POLITECHNIKA WARSZAWSKA

Wydział Elektroniki i Technik Informacyjnych

Instytut Systemów Elektronicznych

Jan Kurdel

Numer albumu: 214460

Praca inżynierska

Identyfikacja nieliniowych obiektów dynamicznych metodą uogólnionej regresji postępującej z ortogonalizacją

Praca wykonana pod kierunkiem:

dr inż. Stanisława Jankowskiego

Warszawa 2012

Identyfikacja nieliniowych obiektów dynamicznych metodą uogólnionej regresji postępującej z ortogonalizacją

Praca opisuje identyfikację obiektów dynamicznych metodą uogólnionej regresji postępującej z ortogonalizacją. Identyfikacja została przeprowadzona na przykładzie nieliniowego obiektu dynamicznego opisanego równaniem Van der Pola. Jakość modelu została następnie oceniona na podstawie błędu średniokwadratowego predykcji na 100 kroków w przód. Model był testowany dla 440 różnych warunków początkowych. Zastosowana metoda pozwoliła poprawnie odwzorować obiekt dynamiczny dla zdecydowanej większości warunków początkowych.

Identification of nonlinear dynamical objects based on Generalized Orthogonal Forward Regression

The aim of this thesis is to identify nonlinear dynamical objects through the Generalized Orthogonal Forward Regression method. The identification was based on a nonlinear dynamic object described by the Van der Pol differential equation. The quality of a model was evaluated on the basis of mean squared error of 100-step ahead prediction. The model was tested for the set of 440 different initial conditions. The results showed that the dynamic object has been correctly represented for the vast majority of initial conditions.

Spis treści

1 Cel pracy

W dzisiejszych czasach częstą koniecznością jest tworzenie modeli matematycznych rzeczywistych obiektów. Model matematyczny obiektu opisuje związek między sygnałem podawanym na wejście (sterowaniem) oraz sygnałem wyjściowym. Modelowanie takie wykorzystuje się pracując nad silnikami, robotami przemysłowymi, klimatyzacją, systemami hamulcowymi[11]. Oprócz szerokiego spektrum dziedzin, w których ma zastosowanie tworzenie modeli, istnieje szereg różnych metod ich tworzenia na postawie danych doświadczalnych, czyli metod identyfikacji obiektów.

Wśród metod identyfikacji można spotkać się metodą charakterystyk czasowych, w której badane są stany przejściowe po podaniu na wejście obiektu określonego wymuszenia, np. odpowiedzi na skok jednostkowy. Inna metodą jest metoda charakterystyk częstotliwościowych, gdzie na wejście podaje się sygnał sinusoidalny o zadanej amplitudzie i częstotliwości, a badany jest sygnał wyjściowy - jego amplituda oraz przesunięcie fazowe względem sygnału wejściowego[7].

W pracy zdecydowano się na inną metodę identyfikacji dynamiki obiektu - wykorzystano w tym celu sieć neuronową. Można spotkać się z wieloma przykładami zastosowania sieci neuronowych w zadaniu identyfikacji dynamiki jednak struktura oraz algorytmy trenowania takiej sieci mogą znacznie się różnić. W pracy zostanie przedstawiona metoda uogólnionej regresji postępującej z ortogonalizacją (Generalized Orthogonal Forward Regressoin - GOFR) oraz jej wykorzystanie do identyfikacji nieliniowego obiektu dynamicznego opisanego równaniem różniczkowym Van der Pola. Następnie zostanie dokonana ocena jakości modelowania z wykorzystaniem zaproponowanego rozwiązania. Równanie Van der Pola ma zastosowanie między innymi w dziedzinie układów elektronicznych[16]. Należy zaznaczyć, że zaproponowany algorytm może mieć zastosowanie również do obiektów opisanych innymi równaniami różniczkowymi.

2 Dynamika obiektów

Układ dynamiczny to matematyczny model, który opisuje ewolucję realnego zjawiska. Model taki pozwala na analizę reakcji badanego obiektu na pojawiające się zmiany. Układ dynamiczny z czasem ciągłym jest opisywany układem równań ruchu w postaci ogólnej[14]

$$\frac{dx}{dt} = F(x, r), \quad x \in R \tag{1}$$

gdzie:
F - odwzorowanie $F : U \to R^n$, r - zbiór parametrów kontrolnych układu, U – podzbiór R^n określający przestrzeń fazową układu.

Przestrzeń fazowa jest przestrzenią wszystkich możliwych stanów w jakich może znajdować się badany układ. Każdy stan układu jest pojedynczym punktem tej przestrzeni. Konkretne rozwiązanie układu równań ruchu $\phi_i(t_0, r)$ - dla danych wartości parametrów r i warunków początkowych zwane jest orbitą (trajektorią). Jeśli odwzorowanie F jest odwzorowaniem liniowym to układ dynamiczny jest układem liniowym, natomiast jeśli F jest odwzorowaniem nieliniowym to układ dynamiczny nazywamy układem nieliniowym.

Uniwersalnym sposobem opisu dynamiki obiektów z czasem ciągłym są równania różniczkowe. Dla opisu dynamiki obiektów z czasem dyskretnym odpowiednio stosowane są równania różnicowe[10].

2.1 Równania różniczkowe zwyczajne

Równaniem różniczkowym zwyczajnym nazywamy równanie zawierające zmienną niezależną x, nieznaną funkcję y, oraz jej pochodne $y', y'', \ldots, y^{(n)}$ [6]

$$F(x, y, y', \ldots, y^{(n)}) = 0 \tag{2}$$

gdzie $F : R^{n+2} \to R$

Warunek początkowy (Cauchy'ego) dla równania 2 określony jest poprzez:

$$\begin{aligned} y(x_0) &= y_0, \\ y'(x_0) &= y_1, \\ &\vdots \\ y^{n-1}(x_0) &= y_{n-1} \end{aligned} \tag{3}$$

gdzie $y_0, y_1, \ldots, y_{n-1}$ są zadanymi liczbami.

Stopień pochodnej w równaniu 2 decyduje o rzędzie równania różniczkowego. Równanie różniczkowe pierwszego rzędu wyraża się zatem wzorem:

$$y' = f(x, y) \tag{4}$$

Równanie 4 może zostać rozwiązane numerycznie na wiele sposobów. Można wyróżnić podział na metody jednokrokowe, które wymagają jedynie informacji z poprzedniego kroku (zatem warunek początkowy jest wystarczający do obliczenia równania różniczkowego dla kolejnego punktu) oraz metody wielokrokowe, w których rozwiązanie jest dokonowywane z wykorzystaniem znajomości kilku poprzednich kroków. Wśród metod jednokrokowych znajdują się m.in. metody Eulera oraz Rungego-Kutty.

2.2 Metody rozwiązywania równań różniczkowych

Metoda Eulera

Metoda Eulera jest najprostszą metodą numeryczną rozwiązywania równań różniczkowych. Mając dane równanie 4 oraz warunek początkowy postaci:

$$y(x_0) = y_0 \tag{5}$$

poszukujemy rozwiązania dla kolejnego momentu $x_1 = x_0 + h$, gdzie h jest krokiem metody. W tym celu możemy skorzystać z rozwinięcia szeregu Taylora w pobliżu punktu początkowego x_0:

$$y(x_0 + h) = y(x_0) + hy'(x_0) + \frac{h^2}{2!}y''(x_0) + \ldots \tag{6}$$

Korzystając tylko z dwóch pierwszych elementów szeregu Taylora otrzymamy:

$$y(x_0 + h) = y(x_0) + hy'(x_0) = y(x_0) + hf(x_0, y_0) \tag{7}$$

Powtarzając tę czynność wielokrotnie otrzymamy wzór:

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{8}$$

określany jawną metodą Eulera.

Metoda Rungego-Kutty

Kolejną metodą numerycznego rozwiązywania równań różniczkowych jest metoda Rungego-Kutty, która jest metodą wieloetapową. W każdym kroku konieczne jest obliczenie wartości funkcji f dla różnych argumentów. Wśród metod Runge-Kutty można wyróżnić różne warianty tej metody w zależności od rzędu metody. Najprostszą z nich jest metoda Runge-Kutty rzędu II. Przyjmuje ona postać:

$$y_{i+1} = y_i + \frac{h}{2}(k_1 + k_2) \tag{9}$$

gdzie

$$k_1 = f(t_i, y_i)$$
$$k_2 = f(t_i + h, y_i + hk_1)$$

Najbardziej popularną jest metoda rzędu IV, która przyjmuje postać:

$$y_{i+1} = y_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{10}$$

gdzie

$$k_1 = f(t_i, y_i)$$
$$k_2 = f(t_i + \tfrac{h}{2}, y_i + \tfrac{h}{2}k_1)$$
$$k_3 = f(t_i + \tfrac{h}{2}, y_i + \tfrac{h}{2}k_2)$$
$$k_4 = f(t_i + h, y_i + hk_3)$$

Równania różniczkowe zwyczajne wyższych rzędów

Równanie różniczkowe wyższych rzędów może zostać sprowadzone do układu równań różniczkowych rzędu pierwszego [13]. Mając dane równanie różniczkowego n-tego rzędu:

$$y^{(n)}(t) = f(t, y(t), y'(t), \ldots, y^{(n-1)}(t)), \qquad a \leqslant t \leqslant b$$

z warunkiem początkowym w postaci:

$$y(a) = \alpha_0,$$
$$y'(a) = \alpha_1,$$
$$\vdots$$
$$y^{n-1}(a) = \alpha_{n-1}$$

oraz wprowadzając nowe zmienne:

$$y_1(t) = y(t), y_2(t) = y'(t), \ldots, y_n(t) = y^{(n-1)}(t)$$

otrzymujemy układ równań różniczkowych pierwszego rzędu postaci:

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= y_3 \\ &\vdots \\ y_{n-1}' &= y_n \\ y_n' &= f(t, y_1, y_2, \ldots, y_n) \end{aligned}$$

z warunkami początkowymi postaci:

$$y_1(a) = \alpha_0, y_2(a) = \alpha_1, \ldots, y_n(a) = \alpha_{n-1}$$

Dzięki takiemu zabiegowi możliwe jest rozwiązanie równania różniczkowego wyższego rzędu metodami służącymi do rozwiązywania równań różniczkowych rzędu pierwszego.

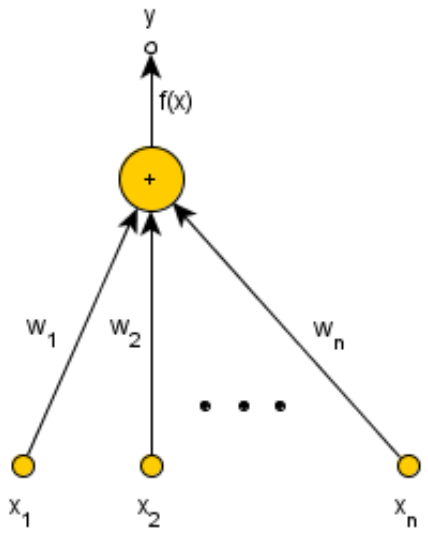
Należy zaznaczyć, że wymienione metody rozwiązywania równań różniczkowych mogą mieć zastosowanie jedynie w przypadku znajomości postaci równania różniczkowego.

3 Sieci neuronowe

Sieci neuronowe w dzisiejszych czasach mają bardzo szerokie zastosowanie. Wykorzystywane są m. in do zadań klasyfikacji, aproksymacji, predykcji, sterowania w wielu różnych dziedzinach życia. Definicja sieci neuronowej nie została dokładnie sprecyzowana ale sieci neuronowe można określić jako adaptacyjne metody tworzenia modeli nieliniowych na podstawie danych eksperymentalnych bądź też formalne modele statystyczne zbudowane na podstawie danych eksperymentalnych[12].

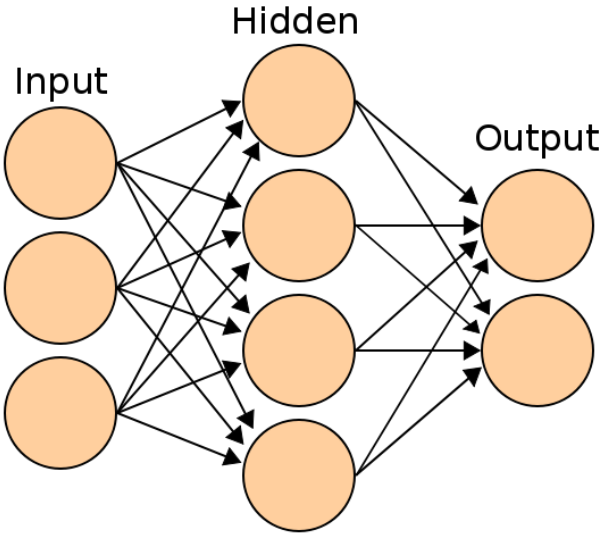
3.1 Struktura sieci neuronowych

Podstawą budowy sieci neuronowych są neurony. Budowa pojedynczego neuronu jest przedstawiona na rysunku 1. Neuron taki sumuje sygnał podany na jego wejścia x_i z odpowiednimi wagami w_i, dając na wyjście wynik y sumy przeprowadzonej przez funkcję aktywacji $f(x)$.



Rysunek 1: Budowa pojedynczego neuronu

Sieć neuronową budują grupy połączonych ze sobą neuronów. Uproszczoną strukturę sieci wielowarstwowej przedstawia rysunek 2. Pierwsza warstwa nosi nazwę warstwy wejściowej (*Input*), ostatnia warstwa sieci nosi nazwę warstwy wyjściowej (*Output*). Wszystkie pozostałe warstwy noszą nazwę warstw ukrytych (*Hidden*).



Rysunek 2: Schemat budowy wielowarstwowej sieci neuronowej
źródło: http://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg

Perceptron wielowarstwowy

Często wykorzystywanym typem sieci jest perceptron wielowarstwowy, którą stanowi jednokierunkowa wielowarstwowa sieć o neuronach typu sigmoidalnego (z sigmoidalną funkcją aktywacji)[8]. Funkcja aktywacji często przyjmuje postać $f(x) = \frac{1}{1+\exp(-\beta x)}$ określanej mianem funkcji unipolarnej, bądź też $f(x) = tgh(\beta x)$ określanej mianej funkcji bipolarnej.

Sieć RBF

Sieć RBF (Radial Basis Function) jest sztuczną siecią neuronową o radialnych funkcjach bazowych. Sieć ta ma szerokie zastosowanie i jest wykorzystywana do szeregu problemów m.in: modelowania 3D, rozpoznawania mowy, identyfikacji systemów, modelowania parametrów urządzeń elektronicznych[4]. Sieć taka składa się zazwyczaj z jednej warstwy ukrytej, gdzie funkcja aktywacji jest radialną funkcją bazową oraz warstwy wyjściowej w postaci neuronu liniowego. Radialna funkcja bazowa jest określona w sposób następujący[1]

$$G(x, c) = G(r(x, c)) \tag{11}$$

gdzie $r(x, c)$ jest odległością między punktami r i c, a centrum c jest ustalone i pełni rolę parametru funkcji. Jedną z bardziej popularnych radialnych funkcji bazowych jest funkcja

Gaussa:

$$G(r) = \exp(\frac{-r^2}{2\sigma^2}) \tag{12}$$

Często w sieciach RBF stosowane są również następujące funkcje bazowe[5]:
$G(r) = r^2 log(r)$
$G(r) = (r^2 + \sigma^2)^{\frac{1}{2}}$
$G(r) = (r^2 + \sigma^2)^{-\frac{1}{2}}$

3.2 Metody gradientowe uczenia sieci

Do najbardziej skutecznych metod uczenia sieci neuronowych należą metody gradientowe. Algorytmy te bazują na rozwinięciu w szereg Taylora funkcji celu $E(W)$ w najbliższym sąsiedztwie znanego rozwiązania[8].

$$E(w + p) = E(w) + [(g(w)]^T p + \frac{1}{2} p^T H(w) p + \dots \tag{13}$$

gdzie

$$g(w) = \nabla E = [\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_n}]^T$$

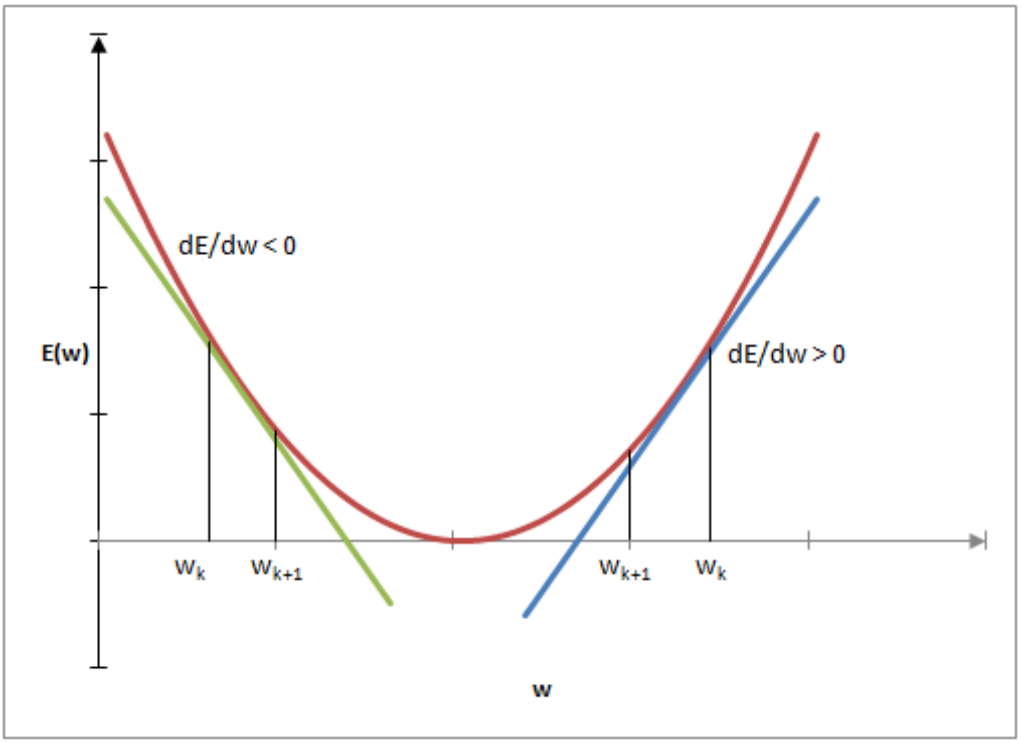
$$H(w) = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_n} \\ \vdots & & \vdots \\ \frac{\partial^2 E}{\partial w_n \partial w_1} & \cdots & \frac{\partial^2 E}{\partial w_n \partial w_n} \end{bmatrix}$$

∇E jest wektorem pierwszych pochodnych - gradientem, natomiast $H(w)$ jest macierzą drugich pochodnych - hesjanem. W praktyce są używane co najwyżej 3 pierwsze elementy równania 13. W metodzie gradientowej poszukuje się minimum funkcji celu poprzez taką aktualizację wektora kierunkowego p oraz kroku η dla punktu $w_{k+1} = w_k + \eta_k p_k$ aby prawdziwa była nierówność $E(w_{k+1}) < E(w_k)$.

Algorytm największego spadku

Metoda największego spadku chociaż jest metodą wolno zbieżną to warto ją omówić ze względu na jej prostotę. Ograniczając się do liniowej aproksymacji funkcji celu opisanej równaniem 13 zapisanej w postaci rozwinięcia szeregu Taylora otrzymujemy wzór:

$$E(w_{k+1}) = E(w) + [(g(w)]^T p \tag{14}$$



Rysunek 3: Funkcja kosztu

Aby była spełniona nierówność $E(w_{k+1}) < E(w_k)$ wystarczy spełnić warunek $[(g(w)]^T p < 0$. Wektor kierunkowy spełniający nierówność wynosi $p = -[g(w)]^T$. Graficznie działanie algorytmu przedstawia rysunek 3. Czerwoną linią została przedstawiona funkcja kosztu $E(w)$ jako funkcja jednej zmiennej w. Zieloną linią została oznaczona styczna do wykresu w punkcie w_k, gdzie pochodna jest mniejsza od zera. W tym przypadku należy przemieścić się w stronę dodatnich wartości w czyli zgodnie z kierunkiem $-\frac{dE}{dw}$. Niebieską linią została oznaczona styczna do wykresu w punkcie w_k, gdzie pochodna jest większa od zera. W tym przypadku należy przemieścić się w stronę ujemnych wartości w czyli również zgodnie z kierunkiem $-\frac{dE}{dw}$.

Algorytm Levenberga-Marquardta

Podczas gdy wcześniej nie została określona konkretna funkcji celu, tak w przypadku algorytmu Levenberga-Marquardta funkcją celu jest błąd średniokwadratowy (sum-of squares error, Mean Squared Error, MSE)[3]

$$E = \frac{1}{2}\sum_{n=1}^{N}(e_n)^2 = \frac{1}{2}\sum_{n=1}^{N}(y_n - \tilde{y_n})^2 \tag{15}$$

gdzie e_n jest błędem n-tego wzorca, y_n n-tym wzorcem, $\tilde{y_n}$ wyjściem sieci dla n-tego wzorca.

Zakładając, że aktualnie jesteśmy w punkcie w_i i chcemy przemieścić się do punktu w_{i+1}, który jest niedaleko oddalony od w_i, w celu obliczenia nowej wartości błędu e możemy skorzystać z rozwinięcia w szereg Taylora:

$$e(w_{i+1}) = e(w_i) + Z(w_{i+1} - w_i) \tag{16}$$

gdzie $Z \equiv \nabla e$. Po podstawieniu równania 16 do równania 15 wzór na błąd średniokwadratowy może zostać zapisany jako:

$$E = \frac{1}{2}[e(w_i) + Z(w_{i+1} - w_i)]^2 \tag{17}$$

Chcą zminimalizować równanie 17 ze względu na zmienną w_{i+1} otrzymujemy wzór:

$$w_{i+1} = w_i - (Z^T Z)^{-1} Z^T e(w_i) \tag{18}$$

Jednak taki sposób obliczania w_{i+1} może powodować, że krok $w_{i+1} - w_i$ będzie duży a wtedy liniowa aproksymacja szeregiem Taylora może stać się niedokładna. Z tego powodu algorytm Levenberga-Marquardta korzysta ze zmodyfikowanej funkcji celu w postaci:

$$E = \frac{1}{2}(e(w_i) + Z(w_{i+1} - w_i))^2 + \lambda(w_{i+1} - w_i)^2 \tag{19}$$

Chcą zminimalizować równanie 19 otrzymujemy ostatecznie równanie opisujące sposób obliczania wag w kolejnych iteracjach dla algorytmu Levenberga-Marquardta:

$$w_{i+1} = w_i - (Z^T Z + \lambda I)^{-1} Z^T e(w_i) \tag{20}$$

gdzie I jest macierzą jednostkową. Wartość λ zmienia się w trakcie obliczania kolejnych wartości w_{i+1}. Jeśli błąd E zmniejsza się wartość λ jest zmniejszana o określony współczynnik. W przypadku wzrostu wartości błędu wartość λ jest zwiększana o określony współczynnik oraz ponownie przyjmowana jest wartość w_i.

4 Selekcja funkcji bazowych oparta o metodę GOFR

W przypadku sieci RBF problem może stanowić wybór odpowiednich funkcji bazowych oraz odpowiedniej ich ilości. Zbyt mała ilość funkcji nie pozwoli na wystarczające zminimalizowanie błędu sieci na zbiorze uczącym. Z kolei zbyt duża ilość funkcji może doprowadzić do problemu nadmiernego dopasowania a co za tym idzie sieć traci zdolność do generalizacji. Jedną z metod rozwiązania tego problemu jest zastosowanie metody GOFR opartej o metodę OLS (Orthogonal Least Square).

4.1 Selekcja oparta o ortogonalizację oraz metodę najmniejszych kwadratów

Jedną z metod selekcji najbardziej znaczących funkcji bazowych jest metoda oparta o metodę najmniejszych kwadratów oraz ortogonalizację (OLS)[5]. Metoda ta umożliwia określenie wkładu każdej funkcji bazowej i wybranie tych najbardziej istotnych. Mając dany wektor wag sieci $w = [w_0, w_1, \ldots, w_K]^T$ oraz wektor danych uczących $d = [d_1, d_2, \ldots, d_p]^T$ można zapisać macierz G w postaci:

$$G = \begin{bmatrix} \phi_{11} & \phi_{21} & \ldots & \phi_{K1} \\ \phi_{12} & \phi_{22} & \ldots & \phi_{K2} \\ \ldots & \ldots & \ldots & \ldots \\ \phi_{1p} & \phi_{2p} & \ldots & \phi_{Kp} \end{bmatrix} \tag{21}$$

gdzie ϕ_{ji} oznacza odpowiedź i-tej funkcji radialnej na j-ty wzorzec uczący. Oznaczając przez $g_i = [g_{i1}, g_{i2}, \ldots, g_{ip}]^T$ odpowiedź i-tej funkcji radialnej na wszystkie wzorce uczące można macierz G przedstawić w postaci:

$$G = [g_1, g_2, \ldots, g_K] \tag{22}$$

Dla takich oznaczeń, dla sieci RBF można zapisać następujące równanie:

$$d = Gw + e \tag{23}$$

gdzie e jest błędem niedopasowania sieci. Ortogonalizacja macierzy G pozwala na określenie osobno wpływu każdej funkcji g_i na wartość części pożądanej energii zdefiniowanej jako $(Gw)^2$ [8]. Dzięki temu możliwa jest selekcja najbardziej istotnych funkcji bazowych oraz znaczne

ograniczenie ilości funkcji tworzących sieć. Sama ortogonalizacja może zostać dokonana różnymi metodami z czego często stosowaną jest metoda Grama-Schmidta, gdzie w procesie ortogonalizacji macierzy G powstaje macierz ortogonalna Q oraz macierz górnotrójkątna A.

$$G = QA \tag{24}$$

$$A = \begin{bmatrix} 1 & a12 & a13 & \dots & a_{1K} \\ 0 & 1 & a23 & \dots & a_{2K} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{25}$$

Po procesie ortogonalizacji oraz wprowadzając nowe oznaczenie $b = Aw$ można zapisać nową postać równania 23:

$$d = QAw + e = Qb + e \tag{26}$$

Rozwiązując równanie 26 metodą najmniejszych kwadratów otrzymujemy:

$$b = (QQ^T)^{-1}Q^T d \tag{27}$$

Mając dane b oraz A jesteśmy w stanie określić wektor wag $w = A^{-1}b$. Jak wspomniano wcześniej wartość pożądana energii jest określona jako $(Gw)^2$. Na tej podstawie można zdefiniować wzór określający udział danej funkcji w ogólnym bilansie energii jako:

$$\epsilon_i = \frac{b_i^2 q_i^T q_i}{d^T d} \tag{28}$$

Warunek stopu metody może zostać określony m. in. poprzez doprowadzenie do sytuacji, gdzie wartość sumaryczna energii wnoszonej przez wszystkie funkcje będzie bliska jedności, czyli:

$$1 - \sum_{i=1}^{N} \epsilon_i < \delta \tag{29}$$

gdzie δ jest z góry ustaloną wielkością warunkując koniec algorytmu.

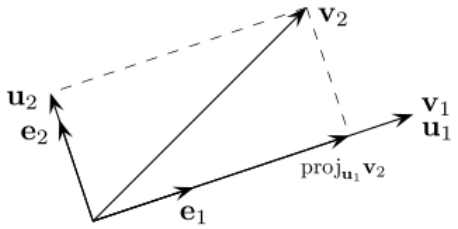
Jak wspomniano wcześniej ortogonalizacja może zostać wykonana różnymi metodami, jednak omówiona zostanie metoda ortogonalizacji Grama-Schmidta ze względu na jej popularność. Stosując algorytm ortogonalizacji Grama-Schmidta można dokonać rozkładu QR macierzy A, gdzie macierz Q jest macierzą ortogonalną zaś macierz R jest macierzą górnotrójkątną z jedynkami na diagonali. Algorytm takiego rozkładu można przedstawić w sposób

następujący[2]:

$$\begin{aligned} q_1 &= a_1 \\ q_k &= a_k - \sum_{j=1}^{k-1} r_{jk} q_j = a_k - \sum_{j=1}^{k-1} \frac{<a_k, q_j>}{<q_j, q_j>} q_j, \quad 2 \leqslant l \leqslant n \end{aligned} \tag{30}$$

$\frac{<a_k,q_j>}{<q_j,q_j>} q_j$ jest rzutowaniem ortogonalnym wektora a_k na wektor q_j.

Graficzna reprezentacja procesu ortogonalizacji dla dwóch pierwszych kroków przedstawiona jest na rysunku 4. Mając wybrany wektor u_1 (literą e oznaczono znormalizowany wektor u) należy dokonać projekcji (rzutu ortogonalnego) kolejnego wektora na przestrzeń rozpiętą przez wszystkie ortogonalne wektory (w tym przypadku tylko wektor u_1). Różnica wektora ortogonalizowanego oraz projekcji wektora ortogonalizowanego ($proj_{u_1} v_2$)na przestrzeń rozpiętą przez wektory ortogonalne pozwala uzyskać nam kolejny wektor ortogonalny (u_2).



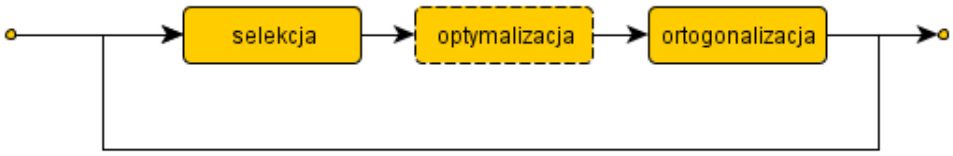
Rysunek 4: Proces ortogonalizacji dla dwóch pierwszych kroków metody Grama-Schmidta
źródło: http://commons.wikimedia.org/wiki/File:Gram%E2%80%93Schmidt_process.svg

Funkcje wybiera się spośród wcześniej wygenerowanej biblioteki funkcji. Metoda ta zapewnia wybór najbardziej optymalnego zbioru funkcji z biblioteki. Jednak aby wybór ten był optymalny konieczne jest wygenerowanie odpowiedniej biblioteki, która może zawierać znaczną ilość funkcji bazowych przez co proces uczenia się sieci może stać się nieefektywny. Rozwiązaniem w tym przypadku jest zastosowanie metody GOFR

4.2 Selekcja oparta o uogólnioną regresję postępującą z ortogonalizacją (GOFR)

Metoda GOFR oparta jest na metodzie OLS[9]. Modyfikację w stosunku do metody OLS stanowi optymalizacja dokonywana na etapie selekcji każdej funkcji bazowej. Optymalizacja ta polega na minimalizacji błędu sieci na zbiorze uczącym (np. zdefiniowanego jako MSE), poprzez odpowiedni dobór parametrów każdej wybranej funkcji. Optymalizacja taka może zostać przeprowadzona np. jedną z metod gradientowych.

Porównanie obu metod przedstawia schemat z rysunku 5. Linią przerywaną zaznaczony jest krok specyficzny dla metody GOFR. Optymalizacja pozwala na zmniejszenie ilości funkcji tworzących bibliotekę. Dzieje się tak dzięki temu, że po wyborze odpowiedniej funkcji algorytm zmodyfikuje parametry wybranej funkcji tak aby zminimalizować błąd. W przypadku algorytmu opartego o OLS parametry funkcji były stałe, a ilość funkcji tworzących bibliotekę musiała być duża w celu zapewnienia odpowiednio wysokiego prawdopodobieństwa wyboru optymalnych funkcji bazowych.



Rysunek 5: Schemat algorytmu OFR oraz GOFR

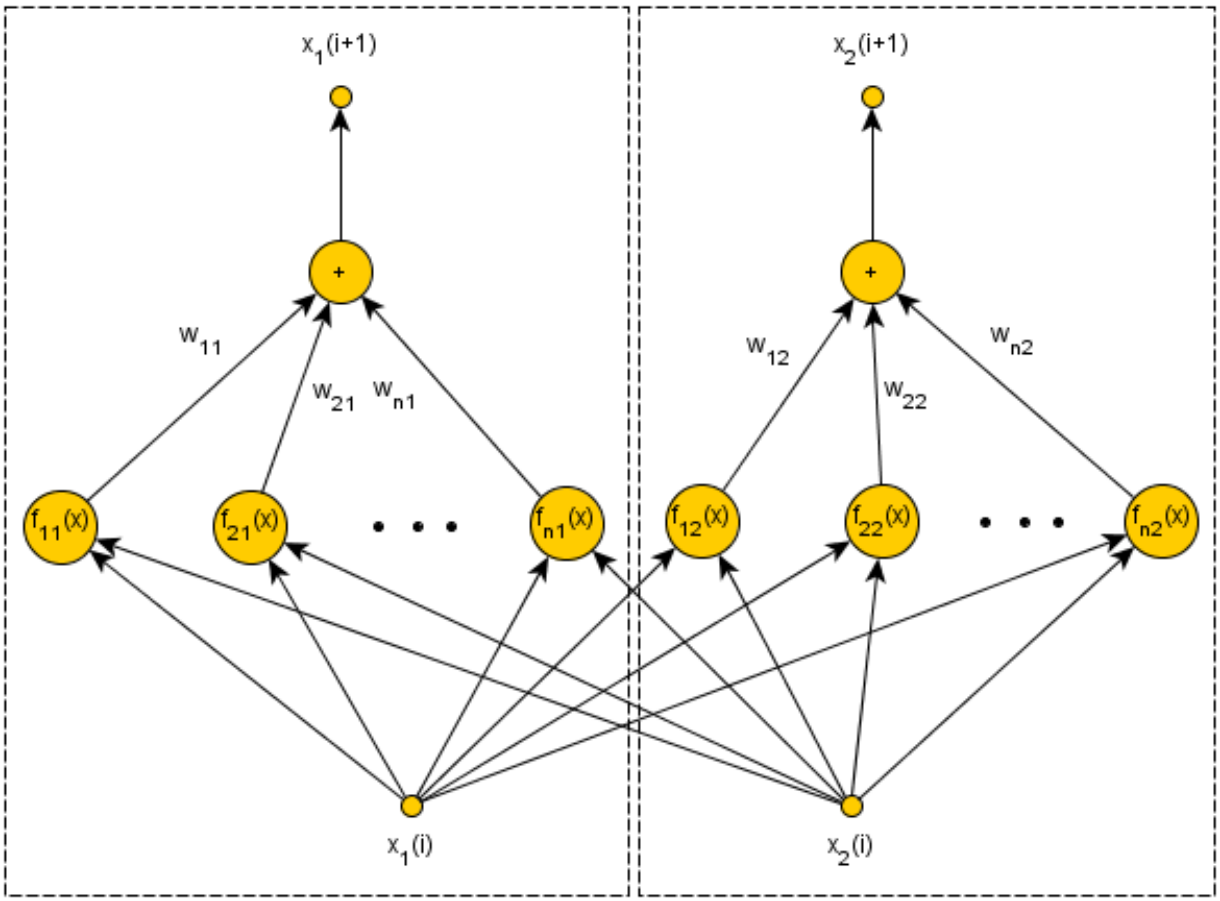
5 Identyfikacja systemu opisanego równaniem Van der Pola

Równanie Van der Pola to równanie różniczkowe wyrażone wzorem:

$$y''(t) - \mu(1 - y^2(t))y'(t) + y(t) = 0 \tag{31}$$

Równanie Van der Pola aktualnie jest często brane pod uwagę jako podstawowy model procesu oscylacji w fizyce, elektronice, biologii, neurologii, socjologii i ekonomii[17].

Identyfikację systemu opisanego równaniem 31 (dla $\mu = 1$) oparto o sieć RBF i metodę GOFR selekcji funkcji bazowych. Strukturę zaproponowanej sieci przedstawia rysunek 6. Wejście sieci stanowią $x_1 = y(t)$ oraz $x_2 = y'(t)$, czyli aktualny stan obiektu. Wyjściem sieci



Rysunek 6: Schemat zaprojektowanej sieci RBF

jest stan obiektu w kolejnym kroku. Jak można zauważyć sieć została podzielona na dwie do pewnego stopnia niezależne części - każde z wejść posiada oddzielny zestaw funkcji RBF.

W związku z tym każda z części sieci była uczona oddzielnie (dla tych samych danych uczących).

Zbiór danych uczących sieć neuronową wygenerowano rozwiązując równanie metodą Runge-Kutty IV rzędu dla warunków początkowych $[y(0) = 0, y'(0) = 2]$, kroku metody $h = 0.1$, w przedziale $t \in [0, 40]$. Równanie różniczkowe 31 drugiego rzędu zapisano w postaci układu dwóch równań różniczkowych pierwszego rzędu:

$$\begin{aligned} y'(t) &= y_1 \\ y_1'(t) &= (1 - y_1^2)y - y_1 \end{aligned} \tag{32}$$

Powstała w wyniku rozwiązania równania Van der Pola trajektoria, wykres funkcji $y(t)$ oraz jej pierwszej pochodnej $y'(t)$ przedstawione są odpowiednio na rysunkach 7a, 7b oraz 7c. Dane powstałe w wyniku rozwiązania równania zostały użyte jako zbiór danych uczących.

Kolejnym etapem było stworzenie odpowiedniej biblioteki funkcji RBF. W tym celu najpierw wygenerowano bibliotekę 115 funkcji RBF postaci:

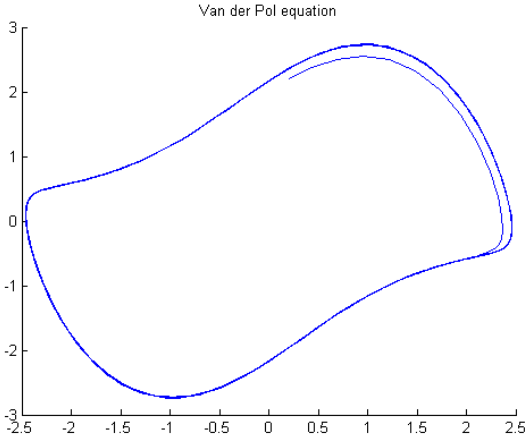
$$f(x_1(t), x_2(t), c_1, c_2, \sigma) = \exp(-\frac{(x_1(t) - c_1)^2 + (x_2(t) - c_2)^2}{2\sigma^2}) \tag{33}$$

gdzie:
$x_1(t) = y(t)$
$x_2(t) = y'(t)$

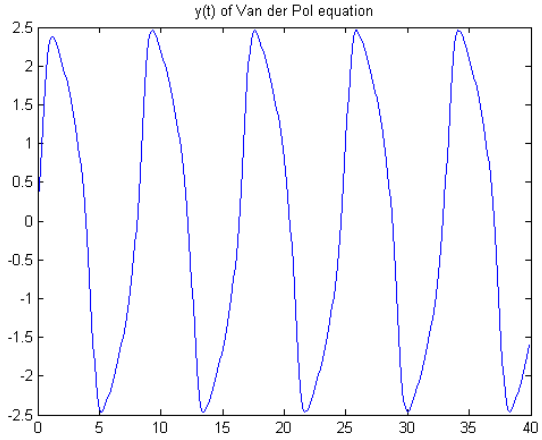
Generowanie biblioteki odbywało się poprzez wybór równomiernie rozłożonych w przestrzeni dwuwymiarowej $D = [min(x_1), max(x_1)] \times [min(x_2), max(x_2)]$ centrów. Dla każdego kolejnego poziomu generowania biblioteki funkcji, nowe centra c_1 oraz c_2 były wybierane poprzez dwukrotnie zagęszczenie punktów w każdej z osi oraz zmniejszanie szerokości funkcji gaussa σ o połowę. Centra dla tak wygenerowanych funkcji przedstawia rysunek 8. Ilość poziomów ograniczono do trzech, generując w ten sposób 115 funkcji RBF.

Następnie dokonano selekcji najbardziej istotnych funkcji z biblioteki wykorzystując metodę GOFR. Po selekcji każdej z funkcji dokonywana jest optymalizacja wybranej funkcji metodą Levenberga-Marquardta. Sieć neuronowa o funkcjach bazowych zdefiniowanych wzorem 33 realizuje funkcję:

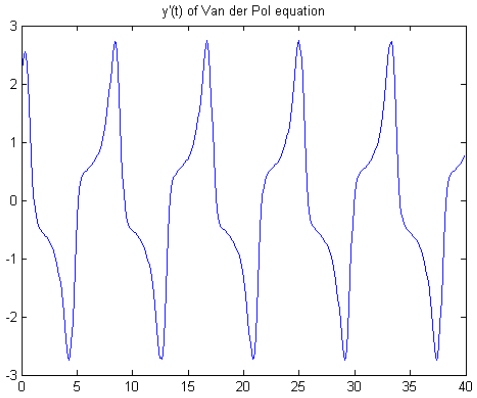
$$\tilde{y} = \sum_{i=1}^{N} w_i f_i(x_1, x_2, c_1, c_2, \sigma_i) \tag{34}$$



(a) Trajektoria

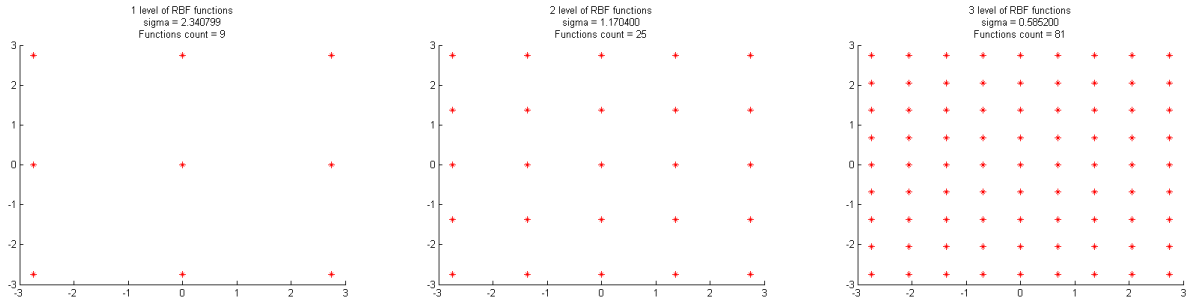


(b) Wykres funkcji $y(t)$



(c) Wykres pierwszej pochodnej $y'(t)$

Rysunek 7: Wykresy powstałe w wyniku rozwiązania równania różniczkowego Van der Pola dla warunków początkowych $[y(0) = 0, y'(0) = 2]$



Rysunek 8: Centra wygenerowanych funkcji RBF

Optymalizowano następujące parametry sieci RBF: c_1, c_2, σ oraz w. W celu skorzystania z metody Leveberga-Marquardta konieczne jest określenie gradientu funkcji błędu $e = y - \tilde{y}$ oraz jej pochodnych cząstkowych. Funkcja błędu e jest wyrażona wzorem:

$$e = y - \tilde{y} = y - wf(x_1, x_2, c_1, c_2, \sigma) = y - w\exp(-\frac{(x_1 - c_1)^2 + (x_2 - c_2)^2}{2\sigma^2}) \tag{35}$$

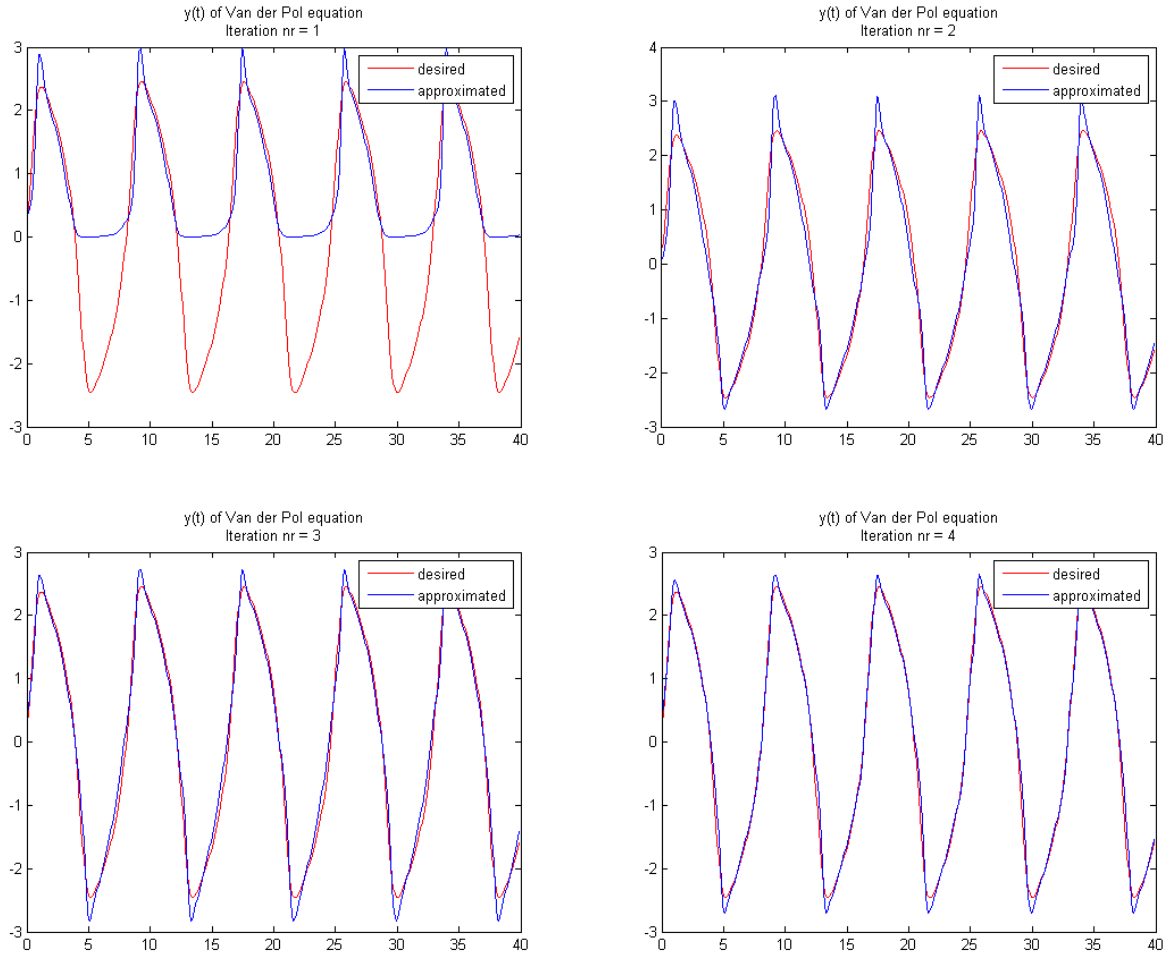
Gradient e wynosi zatem:

$$\begin{bmatrix} \frac{\partial e}{\partial w} \\ \frac{\partial e}{\partial \sigma} \\ \frac{\partial e}{\partial c_1} \\ \frac{\partial e}{\partial c_2} \end{bmatrix} = \begin{bmatrix} f(\cdot) \\ wf(\cdot)\frac{(x_1-c_1)^2+(x_2-c_2)^2}{\sigma^3} \\ wf(\cdot)\frac{(x_1-c_1)}{\sigma^2} \\ wf(\cdot)\frac{(x_2-c_2)}{\sigma^2} \end{bmatrix} \tag{36}$$

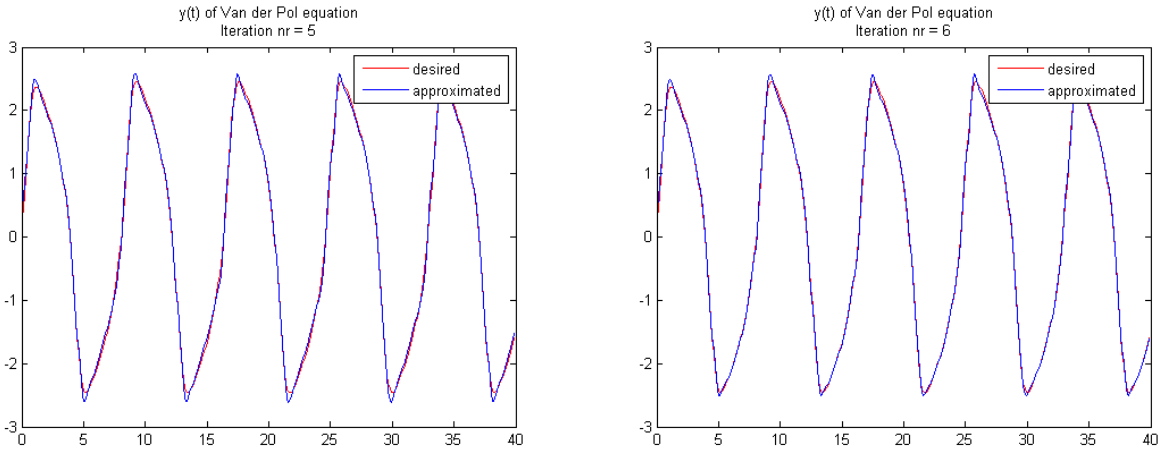
W przypadku zwiększania się błędu na zbiorze uczącym parametr λ ze wzoru 19 był zwiększany 10-krotnie, w przypadku zmniejszania się błędu parametr λ był zmniejszany 10-krotnie. Warunek stopu został ustalony tak aby błąd średniokwadratowy dla każdego z wyjść sieci był mniejszy od 1e-6.

6 Wyniki identyfikacji obiektu opisanego równaniem Van der Pola

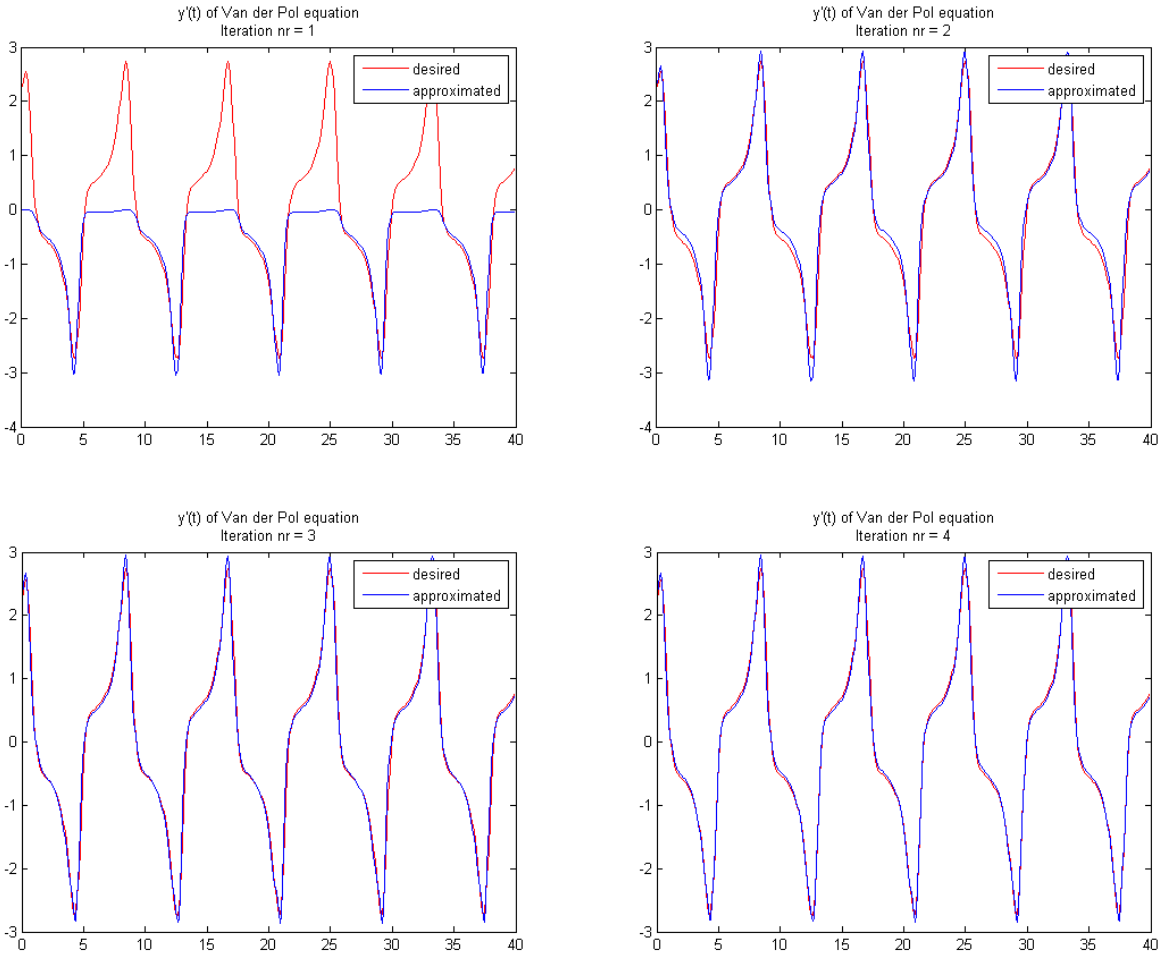
Osiągnięcie błędu średniokwadratowego poniżej 1e-6 udało się osiągnąć dla 24 funkcji bazowych dla $y(t)$ (błąd równy = 6.8925e-7) oraz 25 funkcji bazowych dla $y'(t)$ (błąd równy = 7.2683e-7). Na rysunkach 9, 10 oraz rysunkach 11, 12 zostały przedstawione odpowiednio: aproksymacja funkcji $y(t)$ oraz aproksymacja pierwszej pochodnej $y'(t)$ równania Van der Pola dla pierwszych sześciu iteracji. Jak można zauważyć kształt funkcji $y(t)$ oraz pochodnej $y'(t)$ jest dokładnie odwzorowany już po selekcji kilku pierwszych funkcji. Dzieje się tak ze względu na to, że algorytm selekcji funkcji bazowych GOFR wybiera funkcje wnoszące największy wkład energii w kolejnych iteracjach. Kolejne wybierane funkcje mają coraz mniejsze znaczenie w odwzorowywaniu obiektu opisanego równaniem Van der Pola.



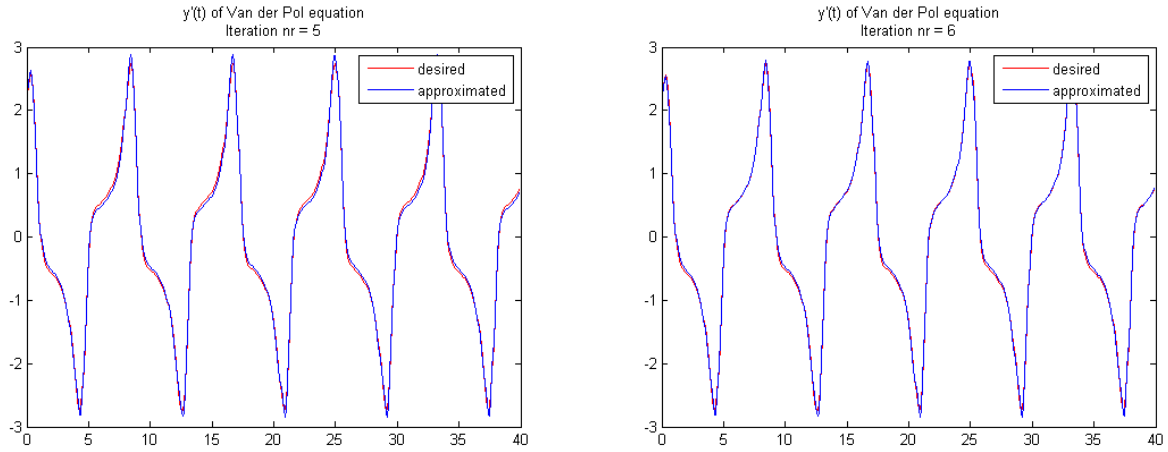
Rysunek 9: Aproksymacja funkcji y(t) po selekcji funkcji numer 1-4



Rysunek 10: Aproksymacja funkcji y(t) po selekcji funkcji numer 5-6

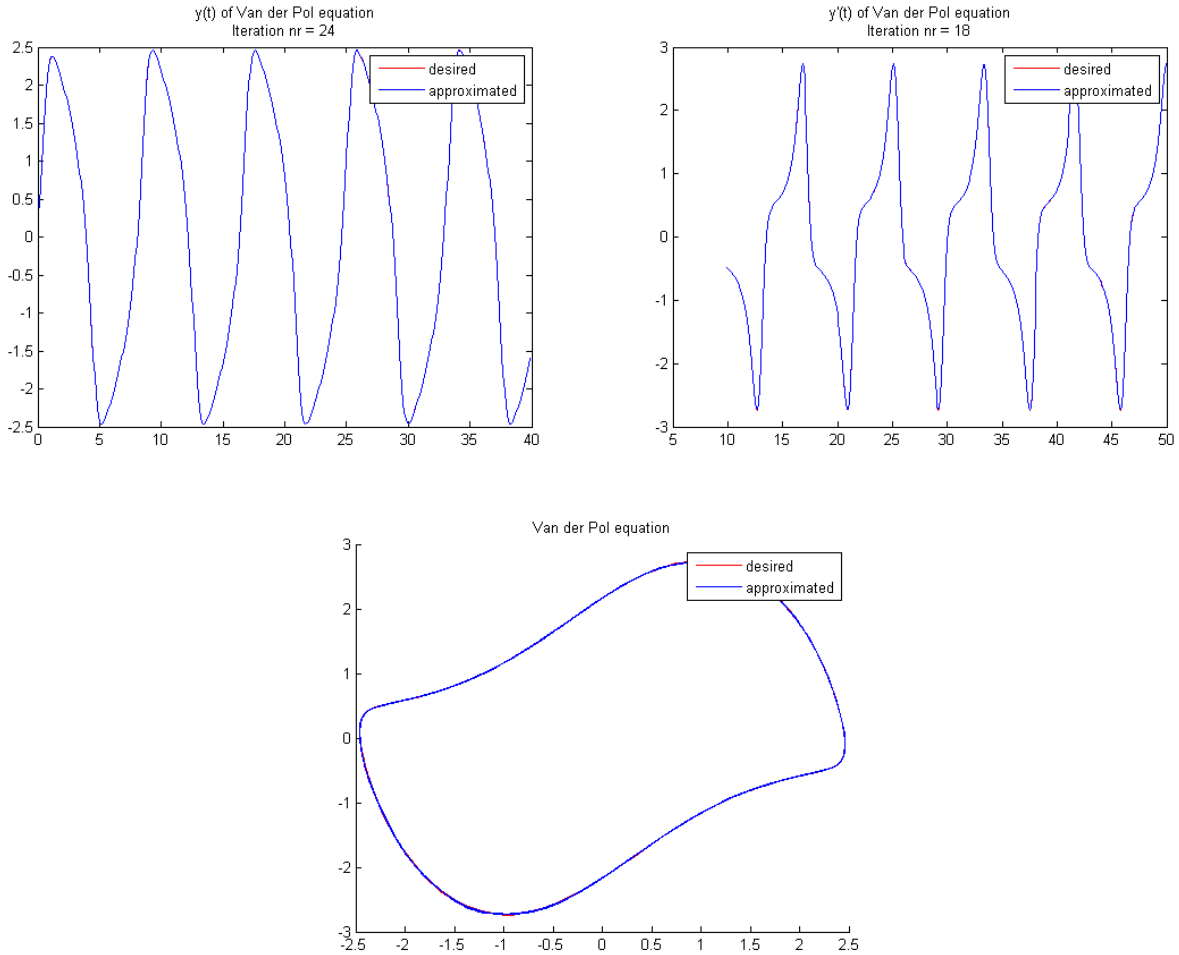


Rysunek 11: Aproksymacja funkcji y'(t) po selekcji funkcji numer 1-4



Rysunek 12: Aproksymacja funkcji y'(t) po selekcji funkcji numer 5-6

Rysunek 13 przedstawia aproksymację równania Van der Pola przez wytrenowaną sieć.



Rysunek 13: Sygnał zadany oraz aproksymowany przez wytrenowaną sieć RBF

W tabeli 1 zostały zebrane dane na temat wybranych funkcji RBF oraz błąd średniokwadratowy aproksymacji funkcji $y(t)$ równania Van der Pola po selekcji kolejnych funkcji RBF.

number	indeks RBF	centrum 1	centrum 2	szerokość σ	energia	błąd MSE
1	32	2.9948	0.7222	1.5295	4.7410e-1	1.5872e+0
2	2	-5.8209	-1.0474	2.6805	4.9905e-1	8.1038e-2
3	8	7.0671	1.4665	3.8568	1.5418e-2	3.4504e-2
4	52	-2.2270	3.0342	0.7155	5.3306e-3	1.8416e-2
5	15	-1.0761	-3.1178	1.0496	4.5295e-3	4.7450e-3
6	60	-1.3884	2.1498	0.5889	7.8749e-4	2.3682e-3
7	103	2.0854	0.7488	0.6221	4.1060e-4	1.1290e-3
8	47	-1.9902	-0.6401	0.5865	1.0880e-4	8.0058e-4
9	106	1.9631	3.4429	0.4688	1.1553e-4	4.5188e-4
10	20	0.6923	-3.9854	1.3350	4.7597e-5	3.0823e-4
11	68	-0.4170	1.0838	0.6116	3.8596e-5	1.9174e-4
12	109	2.7940	-1.5663	0.5293	1.5792e-5	1.4408e-4
13	62	-0.7083	-2.5993	0.5065	8.2334e-6	1.1923e-4
14	79	0.5131	2.2032	0.2673	1.8315e-5	6.3952e-5
15	5	-0.0016	0.0540	2.3580	4.6821e-6	4.9821e-5
16	4	0.0067	-2.7505	2.3463	5.5652e-6	3.3024e-5
17	92	1.4041	-0.7927	0.5950	4.2814e-6	2.0102e-5
18	6	-0.0072	2.7806	2.3611	1.4043e-6	1.5864e-5
19	14	-2.7163	2.7149	1.1521	1.0246e-6	1.2771e-5
20	102	1.9456	0.0229	0.5151	2.3885e-6	5.5625e-6
21	94	1.3554	0.6871	0.5818	4.3525e-7	4.2489e-6
22	115	2.9594	2.8968	0.5772	5.1100e-7	2.7066e-6
23	28	1.3478	1.3509	1.1561	4.2283e-7	1.4304e-6
24	35	-2.7401	-2.7393	0.5862	2.4558e-7	6.8925e-7

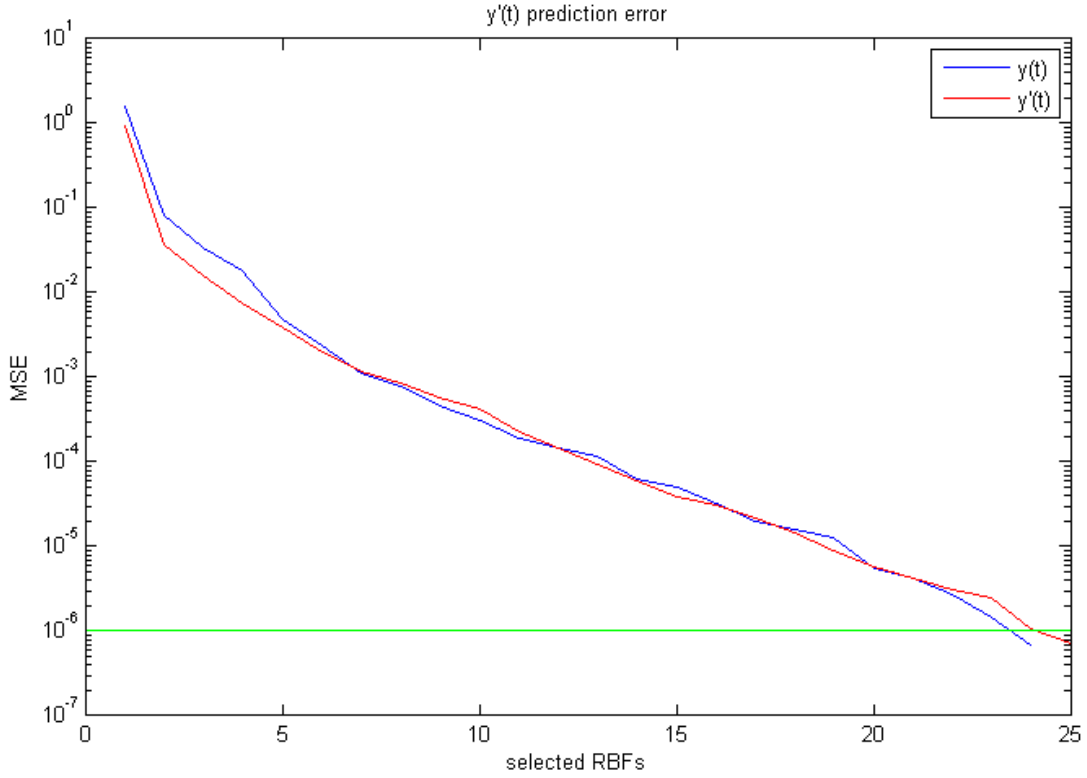
Tabela 1: Wybrane funkcje RBF dla y(t)

W tabeli 2 zostały zebrane dane na temat wybranych funkcji RBF oraz błąd średniokwadratowy aproksymacji pochodnej $y'(t)$ równania Van der Pola po selekcji kolejnych funkcji RBF.

numer	indeks RBF	centrum 1	centrum 2	szerokość σ	energia	błąd MSE
1	20	1.1446	-5.7343	2.0595	4.9501e-1	9.4653e-1
2	24	-0.5846	5.1218	2.1860	4.8488e-1	3.7690e-2
3	71	0.3756	-3.5358	0.8221	1.1585e-2	1.5975e-2
4	55	-1.3838	-1.3455	0.5934	4.5029e-3	7.5351e-3
5	105	2.6435	2.2556	0.6908	1.9649e-3	3.8522e-3
6	18	-1.7286	1.9662	0.9173	9.8872e-4	1.9989e-3
7	31	2.7142	-1.5228	1.1111	4.3126e-4	1.1906e-3
8	35	-2.4433	-2.8083	0.4078	1.8653e-4	8.4097e-4
9	34	2.4846	2.5536	1.0744	1.3652e-4	5.8508e-4
10	4	-0.0373	-2.1985	2.0320	8.6977e-5	4.2205e-4
11	51	-2.0368	2.1029	0.5648	1.0665e-4	2.2215e-4
12	44	-2.3244	-3.0361	0.5831	4.1302e-5	1.4473e-4
13	87	0.6683	2.1058	0.5772	2.7287e-5	9.3585e-5
14	108	2.5990	-1.9039	0.5283	1.8039e-5	5.9773e-5
15	5	-0.0008	0.0006	2.3321	1.0989e-5	3.9175e-5
16	66	-0.8763	0.2859	0.5596	4.5942e-6	3.0564e-5
17	56	-1.2921	-0.5707	0.4300	4.8491e-6	2.1475e-5
18	112	2.9438	0.5975	0.4572	3.6241e-6	1.4682e-5
19	54	-1.3559	-2.0394	0.5777	2.9742e-6	9.1075e-6
20	97	1.6974	3.4300	0.7467	1.8032e-6	5.7277e-6
21	86	0.7040	1.3515	0.5634	8.1315e-7	4.2035e-6
22	63	-0.6910	-2.0473	0.5768	6.0731e-7	3.0652e-6
23	39	-3.3163	0.0390	0.6737	3.4036e-7	2.4273e-6
24	111	2.3400	0.0310	0.4559	7.2918e-7	1.0605e-6
25	53	-1.3701	-2.7398	0.5851	1.7802e-7	7.2683e-7

Tabela 2: Wybrane funkcje RBF dla y'(t)

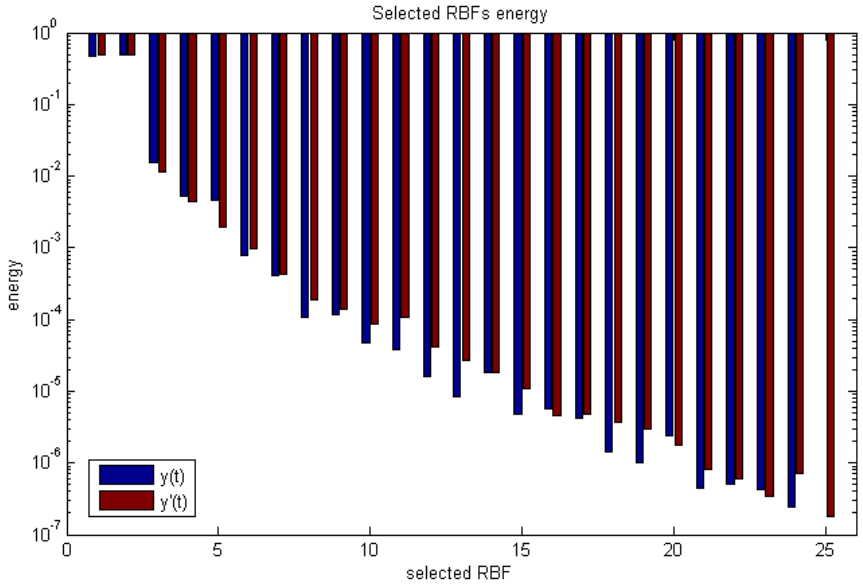
Błąd średniokwadratowy w funkcji wybranych funkcji RBF przedstawia rysunek 14 natomiast rysunek 15 przedstawia energię jaką wnosi każda kolejna wybrana funkcja. Jak można zauważyć główny wpływ na wartość błędu oraz energii mają dwie pierwsze wybrane funkcje. Wraz z wyborem kolejnych funkcji, błąd oraz energia wnoszona przez wybraną funkcję spadają w przybliżeniu logarytmicznie. Zieloną linią na wykresie 14 została oznaczona wartość MSE, która jest warunkiem stopu uczenia sieci.



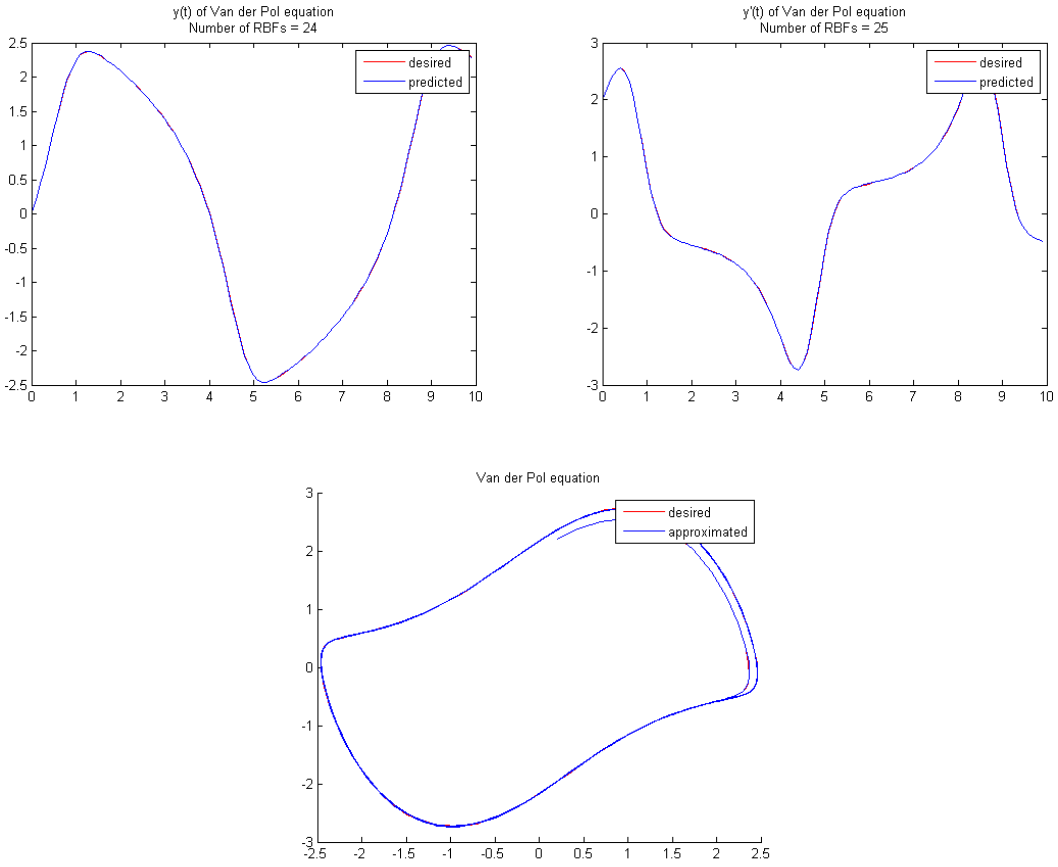
Rysunek 14: Błąd MSE w funkcji wybranych funkcji RBF

Po etapie trenowania sieci RBF sprawdzono zdolność tej sieci do predykcji. Predykcja na 100 kroków do przodu została przedstawiona na rysunku 16. Jak można zauważyć sieć dobrze modeluje obiekt opisany równaniem Van der Pola. Błąd średniokwadratowy dla wyznaczania $y(t)$ wyniósł 5.6108e-5, natomiast błąd predykcji wyznaczania $y'(t)$ wyniósł 9.4761e-5.

Trudno jest jednak dokładniej określić jakość predykcji na podstawie rysunku 16 ze względu na nieznaczne różnice w wartości zadanej oraz wartości przewidywanej przez sieć w wyniku czego sygnał zadany oraz przewidywany pokrywają się. Aby można było dokładnie określić odstępstwo wartości przewidywanej od wartości rzeczywistej na rysunku 17 został przedsta-

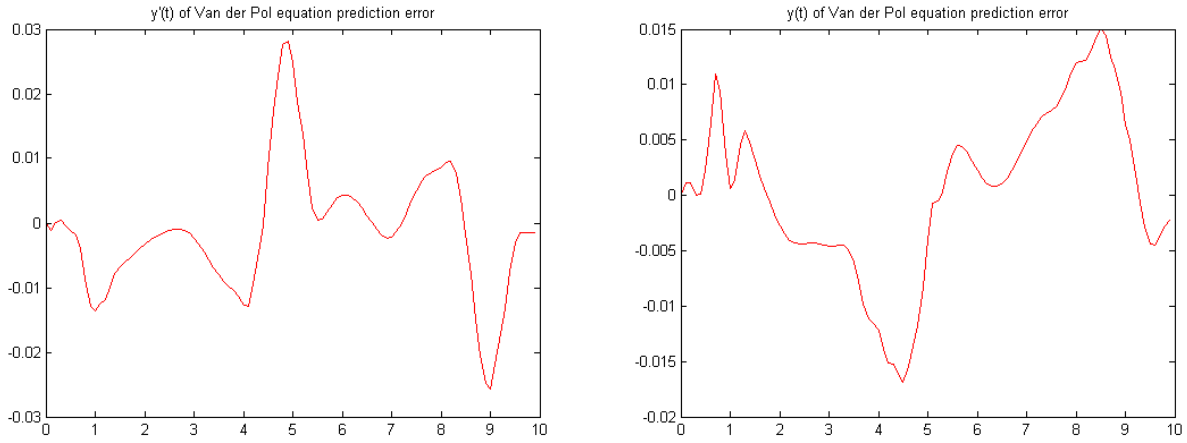


Rysunek 15: Energia wnoszona przez kolejne wybrane funkcji RBF



Rysunek 16: Sygnał zadany oraz przewidywany przez wytrenowaną sieć RBF $t \in [0, 10]$

wiony błąd predykcji w kolejnych krokach. Wartość bezwzględna błędu wyznaczenia $y(t)$ nie przekracza wartości 0.02. Maksymalny błąd wyznaczenia $y'(t)$ jest nieznacznie większy i nie przekracza wartości 0.03. Na tej podstawie można stwierdzić, że dla 100 kroków w przód sieć spełnia swoje zadanie.

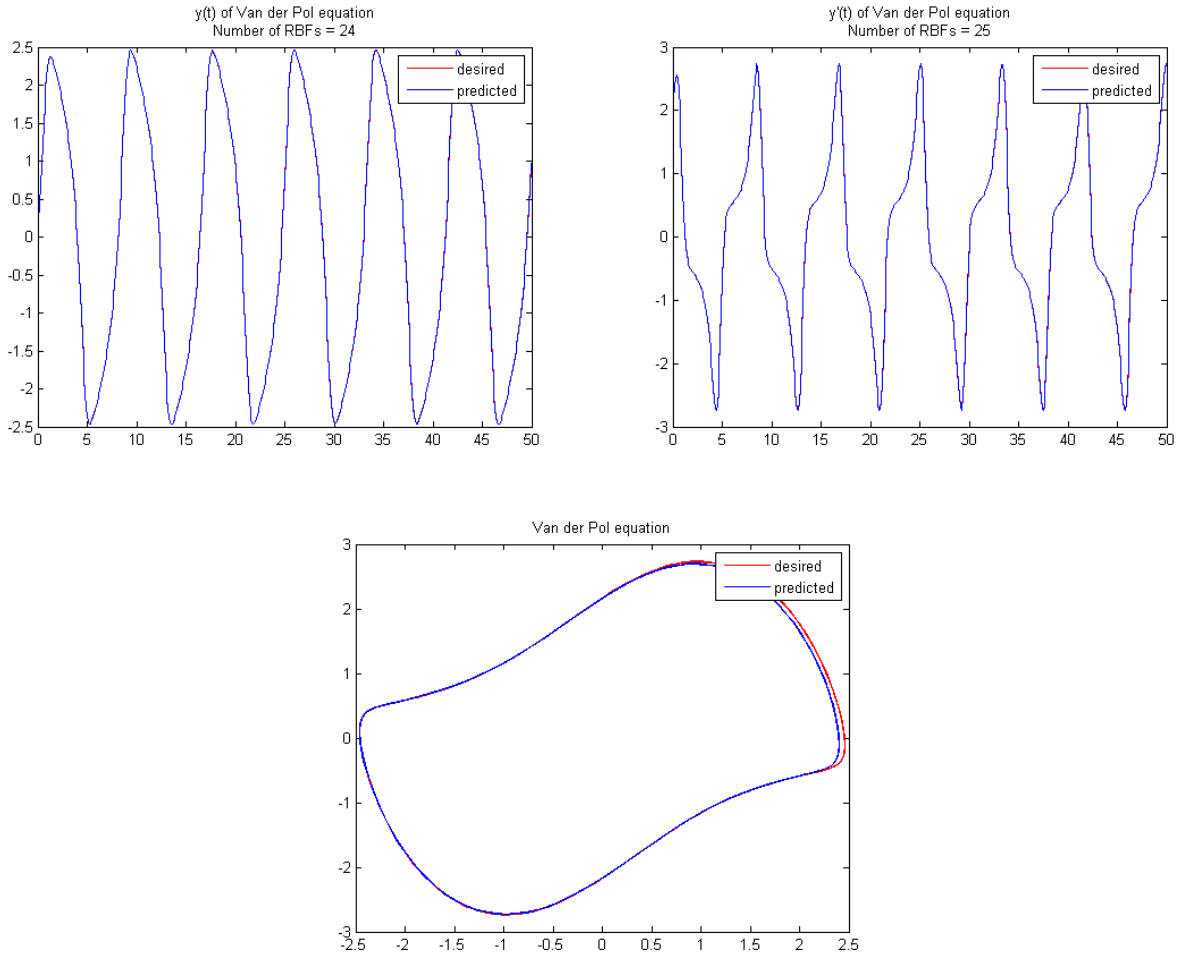


Rysunek 17: Błąd predykcji w funkcji czasu $t \in [0, 10]$

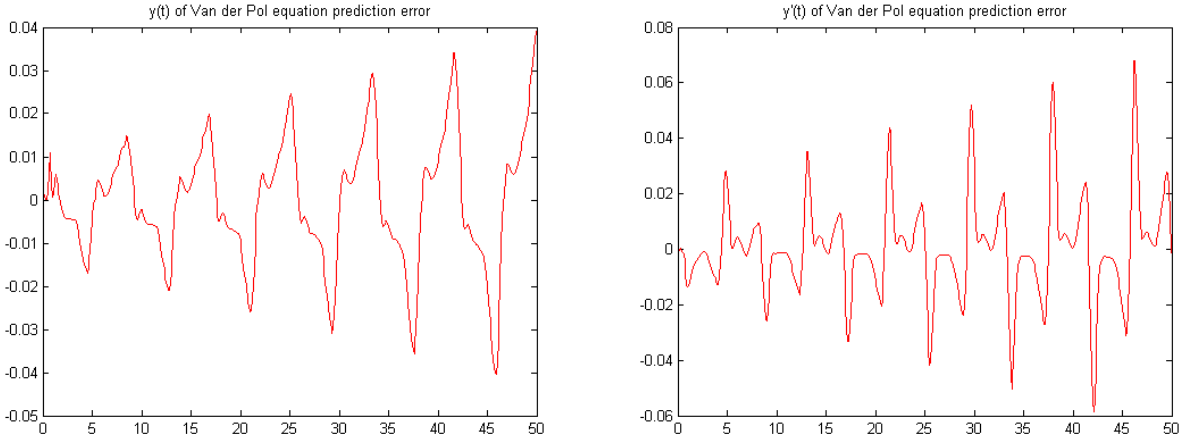
Następnie zdecydowano się na sprawdzenie jakości predykcji w 5-krotnie dłuższym przedziale czasu $t \in [0, 50]$. Na rysunku 18 został przedstawiany sygnał rzeczywisty oraz przewidywany. Jak można zauważyć jakość predykcji jest nadal dobra, ale podobnie jak wcześniej trudno jest określić dokładnie błąd między wartością przewidywaną a rzeczywistą. Nieznaczną rozbieżność można zauważyć na wykresie trajektorii. Błąd średniokwadratowy dla wyznaczania $y(t)$ wyniósł 1.9873e-4, natomiast błąd wyznaczania $y'(t)$ wyniósł 3.0130e-4, czyli kilkukrotnie wzrósł w stosunku do błędu dla 5-kronie krótszego okresu predykcji.

Aby dokładniej określić jakość predykcji w przedziale czasu $t \in [0, 50]$ ponownie skorzystano z wykresu błędu predykcji dla kolejnych kroków. Jak można zauważyć wraz z oddalaniem się od punktu początkowego błąd predykcji rośnie. Dzieje się tak ze względu na akumulację błędu. Należy zaznaczyć jednak, że odchylenie od wartości zadanej dla 500 kroków dla $y(t)$ nie przekracza wartości 0.04, natomiast dla $y'(t)$ nie przekracza wartości 0.07 co pozwala stwierdzić, że nawet dla tak długiego okresu czasu sieć spełnia swoje zadanie.

Do tej pory badana była jedynie predykcja dla tych samych warunków początkowych dla jakich sieć została wytrenowana. Jednak aby model był dobrze odwzorowany należałoby sprawdzić czy sieć jest zdolna do predykcji dla szerszego zakresu warunków początkowych. W tym

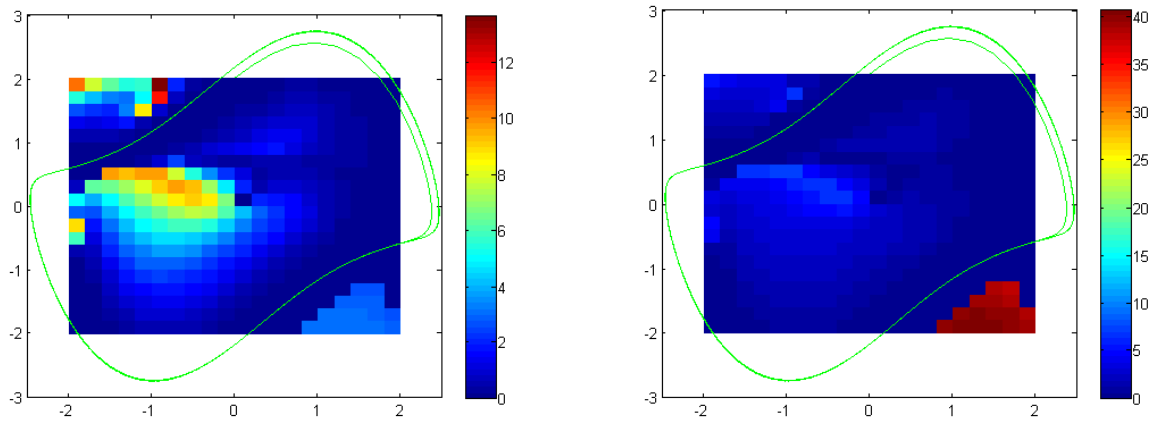


Rysunek 18: Sygnał zadany oraz przewidywany przez wytrenowaną sieć RBF $t \in [0, 50]$



Rysunek 19: Błąd predykcji w funkcji czasu $t \in [0, 50]$

celu została przeprowadzona predykcja dla 440 różnych wartości początkowych równomiernie rozłożonych w przestrzeni $[-2, 2] \times [-2, 2]$ dla 100 kroków w przód. Jakość predykcji została oceniona na podstawie błędu średniokwadratowego. Wykres obrazujący jakość predykcji w zależności od warunków początkowych przedstawia rysunek 20.

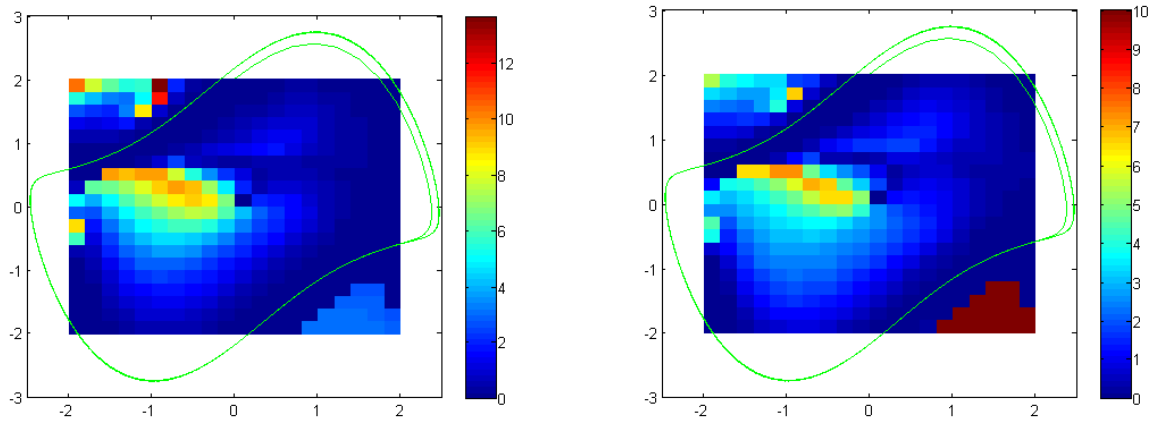


(a) y(t) (b) y'(t)

Rysunek 20: Błąd predykcji w zależności od warunków początkowych $t \in [0, 10]$

Jak można zauważyć sieć dobrze modeluje obiekt nie tylko dla wcześniej wybranego warunku początkowego, ale również dla warunków początkowych położonych w pobliżu trajektorii powstałej w wyniku rozwiązania równania Van der Pola z warunkiem początkowym jaki został wybrany w celu stworzenia zbioru uczącego. Rysunek 20 może być mylący ze względu na skalę barw jaka została użyta dla określenia błędu pochodnej - zakres błędu dla różnych warunków początkowych jest zdecydowanie większy dla $y'(t)$ niż dla $y(t)$ (maksymalny błąd MSE dla $y(t)$ wynosi 13.6417 natomiast dla $y'(t)$ wynosi 40.6341). Po analizie okazało się, że istnieje 16 (na 440) warunków początkowych, dla których błąd średniokwadratowy jest większy od 10. Z tego względu dla lepszego zobrazowania rozkładu błędu zastąpiono wszystkie 16 wartości błędu większe od 10 błędem = 10 i ponownie wykreślono rozkład błędu MSE $y'(t)$ w zależności od warunków początkowych. Przedstawiony jest on na rysunku 21.

Po odrzuceniu skrajnych wartości wykres rozkładu błędu $y'(t)$ znacznie bardziej przypomina wykres rozkładu błędu $y(t)$. Dla większości warunków początkowych sieć jednak nie jest w stanie dobrze modelować obiektu. Ustalając akceptowalność predykcji, dla której błąd MSE był mniejszy niż 0.01, sieć była w stanie poprawnie modelować obiekt dla 51 z 440 warun-



(a) y(t) (b) y'(t)

Rysunek 21: Błąd predykcji w zależności od warunków początkowych dla $t \in [0, 10]$ po odrzuceniu wartości skrajnych błedu MSE dla $y'(t)$

ków początkowych. Stworzony model zatem jest odpowiedni jedynie dla określonego zbioru warunków początkowych, co może być akceptowalne jeśli jesteśmy w stanie założyć, że modelowany obiekt będzie działał jedynie dla takich warunków początkowych. Aby sieć neuronowa była w stanie modelować równanie w szerszym zakresie warunków początkowym potrzebne są kolejne dane. W tym celu wygenerowano 80 trajektorii powstałych w wyniku rozwiązania równania Van der Pola dla warunków początkowych równomiernie rozłożonych w przestrzeni $[-2, 2] \times [-2, 2]$ w przedziale czasu $t \in [0, 40]$ z krokiem $h = 0.1$. Zdecydowano się na zmodyfikowanie warunku stopu do postaci MSE $<$ 1e-5 ze względu na znaczne zwiększenie się zbioru uczącego, a co za tym idzie znacznie wydłużenie uczenia się sieci. Osiągnięcie błędu średniokwadratowego poniżej 1e-5 udało się osiągnąć dla 25 funkcji bazowych dla funkcji $y(t)$ (błąd równy = 9.5037e-6) oraz 26 funkcji bazowych dla pochodnej $y'(t)$ (błąd równy = 7.9001e-6), czyli wykorzystując jedynie jedną funkcję bazową więcej dla każdej z części sieci. Należy jednak w tym miejscu przypomnieć, że warunkiem stopu było uzyskanie błędu średniokwadratowego 10-krotnie większego niż miało to miejsce w przypadku wcześniejszym.

W tabeli 3 zostały zebrane dane na temat wybranych funkcji RBF oraz błąd średniokwadratowy aproksymacji $y(t)$ po selekcji kolejnych funkcji RBF.

W tabeli 4 zostały zebrane dane na temat wybranych funkcji RBF oraz błąd średniokwadratowy aproksymacji $y'(t)$ po selekcji kolejnych funkcji RBF.

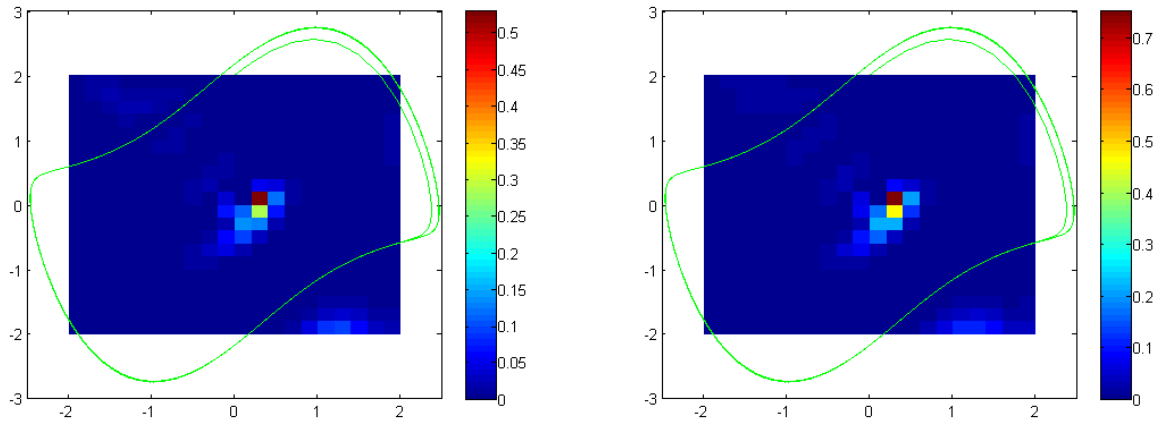
numer	indeks RBF	centrum 1	centrum 2	szerokość σ	energia	błąd MSE
1	12	-4.8953	-0.9105	2.0586	4.7050e-1	1.5515e+0
2	8	5.4193	0.9971	2.5798	5.0360e-1	7.5903e-2
3	2	-4.6034	-0.8540	3.0277	1.4313e-2	3.3965e-2
4	5	0.0222	-0.2106	3.2317	9.8758e-3	5.0279e-3
5	97	2.0527	3.4549	0.7361	1.0031e-3	2.0888e-3
6	77	0.1850	1.2900	0.6728	3.6159e-4	1.0293e-3
7	84	0.8268	-0.0171	0.6789	9.3702e-5	7.5471e-4
8	103	2.0429	0.6027	0.5178	8.7412e-5	4.9858e-4
9	53	-1.4953	-3.9012	0.6884	5.9932e-5	3.2298e-4
10	56	-1.5088	-0.7513	0.6456	4.0396e-5	2.0461e-4
11	60	-1.6142	2.5627	0.6911	2.5830e-5	1.2893e-4
12	108	3.2889	-1.8451	0.4369	8.0323e-6	1.0539e-4
13	57	-1.7669	0.1593	0.3909	6.2747e-6	8.7008e-5
14	69	-0.7720	2.5602	0.7003	4.4067e-6	7.4096e-5
15	11	-2.7698	-1.4594	1.1633	3.9520e-6	6.2516e-5
16	27	1.5564	0.1463	1.2159	3.2820e-6	5.2899e-5
17	30	3.2537	-3.1569	1.1834	3.6829e-6	4.2108e-5
18	88	0.7869	3.5360	0.4310	4.5237e-6	2.8853e-5
19	105	2.6917	2.6612	0.5780	1.8611e-6	2.3400e-5
20	40	-3.9490	1.0595	0.7024	1.0632e-6	2.0285e-5
21	109	3.3322	-1.6728	0.6157	6.1811e-7	1.8474e-5
22	38	-4.0090	-0.9720	0.7012	7.5296e-7	1.6267e-5
23	36	-3.8224	-2.8683	0.7438	1.0858e-6	1.3086e-5
24	65	-0.8844	-0.9351	0.6755	6.7549e-7	1.1107e-5
25	58	-1.5267	0.4323	0.5790	5.4706e-7	9.5037e-6

Tabela 3: Wybrane funkcje RBF dla y(t)

numer	indeks RBF	centrum 1	centrum 2	szerokość σ	energia	błąd MSE
1	24	-1.1250	5.6516	2.0361	4.7516e-1	9.8511e-1
2	20	0.5488	-4.9698	2.1518	5.0392e-1	3.9276e-2
3	78	-0.3925	3.5349	0.8580	1.2219e-2	1.6341e-2
4	104	1.8566	1.5269	0.6389	4.4165e-3	8.0511e-3
5	55	-2.6142	-2.2752	0.7069	1.9361e-3	4.4172e-3
6	26	1.6316	-1.7819	0.9866	1.0335e-3	2.4773e-3
7	13	-2.6461	0.9889	1.0006	3.2912e-4	1.8596e-3
8	71	0.4828	-3.7540	0.7487	2.0188e-4	1.4807e-3
9	86	0.7849	1.5639	0.7424	1.9636e-4	1.1121e-3
10	115	2.9828	3.1229	0.5308	8.5215e-5	9.5215e-4
11	67	-0.6655	0.4598	0.6018	1.2760e-4	7.1264e-4
12	112	2.5479	0.7001	0.3633	7.6301e-5	5.6943e-4
13	105	2.8729	2.8243	0.7434	7.0644e-5	4.3683e-4
14	16	-1.4468	-1.5020	1.1100	3.6943e-5	3.6749e-4
15	51	-2.5600	2.5033	0.6451	3.9223e-5	2.9387e-4
16	95	1.7201	1.7280	0.6390	2.2062e-5	2.5246e-4
17	109	2.9319	-1.0482	0.4542	1.8539e-5	2.1767e-4
18	44	-2.0737	-2.9877	0.4729	1.4993e-5	1.8953e-4
19	6	-0.0106	3.5027	2.6365	1.3682e-5	1.6385e-4
20	5	0.0097	0.0031	2.6574	2.0389e-5	1.2558e-4
21	32	3.3282	-0.0031	1.2907	2.1197e-5	8.5791e-5
22	25	2.0450	-3.8439	1.3209	1.6687e-5	5.4470e-5
23	50	-2.7665	1.9171	0.7318	1.5010e-5	2.6296e-5
24	97	2.0519	4.1067	0.7983	4.8446e-6	1.7203e-5
25	63	-0.8497	-2.6336	0.6773	3.5100e-6	1.0615e-5
26	83	0.7341	-0.7148	0.6868	1.4466e-6	7.9001e-6

Tabela 4: Wybrane funkcje RBF dla y'(t)

Następnie dokonano sprawdzenia czy sieć po zwiększeniu zbioru uczącego jest w stanie dobrze modelować układ dla różnych warunków początkowych. Wykres obrazujący jakość predykcji w zależności od warunków początkowych przedstawia rysunek 22.



(a) y(t)

(b) y'(t)

Rysunek 22: Błąd predykcji w zależności od warunków początkowych $t \in [0, 10]$

Jak można zauważyć dla większości warunków początkowych sieć dobrze modeluje układ. Największa wartość błędu wynosiła odpowiednio 0.5292 dla $y(t)$ oraz 0.7508 dla $y'(t)$. W porównaniu do wartości maksymalnej wartości błędu dla sieci uczonej na podstawie danych z jednej trajektorii (maksymalny błąd MSE dla $y(t)$ wynosił 13.6417 natomiast dla $y'(t)$ wynosił 40.6341) błąd średniokwadratowy zmalał kilkadziesiąt razy. Znacznie wzrosła też ilość warunków początkowych, dla których błąd MSE predykcji był mniejszy niż 0.01. Wartość ta wzrosła z 51 do 348 warunków początkowych.

7 Podsumowanie oraz wnioski

Celem pracy była próba identyfikacji nieliniowych obiektów dynamicznych z wykorzystaniem metody GOFR. Cel udało się zrealizować identyfikując obiekt dynamiczny opisany równaniem Van der Pola. Posiadając dane tylko z jednej trajektorii sieć była w stanie dobrze modelować obiekt dla różnych warunków początkowych. Predykcja na 500 kroków w przód skutkowała błędem średniokwadratowym w wyznaczeniu $y(t)$ równym 1.9873e-4 oraz błędem wyznaczania $y'(t)$ równym 3.0130e-4. Następnie aby zwiększyć jakość odwzorowania modelu obiektu dla szerszego zakresu punktów początkowych zwiększono zbiór danych uczących do 80 trajektorii. Po ponownym wytrenowaniu sieci okazało się, że jakość odwzorowania modelu dla szerokiego zakresu warunków początkowych uległa znacznemu polepszeniu. Określając jako akceptowalną predykcje taką, dla której błąd MSE < 0.01 dla $y(t)$ oraz $y'(t)$ jednocześnie, udało się uzyskać poprawną predykcję dla 348 na 440 warunków początkowych.

Czas uczenia sieci uległ jednak znacznemu wydłużeniu, a ilość funkcji bazowych uległa zwiększeniu pomimo ustalania jako warunku stopu uzyskania błędu MSE dziesięciokrotnie mniejszego. Pomimo zwiększenia zbioru uczącego dla części warunków początkowych sieć nadal nie najlepiej odwzorowywała obiekt dynamiczny. Wartości początkowe dla których błąd był większy od gwarantującego poprawną predykcję były zgrupowane. Dalszą poprawę wyników mogłoby przynieść zwiększenie zbioru uczącego bądź zmniejszenie wartości MSE jako warunku stopu. Jednym z ograniczeń zaproponowanej identyfikacji jest modelowanie obiektu dla z góry określonego kroku (dla identyfikowanego obiektu h = 0.1). Rozwiązanie takiego problemu zostało jednak zaproponowane w pracy Yi-Jen Wanga i Chin-Teng Lina[18]. Wykorzystane w tym celu zostało połączenie metody Runge-Kutty oraz radialnej sieci neuronowej. Sieć przedstawioną w pracy można wykorzystać w tej metodzie tak aby umożliwić swobodny dobór kroku metody.

W pracy "Structure-unknown non-linear dynamic systems: identification through neural networks"[15] została podjęta próba identyfikacji dynamiki obiektu opisanego równaniem Van der Pola. Zastosowano tam sieć jednokierunkową, trzywarstową oraz algorytm propagacji wstecznej błędu. Nie zostały tam zamieszczone konkretne dane liczbowe, ale oceniając identyfikację obiektu na podstawie zamieszczonych tam wykresów można ocenić jakościowo, że zaproponowany w pracy dyplomowej sposób identyfikacji oparty o sieć neuronową i metodę GOFR zdecydowanie lepiej spełnił swoje zadanie.

Przedstawiona w pracy metoda sprawdziła się w identyfikacji dynamiki obiektu nieliniowego opisanego równaniem Van der Pola. W dalszych pracach można wykorzystać metodę GOFR do identyfikacji innych nieliniowych obiektów dynamicznych oraz zastosować ją w połączeniu z metodą Runge-Kutty.

Literatura

[1] Bartkowiak A., Sieci RBF - o radialnych funkcjach bazowych (popr. 05.12.2009 i 14.12.2010), https://www.ii.uni.wroc.pl/~aba/teach/NN/w9rbf.pdf

[2] Michał Bernardelli, Wprowadzenie do Metod Numerycznych - Zajęcia nr 6, 9 listopada 2011, http://akson.sgh.waw.pl/~mbernard/Dydaktyka/Rok_2010_11/WprowadzenieDoMetodNumerycznychLic_1011_let/Zajecia06.pdf

[3] Bishop C. M., Neural Networks for Pattern Recognition, Oxford University Press, Oxford 1995

[4] A. G. Bors, Introduction of the Radial Basis Function (RBF) Networks, Online Symposium for Electronics Engineers, issue 1, vol. 1, DSP Algorithms: Multimedia, Feb. 13 2001, pp. 1-7.

[5] Chen S., Cowan C. F. N., Grant P. M., Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks, IEEE Transaction on Neural Networks, vol 2, no. 2, p. 302-309, 1991

[6] Choczewski B. (red.), Równania różniczkowe zwyczajne i cząstkowe. Zadania z matematyki, Wydawnictwa AGH, Kraków 2001

[7] Czemplik. A, Modele dynamiki układów fizycznych dla inżynierów Zasady i przykłady konstrukcji modeli dynamicznych obiektów automatyki, WNT, Warszawa 2008

[8] Osowski S., Sieci neuronowe do przetwarzania informacji, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 2006

[9] R. Duboisa, B. Queneta, Y. Faisandierb, G. Dreyfusa, Building meaningful representations for nonlinear modeling of 1d- and 2d-signals: applications to biomedical signals, Neurocomputing 69 (2006), p. 2180–2192

[10] J. Gutenbaum, Modelowanie systemów, Akademicka Oficyna Wydawnicza EXIT, Warszawa 2003

[11] Rolf Isermann, Marco Münchhof, Identification of Dynamic Systems: An Introduction with Applications, Springer, New York 2010

[12] S. Jankowski, Szkic materiałów do wykładu: Sieci neuronowe i neurokomputery, Instytut Systemów Elektronicznych, Politechnika Warszawska, Warszawa 2011

[13] A. Kharab, R. B. Guenther, An introduction to numerical methods : A MATLAB Approach, Chapman & Hall, New York 2002

[14] R. A. Kosiński, Sztuczne sieci neuronowe – dynamika nieliniowa i chaos, wyd. 3 uakt., WNT, Warszawa 2007

[15] S F Masrit, A. G. Chassiakosl, T. K. Caughey, Structure-unknown non-linear dynamic systems: identification through neural networks, Smart Materials ans Structures 1 (1992), 45-56

[16] Palczewski A., Równania różniczkowe zwyczajne Teoria i metody numeryczne z wykorzystaniem komputerowego systemu obliczeń symbolicznych. Wyd. 2, WNT, Warszawa 2004

[17] Marios Tsatsos, Dissertation: Theoretical and Numerical Study of the Van der Pol equation, Aristotle University of Thessaloniki School of Sciences, , Thessaloniki July 2006

[18] Yi-Jen Wang, Chin-Teng Lin, Runge–Kutta Neural Network for Identification of Dynamical Systems in High Accuracy, IEEE Transaction on Neural Networks, vol. 9, no. 2, march 1998

Dodatek A: Kod źródłowy programu

rk_van_der_pol.m

```
1  % Function solving Van der Pol equations usign 4-th order Runge-Kutta
       method
2  %
3  % t_end - end of time period for solution (time starts from 0)
4  % h     - time step
5  % y0    - init values
6
7  function [t,y] = rk_van_der_pol(t_end, h, y0)
8      % Runge-Kutta general formula
9      % y_{i+1} = y_i + 1/6*(k_1 + 2k_2 + 2k_3 + k_4), i = 0,1,2,...
10     % k_1 = hf(x_i,y_1)
11     % k_2 = hf(x_i + 1/2h, y_1 + 1/2k_1)
12     % k_3 = hf(x_i + 1/2h, y_1 + 1/2k_2)
13     % k_4 = hf(x_i + h, y_1 + k_3)
14
15     t = 0:h:t_end;
16     n = length(t);
17     h = 0.1;
18
19     y = zeros(n, 2);
20     y(1, 1) = y0(1);
21     y(1, 2) = y0(2);
22
23     % Van der Pol equation
24     % y''(t) - (1 - y^2(t)) * y'(t) + y(t) = 0;
25     % y'(t)  = y(2)
26     % y''(t) = (1-y(1)^2)*y(2)-y(1)
27     for i = 1:n-1
28         k1 = h*(y(i,2));
29         k2 = h*(y(i,2)+0.5*k1);
30         k3 = h*(y(i,2)+0.5*k2);
31         k4 = h*(y(i,2)+k3);
32         y(i+1,1) = y(i,1)+(k1+2*k2+2*k3+k4)/6;
33
34         y1 = y(i,1);
```

```
35          y2 = y(i,2);
36          k1 = h*((1-y1^2)*y2-y1);
37          y1 = y(i,1)+0.5*k1;
38          y2 = y(i,2)+0.5*k1;
39          k2 = h*((1-y1^2)*y2-y1);
40          y1 = y(i,1)+0.5*k2;
41          y2 = y(i,2)+0.5*k2;
42          k3 = h*((1-y1^2)*y2-y1);
43          y1 = y(i,1)+k3;
44          y2 = y(i,2)+k3;
45          k4 = h*((1-y1^2)*y2-y1);
46          y(i+1,2) = y(i,2)+(k1+2*k2+2*k3+k4)/6;
47      end
48  end
```

gofr.m

```
1  function [selected_rbfs, W, E_k, A_k, Q_k, B_k, centers, sigmas, G] =
       gofr(X, y, G, centers, sigmas, K)
2
3       rbf_number = length(centers);
4
5       D = y;
6       Q = zeros(size(G));
7       Q_k = zeros(size(G));
8       B = zeros(1,rbf_number);
9       B_k = zeros(1,rbf_number);
10      E = zeros(1,rbf_number);
11      E_k = zeros(1,K);
12      selected_rbfs = zeros(1,rbf_number);      % indexes of selected rbfs
            order by decreasing energy
13
14      A = cell(1,rbf_number);
15      A_k = eye(rbf_number);
16      for i = 1:rbf_number
17          A{i} = eye(rbf_number);
18      end
19
20      % ----- selection of the most significant functions -----
```

```
21      for k = 1:K
22          % Gram-Schmidt orthogonalization
23          for i = 1:rbf_number
24             Q(:,i) = G(:,i);
25
26             % if rbf is already selected function omit it
27             if ismember(i, selected_rbfs) == 1
28                 continue;
29             end
30
31             for j=1:k-1
32                  A{i}(j,k) = Q_k(:,j)'*G(:,i) / (Q_k(:,j)'*Q_k(:,j));
33                  Q(:,i) = Q(:,i) - A{i}(j,k)*Q_k(:,j);
34             end
35
36             B(i) = Q(:,i)'*D / (Q(:,i)'*Q(:,i));
37             E(i) = B(i)^2*Q(:,i)'*Q(:,i) / (D'*D);
38          end
39
40          % find RBF with maximum energy (save index and copy to Q_k matrix
               )
41          [E_k(k) selected_rbfs(k)] = max(E);
42          B_k(k) = B(selected_rbfs(k));
43          A_k(:,k) = A{selected_rbfs(k)}(:,k);
44          W = A_k\B_k';
45
46          % ------ Levenberg-Marquardt -----
47          Theta = [W(k); sigmas(selected_rbfs(k)); centers(1, selected_rbfs
               (k)); centers(2, selected_rbfs(k))];
48
49          y_rbf = 0;
50          for j = 1:k
51              y_rbf = y_rbf + W(j) * gaussian_2D(X, sigmas(selected_rbfs(j)
                   ), centers(:,selected_rbfs(j))');
52          end
53
54          lambda = 0.1;
55          Theta_old = Theta;
```

```
56          N = 100;   % maximum iterations of gradient method
57          err = zeros(N,1);
58          err(1) = abs(sum((y - y_rbf).^2)) / length(y);
59          err_old = err(1);
60          for n = 2:N
61              dy_dw = gaussian_2d(X, Theta(2), [Theta(3) Theta(4)]);
62              dy_dsigma = Theta(1) * ((Theta(3) - X(:,1)).^2 + (Theta(4) -
                    X(:,2)).^2) ./ (Theta(2)^3) .* gaussian_2d(X, Theta(2), [
                    Theta(3) Theta(4)]);
63              dy_dc1 = (Theta(1)*(X(:,1) - Theta(3))) ./ (Theta(2)^2) .*
                    gaussian_2d(X, Theta(2), [Theta(3) Theta(4)]);
64              dy_dc2 = (Theta(1)*(X(:,2) - Theta(4))) ./ (Theta(2)^2) .*
                    gaussian_2d(X, Theta(2), [Theta(3) Theta(4)]);
65
66              Z = [dy_dw dy_dsigma dy_dc1 dy_dc2];
67              e = y - y_rbf;
68
69              Theta = Theta + pinv(Z'*Z + lambda*eye(4))*Z'*e;
70              W(k) = Theta(1);
71              sigmas(selected_rbfs(k)) = Theta(2);
72              centers(1,selected_rbfs(k)) = Theta(3);
73              centers(2,selected_rbfs(k)) = Theta(4);
74
75              y_rbf = 0;
76              for j = 1:k
77                  y_rbf = y_rbf + W(j) * gaussian_2D(X, sigmas(
                        selected_rbfs(j)), centers(:,selected_rbfs(j))');
78              end
79
80              err(n) = sum((y - y_rbf).^2) / length(y);
81              if (err(n) >= err_old)
82                  Theta = Theta_old;
83                  lambda = lambda * 10;
84              else
85                  Theta_old = Theta;
86                  err_old = err(n);
87                  lambda = lambda / 10;
88              end
```

```
89
90              % if lambda is too big or too small stop
91              if (lambda > 1e6 || lambda < 1e-6)
92                  break;
93              end;
94
95              if (err(n) < 1e-6)
96                  break;
97              end
98          end;
99
100         % set the optimal parameters
101         W(k) = Theta_old(1);
102         sigmas(selected_rbfs(k)) = Theta_old(2);
103         centers(1,selected_rbfs(k)) = Theta_old(3);
104         centers(2,selected_rbfs(k)) = Theta_old(4);
105
106         % Gram-Schmidt orthogonalization for modified RBF
107         i = selected_rbfs(k);
108         G(:,i) = gaussian_2D(X, Theta(2), [Theta(3) Theta(4)]);
109         Q(:,i) = G(:,i);
110
111         for j=1:k-1
112             A{i}(j,k) = Q_k(:,j)'*G(:,i) / (Q_k(:,j)'*Q_k(:,j));
113             Q(:,i) = Q(:,i) - A{i}(j,k)*Q_k(:,j);
114         end
115
116         B(i) = Q(:,i)'*D / (Q(:,i)'*Q(:,i));
117         B_k(k) = B(i);
118         E_k(k) = B(i)^2*Q(:,i)'*Q(:,i) / (D'*D);
119         A_k(:,k) = A{i}(:,k);
120         Q_k(:,k) = Q(:,i);
121         W = A_k\B_k';
122         E = zeros(size(E));
123     end
124
125     W = A_k\B_k';
126 end
```

generate_library_2d.m

```
% ----- Function generating radial basis function set -----
% N - number of dividing set iteration - change this variable if want
    other number of RBFs
% G - matrix with values of all rbf functions
% centers - vector of centers of all rbf functions
% sigmas - vector of sigma of all rbf functions
function [G,centers,sigmas] = generate_library_2d(X, N)

    rbf_number = 0; % total number of RBFs

    for i = 1:N
        rbf_number = rbf_number + (2^i + 1)^2;
    end

    G = zeros(length(X), rbf_number);
    centers = zeros(2, rbf_number);
    sigmas = zeros(1, rbf_number);
    k = 1;

    max_x = max(max(X));
    min_x = min(min(X));

    for i = 1:N

        iter_rbf_count = 2^i+1;  % number of RBFs in current iteration
        sigma = sqrt(max_x - min_x) / 2^(i-1);

        for j = 1:iter_rbf_count
            % generating centers for rbf functions
            for l = 1:iter_rbf_count
                centers(1, k) = min_x + (max_x - min_x) / (iter_rbf_count
                    -1) * (j-1);
                centers(2, k) = min_x + (max_x - min_x) / (iter_rbf_count
                    -1) * (l-1);
                G(:,k) = gaussian_2d(X, sigma, centers(:,k)');
                sigmas(k) = sigma;
```

```
34                  k = k + 1;
35              end
36          end
37      end
38  end
```

gofr_van_der_pol.m

```
1  % Neural network with radial basis functions (RBF) approximating
2  % Van der Pol equation using Orthogonal Generalized Orthogonal Forward
3  % Regression (OFR)
4
5  close all;
6  clear all;
7  clc;
8
9  single_trajectory = true; % switch between training based on one
       trajectory or many trajectories
10
11 t_end = 40;          % time end of trajectory (always starts from time =
       0)
12 t_step = 0.1;        % time step
13
14 x1 = 0;
15 x2 = 2;
16 if (single_trajectory)
17 else
18      x1 = -2:0.5:2;
19      x2 = x1;
20 end
21
22 X = [];
23 y1 = [];
24 y2 = [];
25 for i = x1
26      for j = x2
27          if (i == 0 && j == 0)
28              continue;
29          end
```

```
30          [t Y] = rk_van_der_pol(t_end, t_step, [i j]);
31          X = [X; Y(1:end-1,:)];
32          y1 = [y1; Y(2:end,1)];
33          y2 = [y2; Y(2:end,2)];
34      end
35  end
36
37  N = 3; % change number of library division (number of RBFs)
38  MAX_RBFS = 30; % change number of maximum possible selected RBFs
39
40  err = zeros(MAX_RBFS,2);
41  stop_condition = 1e-6;
42  if (single_trajectory)
43  else
44      stop_condition = 1e-5;
45  end
46
47  % Training y(t) of Van der Pol equation
48  for K1 = 1:MAX_RBFS
49      K1
50      % ----- generating radial basis function set -----
51      [G1 centers1 sigmas1] = generate_library_2d(X, N);
52      % ----- teach RBF neural network -----
53      [selected_rbfs1, W1, E_k1, A_k1, Q_k1, B_k1, centers1, sigmas1, G1] =
              gofr(X, y1, G1, centers1, sigmas1, K1);
54      y_rbf1 = 0;
55      for i = 1:K1
56          y_rbf1 = y_rbf1 + W1(i) * G1(:,selected_rbfs1(i));
57      end
58
59      % --- uncomment to show network after each selected RBF
60  %     figure(1)
61  %     plot(0:t_step:length(y1)*t_step-t_step, y1, 'r', 0:t_step:length(
        y_rbf1)*t_step-t_step, y_rbf1, 'b');
62  %     title({'y(t) of Van der Pol equation'; sprintf('Iteration nr = %d',
        K1)});
63  %     legend('desired','approximated');
64  %     pause;
```

```
65
66      err(K1,1) = sum((y1 - y_rbf1).^2) / length(y1);
67      if (err(K1,1) < stop_condition)
68          break;
69      end
70  end
71
72  % Training y'(t) of Van der Pol equation
73  for K2 = 1:MAX_RBFS
74      K2
75      [G2 centers2 sigmas2] = generate_library_2d(X, N);
76      [selected_rbfs2, W2, E_k2, A_k2, Q_k2, B_k2, centers2, sigmas2, G2] =
            gofr(X, y2, G2, centers2, sigmas2, K2);
77      y_rbf2 = 0;
78      for i = 1:K2
79          y_rbf2 = y_rbf2 + W2(i) * G2(:,selected_rbfs2(i));
80      end
81
82      % --- uncomment to show network after each selected RBF
83  %     figure(1)
84  %     plot(0:t_step:length(y2)*t_step-t_step, y2, 'r', 0:t_step:length(
        y_rbf2)*t_step-t_step, y_rbf2, 'b');
85  %     title({'y''(t) of Van der Pol equation'; sprintf('Iteration nr = %d
        ', K2)});
86  %     legend('desired','approximated');
87  %     pause;
88
89      err(K2,2) = sum((y2 - y_rbf2).^2) / length(y2);
90      if (err(K2,2) < stop_condition)
91          break;
92      end
93  end
```

check_prediction.m

```
1  % Script for testing 100-step prediction for set of 440 initial condition
2  % First run script gofr_van_der_pol.m to retrieve data!!!
3  close all;
4
```

```
5  MSE_x1 = [];
6  MSE_x2 = [];
7  n = 0;
8  centers = 0;
9  i_x1 = 0;
10
11 % calulate error for all 440 initial conditions
12 for x1 = -2:0.2:2
13     i_x2 = 0;
14     i_x1 = i_x1 + 1;
15     for x2 = -2:0.2:2
16         i_x2 = i_x2 + 1;
17         n = n + 1
18         centers(n,1) = x1;
19         centers(n,2) = x2;
20         if (x1 == 0 && x2 == 0)
21             MSE2_x1(i_x1, i_x2) = 0;
22             MSE2_x2(i_x1, i_x2) = 0;
23             continue;
24         end
25
26     P = 100;        % prediction steps
27     y_rbf1 = zeros(1,P);
28     y_rbf2 = zeros(1,P);
29     y_rbf1(1) = x1;
30     y_rbf2(1) = x2;
31     [t Y] = rk_van_der_pol(P*0.1-0.1, 0.1, [y_rbf1(1) y_rbf2(1)]);
32     y1 = Y(:,1);
33     y2 = Y(:,2);
34
35     y_rbf1(1) = y1(1);
36     y_rbf2(1) = y2(1);
37
38     for j = 2:P
39         for i = 1:K1
40             y_rbf1(j) = y_rbf1(j) + W1(i) * gaussian_2D([y_rbf1(j-1)
                   y_rbf2(j-1)], sigmas1(selected_rbfs1(i)), centers1(:,
                   selected_rbfs1(i))');
```

```
41          end
42
43          for i = 1:K2
44              y_rbf2(j) = y_rbf2(j) + W2(i) * gaussian_2D([y_rbf1(j-1)
                    y_rbf2(j-1)], sigmas2(selected_rbfs2(i)), centers2(:,
                    selected_rbfs2(i))');
45          end
46      end
47
48      MSE_x1(i_x1, i_x2) = sum((y_rbf1(1:length(y1))' - y1).^2) / length(y1
            );
49      MSE_x2(i_x1, i_x2) = sum((y_rbf2(1:length(y2))' - y2).^2) / length(y2
            );
50
51      end
52  end
53
54  % show graphs with MSE for initital conditons
55  [t Y] = rk_van_der_pol(t_end, t_step, [0 2]);
56
57  figure(1)
58  pcolor(-2:0.2:2, -2:0.2:2, MSE_x1');
59  shading flat;
60  axis([-2.5 2.5 -3 3]);
61  hold on
62  plot(Y(:,1),Y(:,2),'g');
63  colorbar;
64
65  MSE_x2(find(MSE_x2 >= 10)) = 10;
66
67  figure(2)
68  pcolor(-2:0.2:2, -2:0.2:2, MSE_x2');
69  shading flat;
70  axis([-2.5 2.5 -3 3]);
71  hold on
72  plot(Y(:,1),Y(:,2),'g');
73  colorbar;
```