1.28 | 1.29 | 1.29 |
| | **Z rek.** | 219.28 | 77.21 | 2.42 | 2.79 | 2.9 | 3.47 | 237.46 | 76.53 | 1.73 | 1.52 | 1.58 | 1.68 |

Tab. 2.2. Uzyskane błędy dla różnych rzędów dynamiki dla modelu bez i z rekurencją dla N=1,2

| **N** | | 3 | | | | | | 4 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **K** | | 1 | 2 | 3 | 4 | 5 | 6 | 1 | 2 | 3 | 4 | 5 | 6 |
| **Dane** | **Typ** | / | / | / | / | / | / | / | / | / | / | / | / |
| **Ucz.** | **Bez rek.** | 1.59 | 1.39 | 1.04 | 1.03 | 1.02 | 1.02 | 1.54 | 1.36 | 0.96 | 0.95 | 0.94 | 0.94 |
| | **Z rek.** | 162.57 | 56.02 | 1.17 | 1.02 | 1.00 | 0.99 | 155.29 | 55.64 | 0.98 | 0.92 | 0.90 | 0.89 |
| **Test.** | **Bez rek.** | 1.82 | 1.53 | 1.14 | 1.14 | 1.15 | 1.15 | 1.72 | 1.50 | 1.05 | 1.06 | 1.06 | 1.07 |
| | **Z rek.** | 240.90 | 78.08 | 1.44 | 1.26 | 1.30 | 1.32 | 231.03 | 77.96 | 1.17 | 1.11 | 1.17 | 1.22 |

Tab. 2.3. Uzyskane błędy dla różnych rzędów dynamiki dla modelu bez i z rekurencją dla N=3,4

| **N** | | 5 | | | | | | 6 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **K** | | 1 | 2 | 3 | 4 | 5 | 6 | 1 | 2 | 3 | 4 | 5 | 6 |
| **Dane** | **Typ** | / | / | / | / | / | / | / | / | / | / | / | / |
| **Ucz.** | **Bez rek.** | 1.46 | 1.33 | 0.91 | 0.90 | 0.89 | 0.89 | 1.38 | 1.28 | 0.87 | 0.86 | 0.86 | 0.85 |
| | **Z rek.** | 145.54 | 54.89 | 0.92 | 0.88 | 0.86 | 0.84 | 133.17 | 53.47 | 0.90 | 0.85 | 0.84 | 0.82 |
| **Test.** | **Bez rek.** | 1.62 | 1.45 | 1.00 | 1.01 | 1.02 | 1.03 | 1.52 | 1.40 | 0.98 | 0.99 | 1.00 | 1.01 |
| | **Z rek.** | 217.84 | 77.16 | 1.08 | 1.06 | 1.12 | 1.25 | 204.34 | 75.94 | 1.05 | 1.01 | 1.05 | 1.17 |

Tab. 2.4. Uzyskane błędy dla różnych rzędów dynamiki dla modelu bez i z rekurencją dla N=5,6

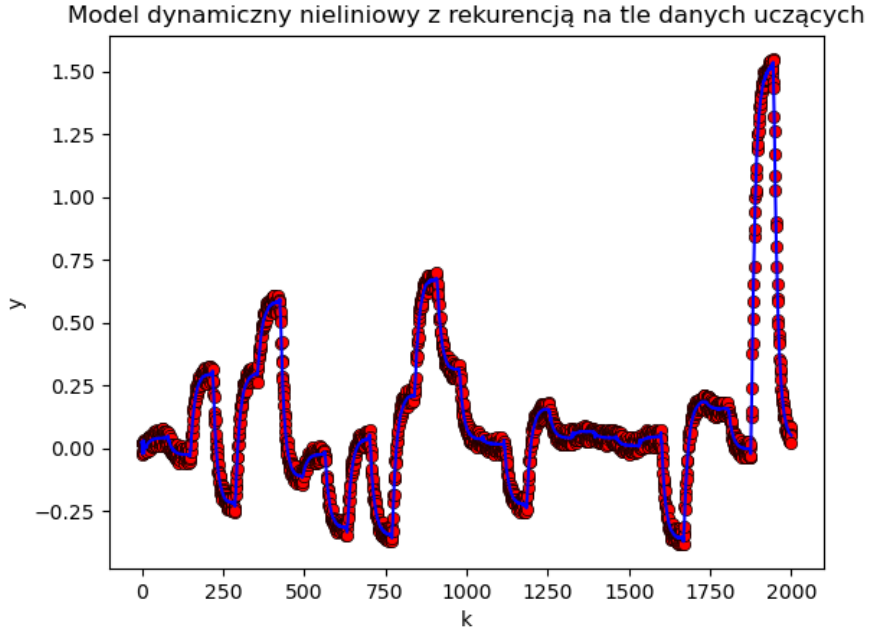
| **N** | | 7 | | | | | | 8 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **K** | | 1 | 2 | 3 | 4 | 5 | 6 | 1 | 2 | 3 | 4 | 5 | 6 |
| **Dane** | **Typ** | / | / | / | / | / | / | / | / | / | / | / | / |
| **Ucz.** | **Bez rek.** | 1.31 | 1.24 | 0.86 | 0.84 | 0.84 | 0.83 | 1.28 | 1.20 | 0.85 | 0.83 | 0.83 | 0.82 |
| | **Z rek.** | 122.30 | 51.57 | 0.88 | 0.83 | 0.82 | 0.80 | 118.56 | 50.16 | 0.87 | 0.82 | 0.81 | 0.79 |
| **Test.** | **Bez rek.** | 1.46 | 1.36 | 0.95 | 0.96 | 0.98 | 1.01 | 1.44 | 1.34 | 0.93 | 0.94 | 0.95 | 0.98 |
| | **Z rek.** | 190.62 | 73.96 | 1.02 | 0.98 | 1.05 | 1.21 | 183.29 | 71.74 | 1.00 | **0.94** | 1.03 | 1.22 |

Tab. 2.5. Uzyskane błędy dla różnych rzędów dynamiki dla modelu bez i z rekurencją dla N=7,8

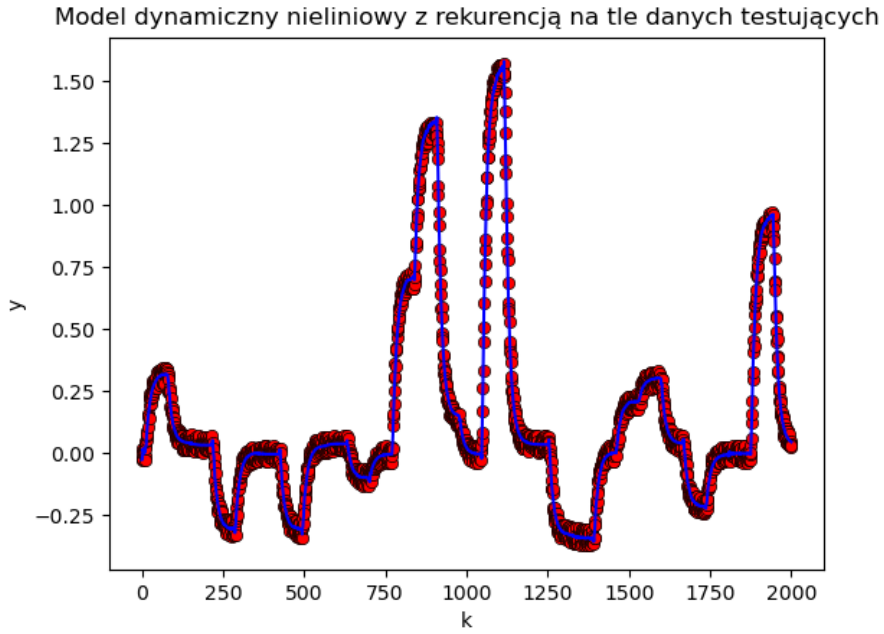
Najlepszym modelem w trybie z rekurencją jest model dla rzędu dynamiki równym 8 oraz stopnia wielomianu równego 4, błąd dla tego modelu został zaznaczony na czerwono w tabeli. Niezależnie od rzędu dynamiki, aby model z rekurencją działał poprawnie, wymagany jest stopień wielomianu równy co najmniej 3. Dla modelu z każdym omawianym rzędem dynamiki, z wyjątkiem modelu dla N=1, stopień wielomianu dający najlepszy wynik to 4, co pokrywa się z wynikami dla danych statycznych. Zatem tutaj także widzimy efekty zbyt dużego dopasowywania się modelu dla danych uczących dla wyższych stopni wielomianu (dla zbioru uczącego błąd staje maleje wraz ze wzrostem stopnia wielomianu), co prowadzi do wzrostu błędu dla danych testowych. W trybie bez rekurencji każde zwiększenie zarówno rzędu dynamiki jak i stopnia wielomianu prowadzi do zmniejszenia błędu dla danych trenujących, dla danych testowych każde zwiększenie rzędu dynamiki również prowadzi do zmniejszenia błędu, ale ponownie, najmniejszy błąd uzyskiwany jest dla stopnia wielomianu równego 3 lub 4. Prawdopodobnie, dla kolejnych rzędów dynamiki, błąd dla modelu w trybie rekurencyjnym także maleje, jednak dalsze testy nie były

przeprowadzane z uwagi na długość obliczeń, dlatego ostatecznie za najlepszy model uznany jest model dla N=8 i K=4. Dla danych uczących model z rekurencją często osiąga mniejszy błąd od modelu bez rekurencji, dla danych testowych te błędy, zwłaszcza dla wysokich rzędów dynamiki, są bardzo podobne, z niewielką przewagą jednak modelów bez rekurencji.

Tak wygląda wyjście najlepszego modelu na tle obu zbiorów:

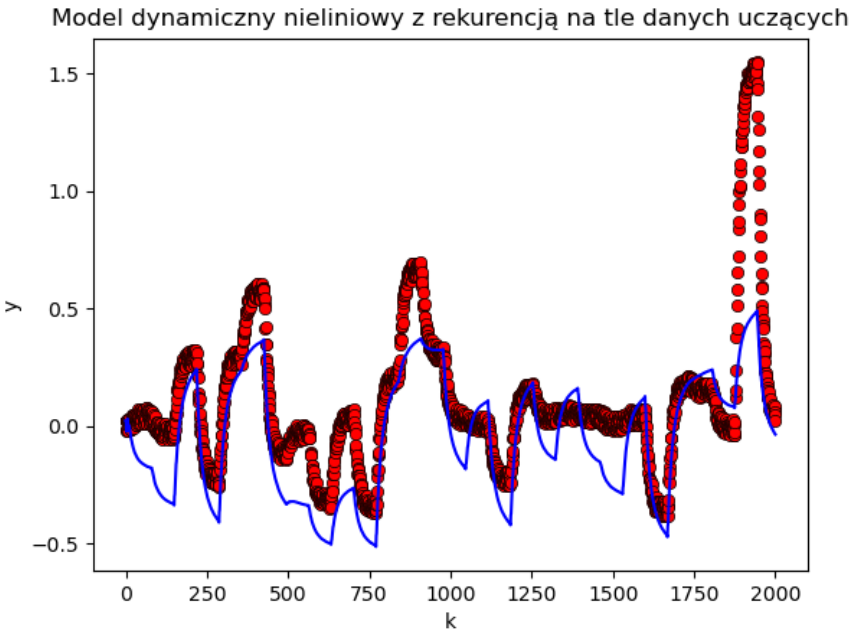



Rys. 2.15. Wyjście modelu z rekurencją dla N=8 i K=4 na tle danych uczących

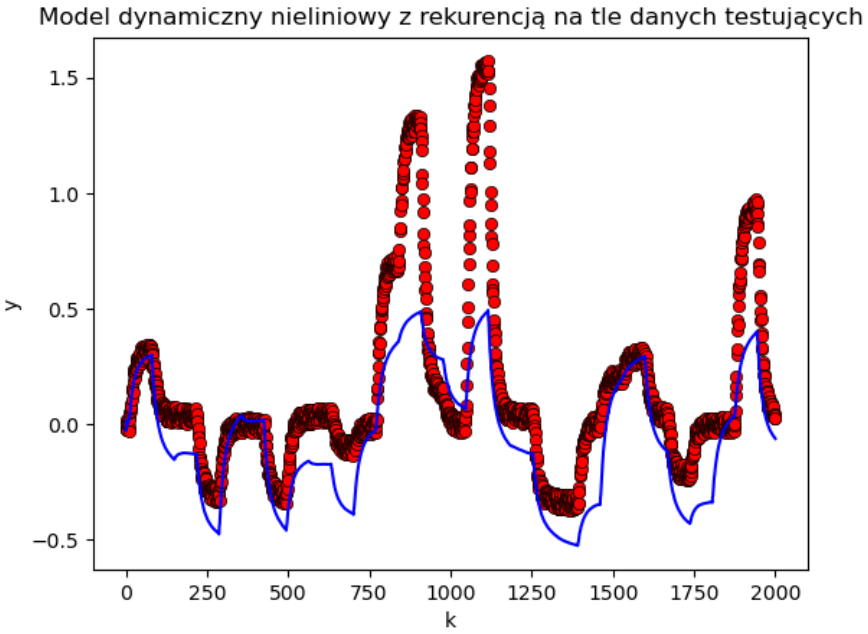



Rys. 2.16. Wyjście modelu z rekurencją dla N=8 i K=4 na tle danych testowych

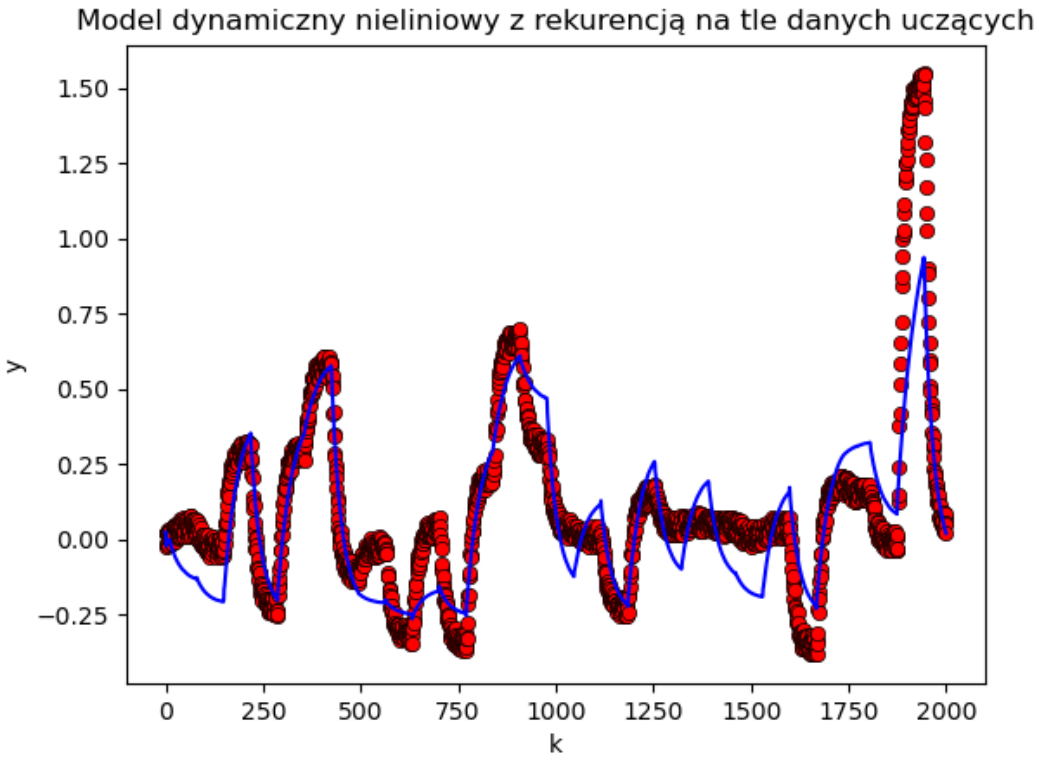
Jak widać, model dobrze naśladuje dane. Podobnie jak w przypadku testowania modelów statycznych, nie będę prezentować każdego wykresu z modeli przedstawionych w tabelach, poniżej pokażę model z rekurencją (bez rekurencji wszystkie modele wyglądają niemal identycznie do tych, co na rys. 2.3, 2.5 itd.) działający słabo, średnio i dobrze, kolejno dla parametrów: N=8, K=1; N=3, K=2; N=5, K=4.



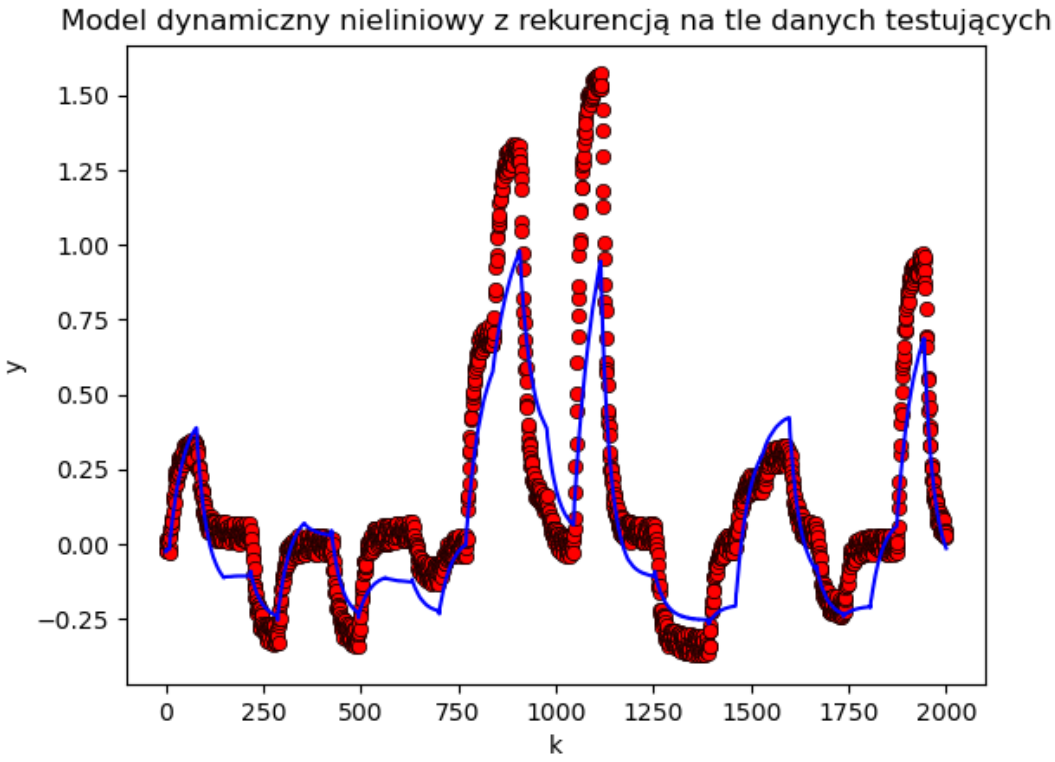

Rys. 2.17. Wyjście modelu z rekurencją dla N=8 i K=1 na tle danych uczących



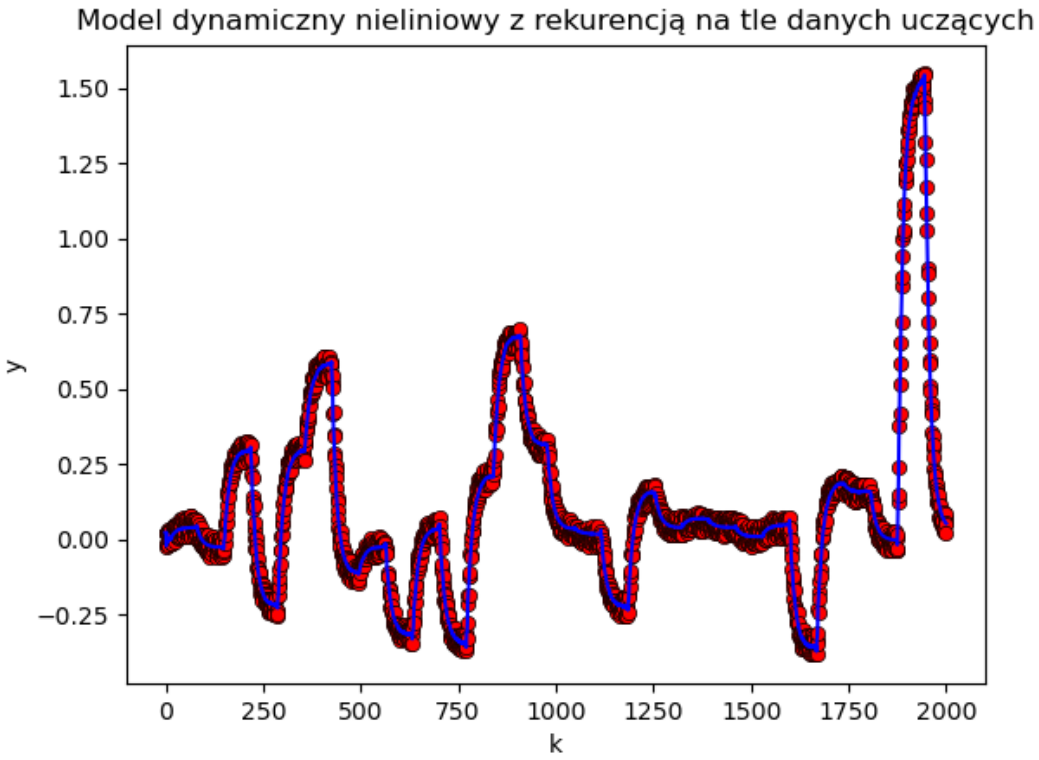

Rys. 2.18. Wyjście modelu z rekurencją dla N=8 i K=1 na tle danych testowych



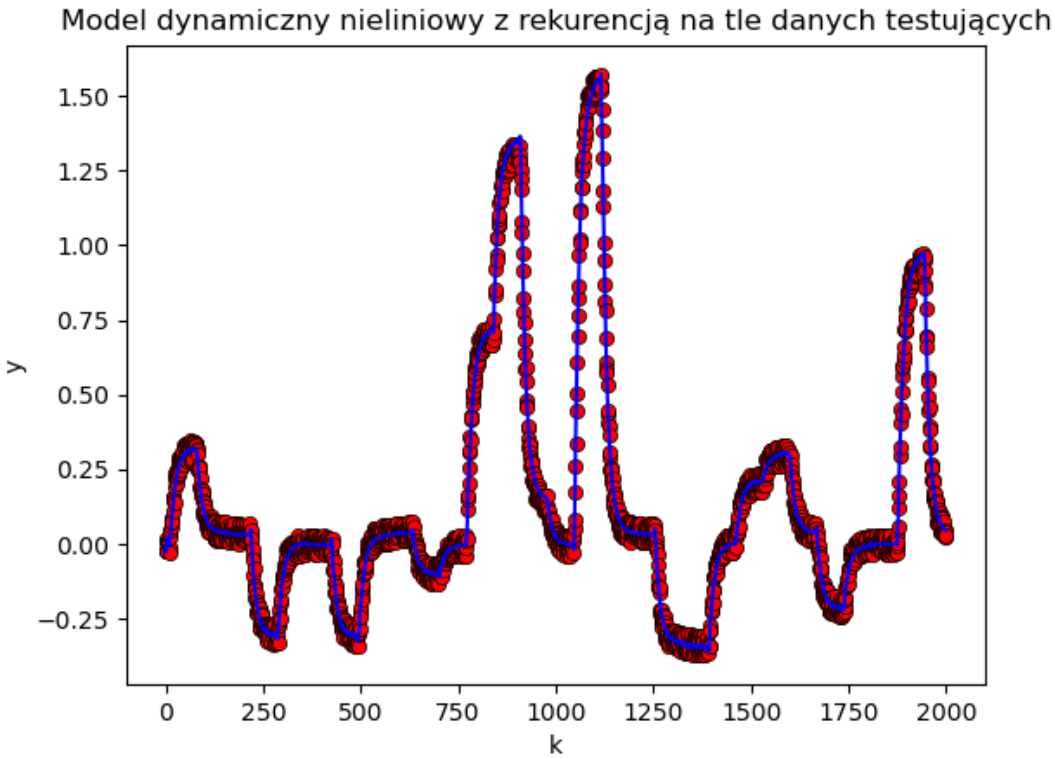

Rys. 2.19. Wyjście modelu z rekurencją dla N=3 i K=2 na tle danych uczących




Rys. 2.20. Wyjście modelu z rekurencją dla N=3 i K=2 na tle danych testowych




Rys. 2.21. Wyjście modelu z rekurencją dla N=5 i K=4 na tle danych uczących

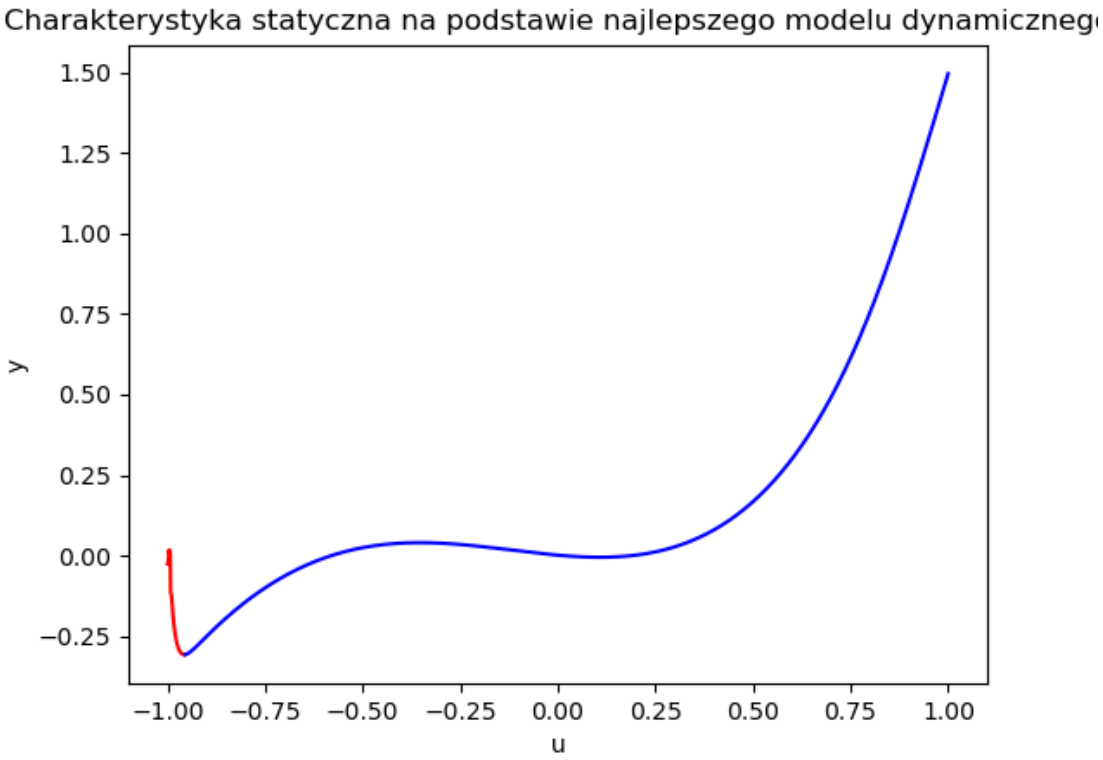



Rys. 2.22. Wyjście modelu z rekurencją dla N=5 i K=4 na tle danych testowych

Na podstawie najlepszego modelu dynamicznego utworzona została także charakterystyka statyczna, stworzona metodą symulacyjną. Kod odpowiadający za stworzenie charakterystyki:

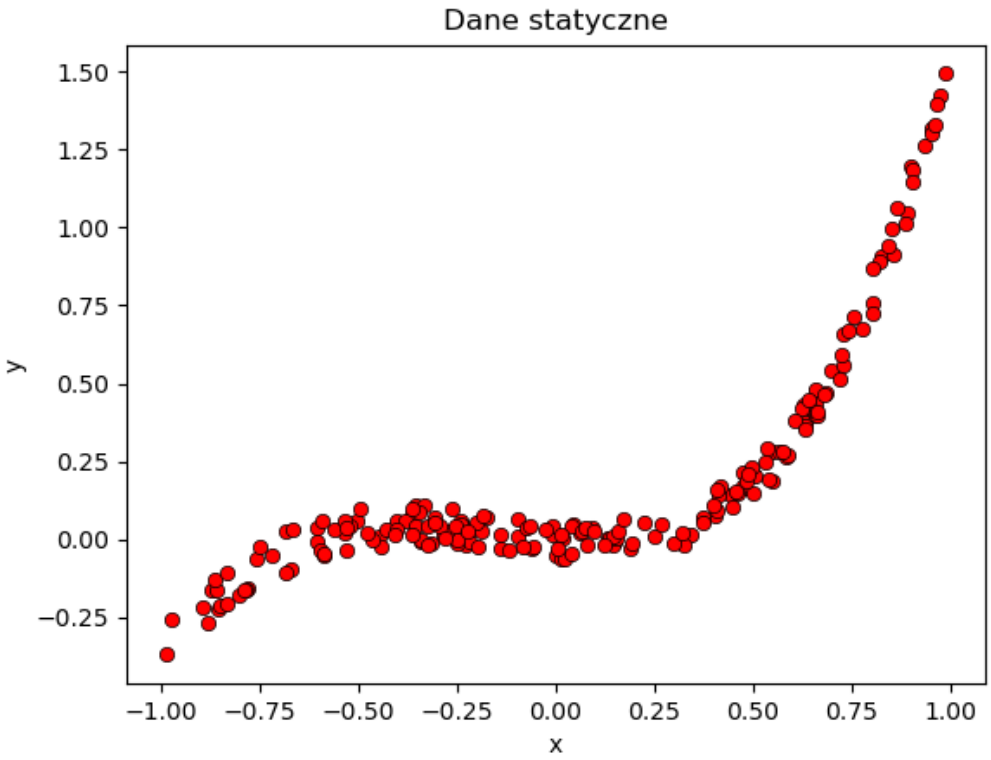
```
u = np.linspace(-1, 1, 2000)
w, _, _ = nonlinear_dynamic_model(N, K, recursive, False)
y = []
# Inicjalizacja modelu
y[:N] = test_data[:N, 1]
for i in range(N-1, len(u)-1):
    # Stworzenie wiersza macierzy na podobieństwo macierzy M,
    # ale z rekurencją, a więc z wyjściem modelu
    row = []
    for j in range(N):
        for k in range(1, K + 1):
            row.append(u[i - j] ** k)
            row.append(y[i - j] ** k)
    # Obliczenie wyjścia modelu dla danych testujących
    temp = 0
    for i in range(len(row)):
        temp += w[i] * row[i]
    y.append(temp)
```

Współczynniki w wywodzą się z uczenia modelu o parametrach N=8 i K=4. Następnie dla każdej próbki sygnału wejściowego u z przedziału od 0 do 2000 i skoku równym 1 jest liczone wyjście modelu na podstawie otrzymanych parametrów. Charakterystyka przedstawia się następująco:




Rys. 2.23. Charakterystyka statyczna na podstawie najlepszego modelu dynamicznego

Dla przypomnienia ponownie charakterystyka statyczna otrzymana w poprzednim zadaniu:




Rys. 2.24. Dane statyczne

Widoczny jest nienaturalny skok z lewej strony przedziału sygnału wejściowego, zaznaczony kolorem czerwonym. Jest on spowodowany tym, że w danych uczących nie znajdują się żadne punkty o wartościach u leżących blisko lewego krańca badanego przedziału. Skutkuje to nienaturalnym zachowaniem się charakterystyki statycznej. Dla punktów należących do przedziału, w którym znajdowały się dane uczące, charakterystyka statyczna wygląda poprawnie (kolor niebieski na wykresie), zarówno kształt, jak i osiągane wartości, są bardzo podobne, co jest zgodne z oczekiwaniami i świadczy o poprawnym działaniu programu.

# 3. Sieci neuronowe

## 3.1. Identyfikacja modelu

Do stworzenia sieci neuronowej wykorzystana została biblioteka Keras (na bazie TensorFlow). Jako funkcję aktywacji wykorzystane zostało Leaky ReLu (bez skalowania danych pozwala na zarówno ujemne jak i dodatnie wyjście modelu). Rząd modelu wynosił 8, zgodnie z rzędem najlepszego modelu z poprzedniego zadania. Przed trenowaniem modelu należało w odpowiedni sposób przygotować dane:

```
data = train_data if data_type else test_data
# Stworzenie wektorów danych wejściowych i wyjściowych
input_data = []
output_data = []
for i in range(N-1, len(data[:, 0])):
    row = []
    for j in range(1, N):
        row.append(data[i - j, 0])
        row.append(data[i - j, 1])
    input_data.append(row)
    output_data.append(data[i, 1])
input_data = np.array(input_data)
if recursive:
    input_data = input_data.reshape((input_data.shape[0], N-1, 2))
return input_data, np.array(output_data)
```

Jak widać, dane mają inną postać w zależności od rekurencji. W przypadku danych dla modelu bez rekurencji są one po prostu listą (np.arrray) kolejno kolumny x i y odpowiednich danych. W przypadku modelu z rekurencją, w danych pojawia się informacja o między innymi rzędzie dynamiki, tego wymaga wykorzystanie warstwy LSTM z biblioteki Keras, a więc warstwy dla modelu rekurencyjnego. Następnie w celu trenowania modelu wykorzystano:

```
# Stworzenie modelu
model = Sequential()
if recursive:
    model.add(LSTM(k, input_shape=(N-1, 2), activation='leaky_relu'))
else:
    model.add(Dense(k, input_dim=2 * (N - 1), activation='relu'))
model.add(Dense(1, activation='leaky_relu'))

# Uczenie modelu
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(input_data, output_data, epochs=100)
```

Czwarta linijka to dodanie warstwy LSTM, a więc stworzenie jednej warstwy ukrytej o k neuronów dla modelu z rekurencją. Modelowi bez rekurencji odpowiada szósta linijka i dodanie jednej, zwykłej warstwy ukrytej. Do obu modeli w siódmej linijce także dodawana jest ostatnia, zwykła

warstwa wyjściowa. Funkcją aktywacji, jak już wspomniano wyżej, jest Leaky ReLu. Optymalizacja zachodzi według algorytmu ADAM, a funkcją straty jest błąd średniokwadratowy. Ilość epok, a więc iteracji, wynosi 100, dla takiego problemu jest to wystarczająca ilość, aby model zdążył się wytrenować. Za obliczanie predykcji odpowiada następująca część kodu:

```
# Stworzenie wektorów z danych testujących i predykcje modelu
predictions_train = model.predict(input_data)
input_data_test, output_data_test = create_data(N, False, recursive)
predictions_test = model.predict(input_data_test)

# Sprowadzenie predykcji do jednego wymiaru
predictions_train = predictions_train.flatten()
predictions_test = predictions_test.flatten()
```

Na obu zbiorze predykcji wykonywana jest metoda flatten(), aby sprowadzić je do jednego wymiaru, aby móc łatwo je zwizualizować.

## 3.2. Wyniki

Na początku warto wspomnieć, że z uwagi na naturę sieci neuronowych, wynik przy każdym wywołaniu programu nieco się różni. Wynika to z faktu, że początkowe wagi są losowe, czasem są one bliskie optymalnych, czasem zdecydowanie dalsze. Jednak zdecydowana większość wywołań daje bardzo zbliżone wyniki do przedstawianych i widać na nich pewne zależności.

Jako optymalną ilość neuronów w warstwie ukrytej przyjęto 6, większa ilość bardzo nieznacznie poprawiała wyniki, a tak naprawdę dla danych testujących ich praktycznie nie poprawiała. Otrzymane wyniki poszczególnych modeli przedstawiono w poniższych tabelach:

| | 1 neuron bez rek. | 1 neuron z rek. | 6 neuronów bez rek. | 6 neuronów z rek. |
|---|---|---|---|---|
| Dane uczące | 1.745 | 3.696 | 1.075 | 0.898 |
| Dane testujące | 2.084 | 4.572 | 1.338 | 1.104 |

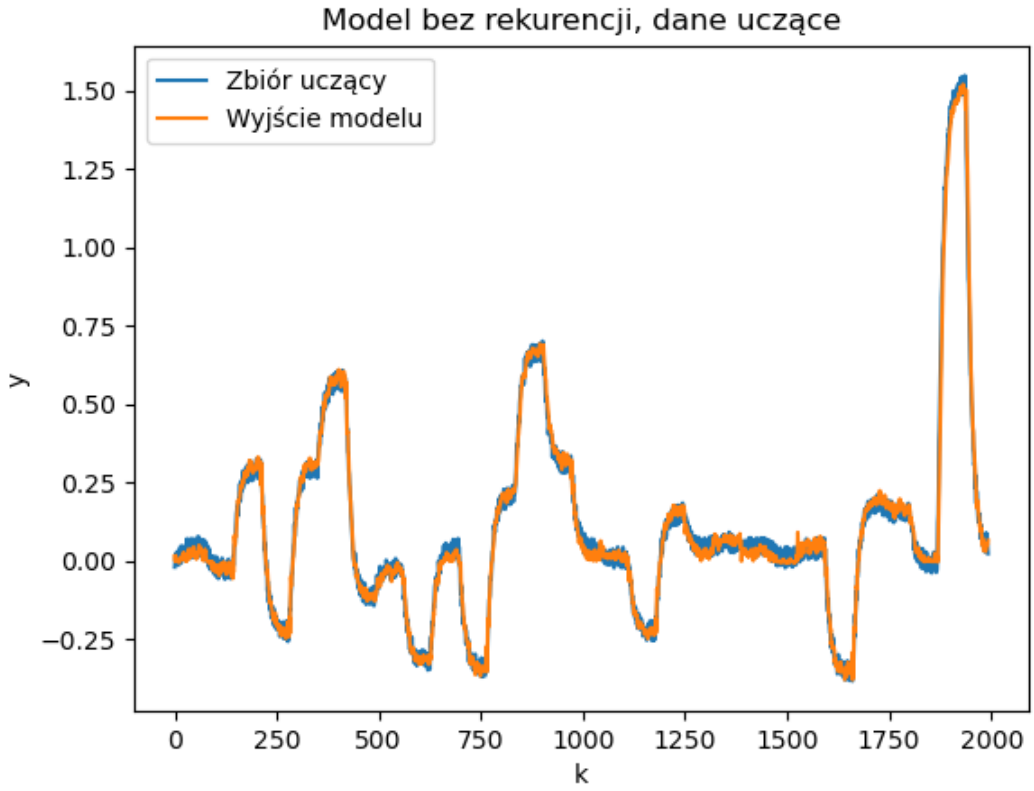
Tab. 3.1. Błędy dla różnych wielkości warstwy ukrytej

| | 1 neuron bez rek. | 1 neuron z rek. | 6 neuronów bez rek. | 6 neuronów z rek. |
|---|---|---|---|---|
| Dane uczące | 8.88e-04 | 20.00e-4 | 5.56e-04 | 4.53e-04 |

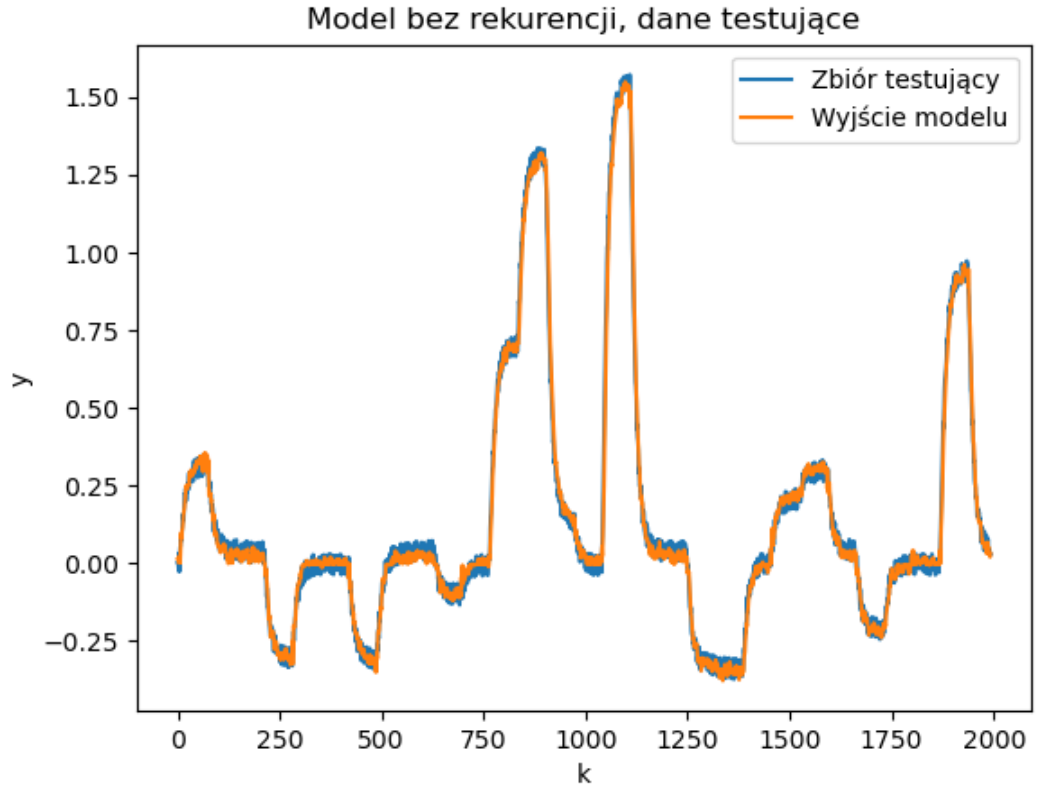
Tab. 3.2. Wynik funkcji straty dla różnych wielkości sieci neuronowej

Jak widać, w przypadku tylko jednego neuronu, oba modele i tak działają przyzwoicie, przy czym lepsze wyniki uzyskane zostały dla modelu bez rekurencji, zarówno dla straty oraz ogólnego błędu. W przypadku warstwy ukrytej składającej się z sześciu neuronów, model z rekurencją daje nieznacznie lepsze wyniki od modelu bez rekurencji w obu kryteriach. Niemniej jednak, oba modele dają bardzo dobre wyniki dla większej warstwy ukrytej.

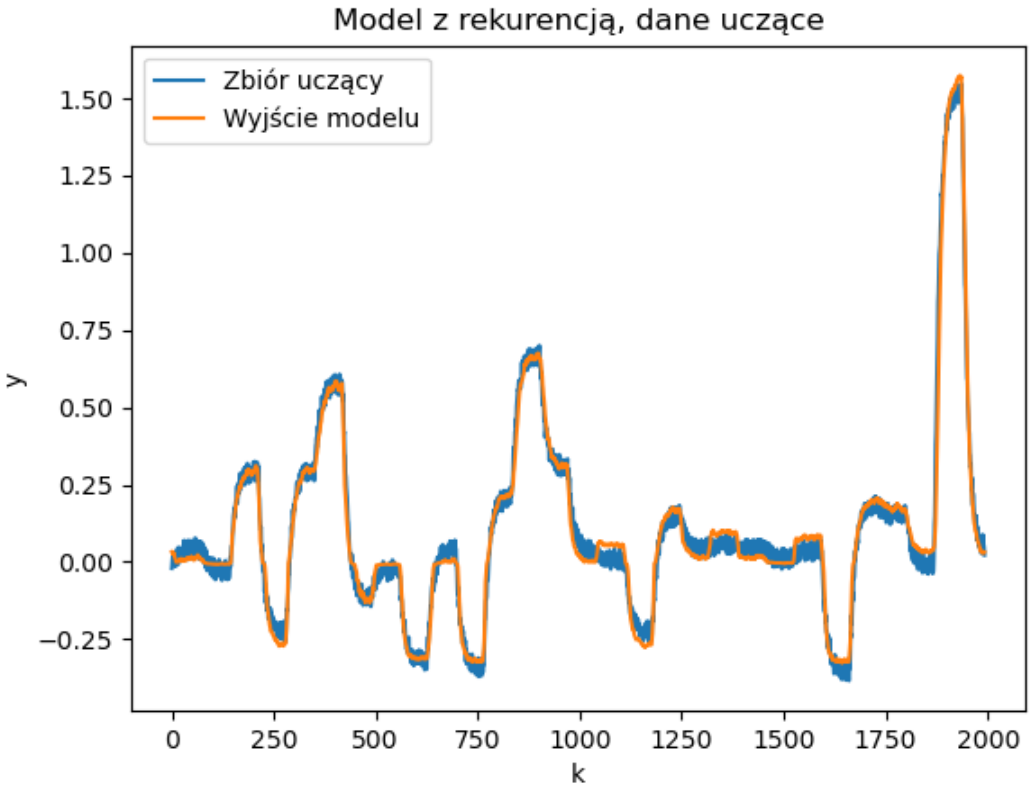
Poniżej przedstawiono odpowiednie wykresy wyjść modelu na tle obu zbiorów danych dla każdego przypadku:



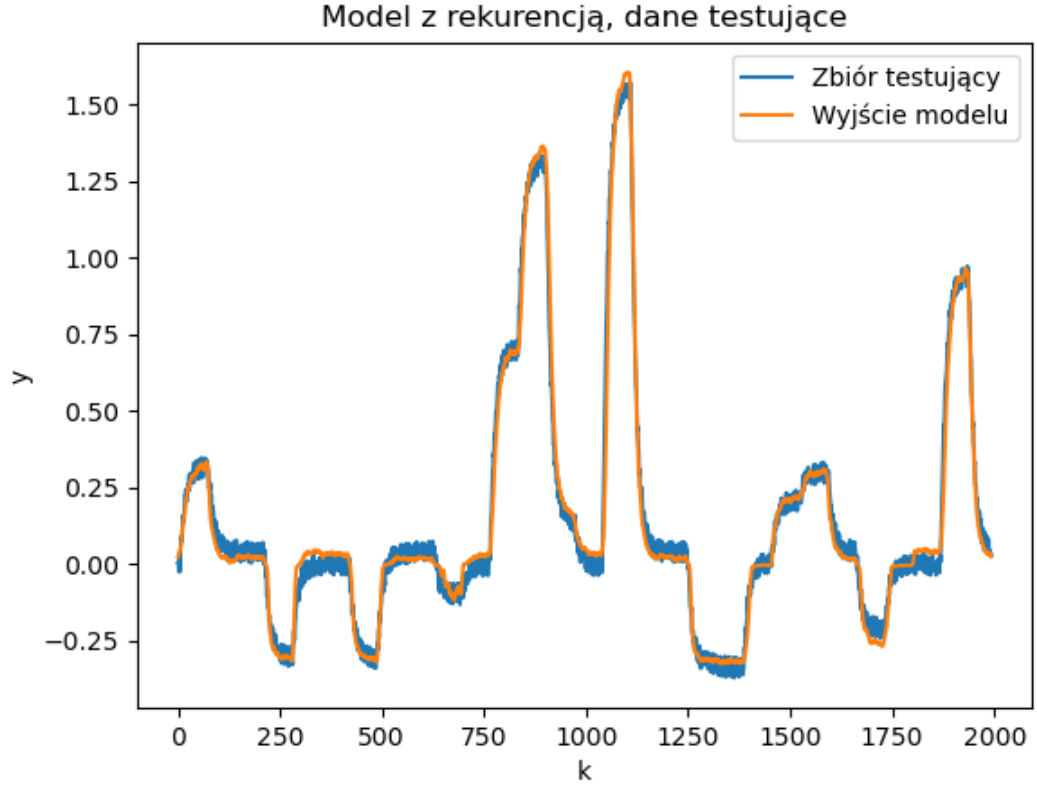

Rys. 3.1. Wyjście modelu bez rekurencji dla jednego neuronu na tle danych uczących



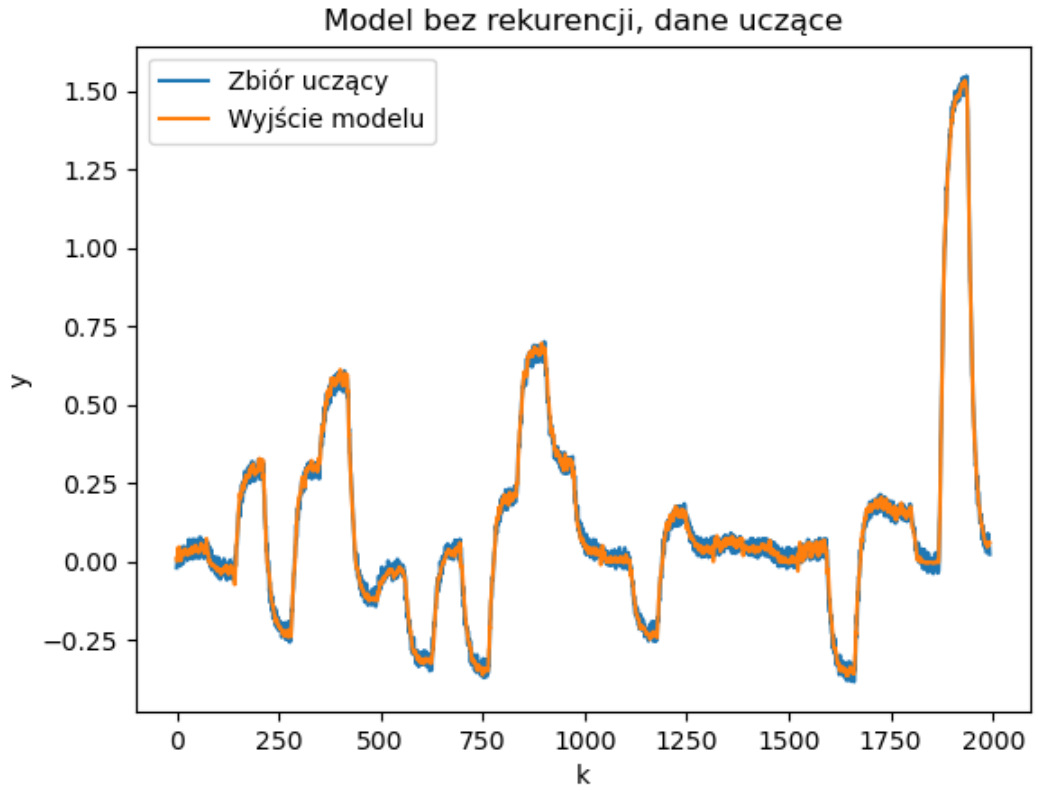

Rys. 3.2. Wyjście modelu bez rekurencji dla jednego neuronu na tle danych testowych



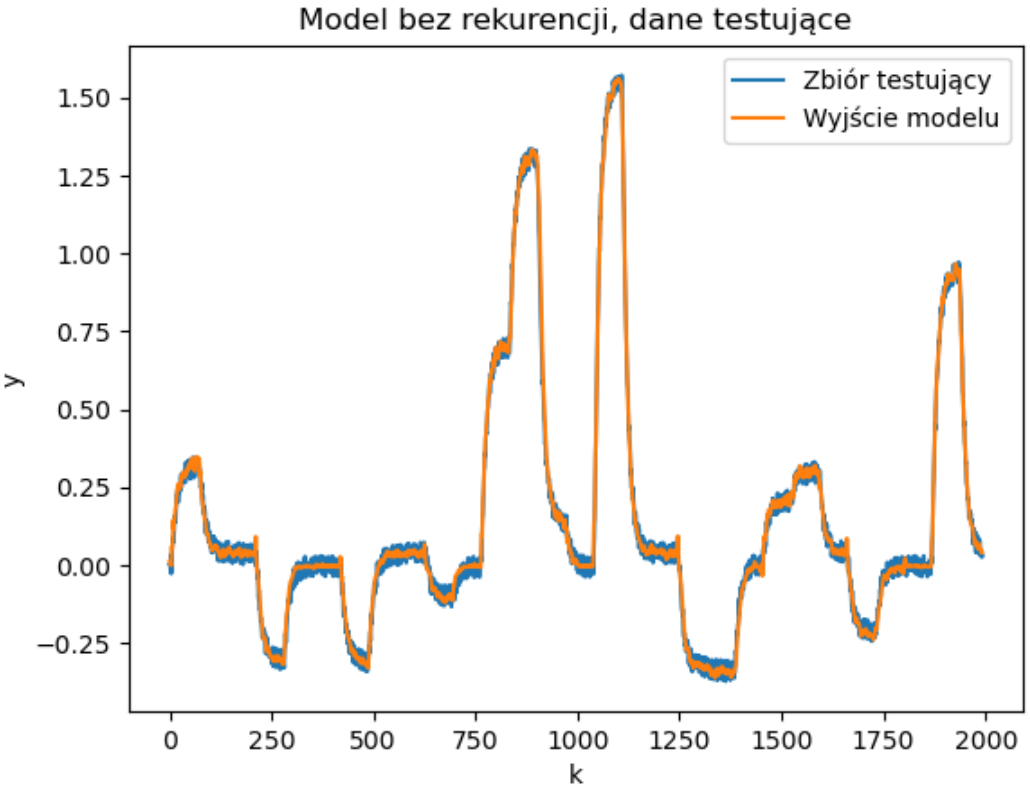

Rys. 3.3. Wyjście modelu z rekurencją dla jednego neuronu na tle danych uczących



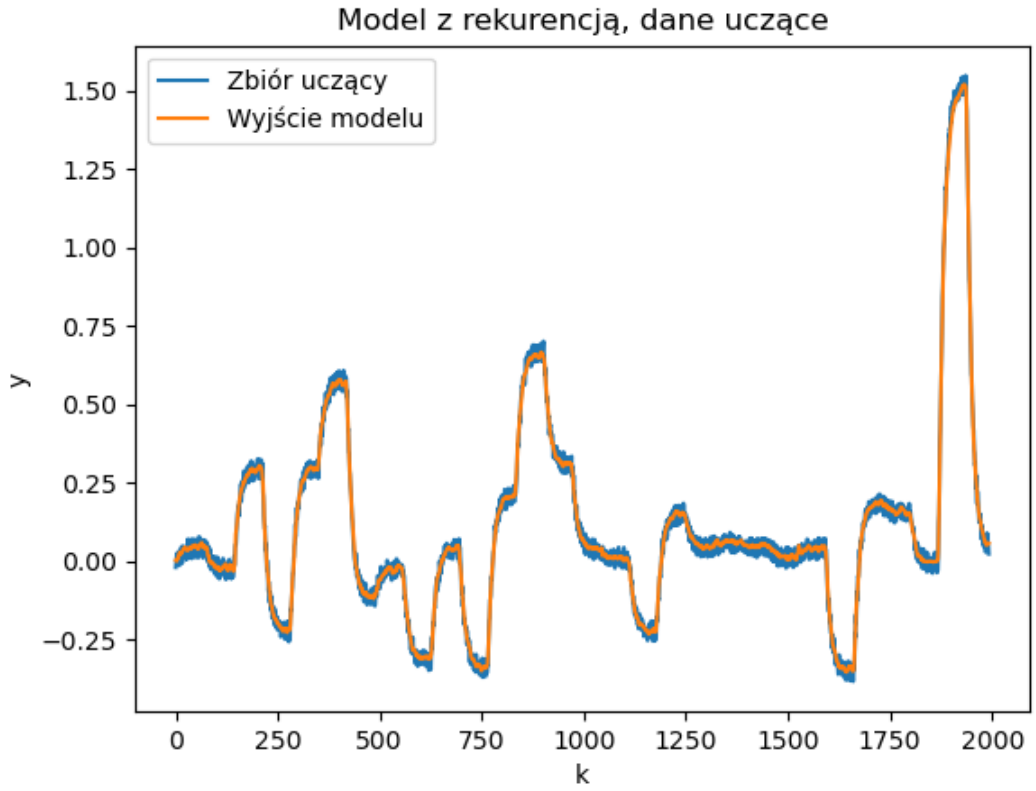

Rys. 3.4. Wyjście modelu z rekurencją dla jednego neuronu na tle danych testowych



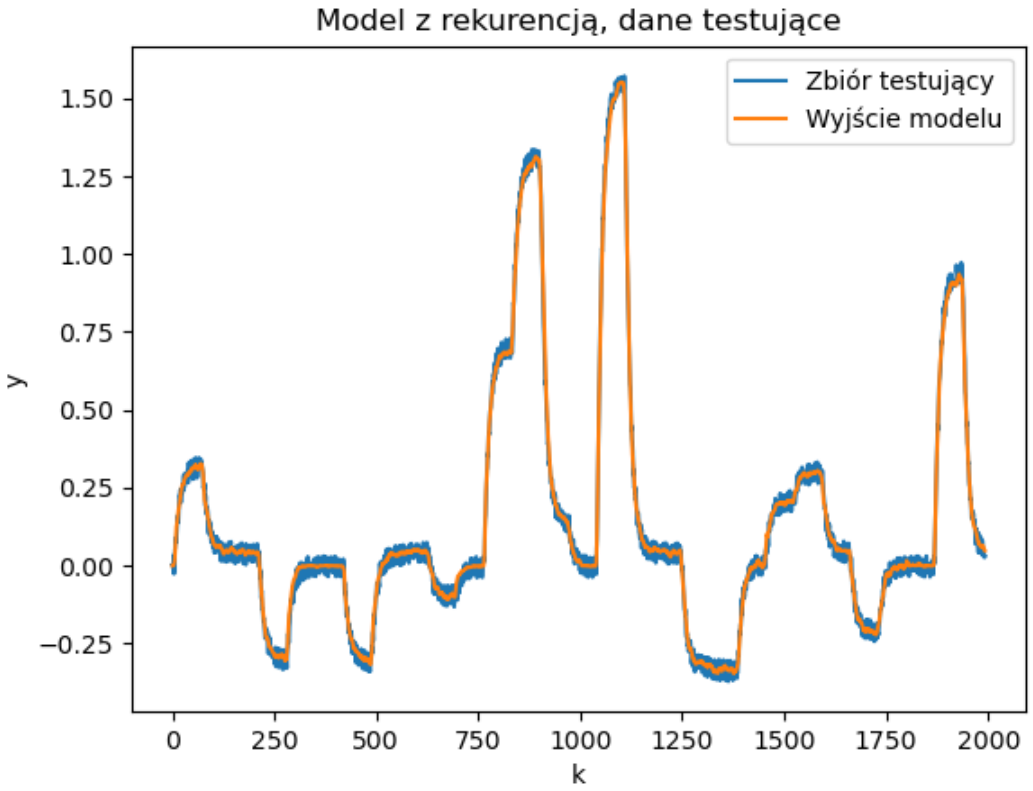

Rys. 3.5. Wyjście modelu bez rekurencji dla sześciu neuronów na tle danych uczących




Rys. 3.6. Wyjście modelu bez rekurencji dla sześciu neronów na tle danych testowych




Rys. 3.7. Wyjście modelu z rekurencją dla sześciu neuronów na tle danych uczących




Rys. 3.8. Wyjście modelu z rekurencją dla sześciu neronów na tle danych testowych

Widać, że wszystkie modele całkiem dobrze naśladują rzeczywisty proces, nawet w niewielkim stopniu szumy, które występują w danych. Model z rekurencją jest lepszy od modelu bez rekurencji, gdy sieć neuronowa będzie posiadała chociaż kilka neuronów w warstwie ukrytej.

W przypadku, gdyby z jakiegoś powodu był problem z plikami źródłowymi, wszystkie także znajdują się na repozytorium: `https://github.com/vpow10/MODI`.