# JARINGAN SARAF TIRUAN DENGAN METODE PEMBELAJARAN PROPAGASI BALIK DALAM SISTEM KENDALI DENGAN METODE *FINE-TUNING* DAN *DIRECT INVERSE CONTROL*


**Disusun Oleh:**

Bryan Indarto Giovanni Firjatulloh (2006531996)

**PROGRAM STUDI TEKNIK ELEKTRO**

**FAKULTAS TEKNIK**

**UNIVERSITAS INDONESIA**

**DEPOK**

# BAB I
# PENDAHULUAN

## 1.1 Latar Belakang

Seiring berjalannya waktu, masalah manusia menjadi semakin beragam. Dalam dunia perkembangan teknologi, teknologi yang berguna adalah teknologi yang memecahkan masalah. Ada banyak masalah kompleks yang dihadapi orang. Kompleksitas ini dapat disebabkan oleh banyak variabel yang perlu dipertimbangkan dan dikaitkan ketika memecahkan masalah. Di zaman sekarang ini, kami memiliki teknologi yang disebut jaringan saraf tiruan yang dapat membantu menjelaskan banyak variabel dan kompleksitas masalah. Jaringan merupakan hasil proses komputer yang mengolah data dengan meniru cara kerja otak manusia. Teknik jaringan saraf tiruan telah diimplementasikan dalam berbagai aplikasi, termasuk diagnosis penyakit, pengenalan ucapan komputer, dan prediksi harga saham.

Lonjakan kesadaran sistem dinamik dalam dunia ini mengharuskan implementasi dari sistem yang dapat beradaptasi secara terus-menerus sehingga terciptalah sebuah solusi yaitu *Neural Network*. Namun dengan segala macam metode untuk membuat model matematika yang cocok masih belum ditemukan cara untuk memprediksi data yang paling ideal. Oleh karena itu, terdapat inovasi fine-tuning yang berguna untuk membuat sebuah model matematika cocok dengan semua datanya dengan perubahan nilai dari parameternya. Dari penyempurnaan dari datanya pula diinovasikan sebuah metode yaitu dengan metode DIC yang digunakan agar data dari hasil pelatihan maupun testing dapat lebih presisi sesuai dengan kebutuhan terutama apabila datanya bernilai random.

## 1.2 Rumusan Masalah

1.2.1 Bagaimana konsep dasar dari *Neural Network*?

1.2.2 Bagaimana cara kerja dari *Backpropagation Neural Network*?

1.2.3 Bagaimana *Neural Network* dapat memprediksi keluaran dari suatu sistem?

1.2.4 Bagaimana fine-tuning bekerja dalam sebuah sistem?

1.2.5 Bagaimana metode DIC bekerja pada sistem?

## 1.3 Tujuan Penelitian

1.3.1 Memahami konsep dasar dari *Neural Network*

1.3.2 Memahami algoritma penyusunan *Neural Network* yang mencakup cara kerja dari *backpropagation*

1.3.3 Memahami cara kerja dari fine-tuning

1.3.4 Mengimplementasikan *Neural Network* yang sudah di-*tuning* untuk memprediksi sebuah keluaran dari suatu sistem

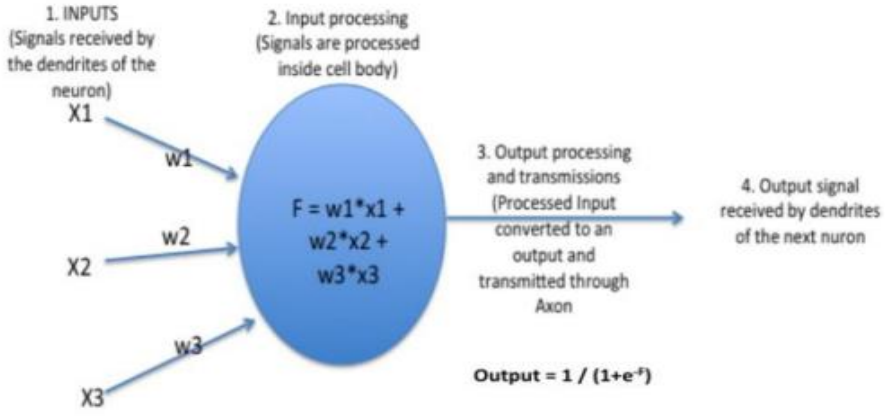
1.3.5 Memahami cara kerja dari metode DIC

# BAB II
# TINJAUAN PUSTAKA

## 2.1 Jaringan Saraf Tiruan (*Neural Network)*

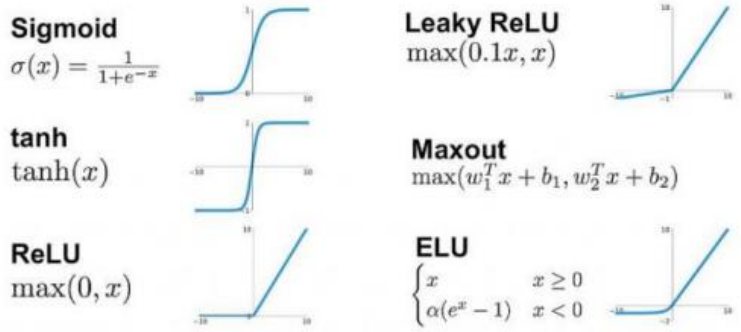
Jaringan syaraf tiruan (JST) adalah sebuah algoritma model matematika adaptif untuk memodelkan dan memprediksi masalah yang kompleks (misalnya sistem dinamis non-linier) dengan meniru jaringan otak manusia saat mengenali pola. Otak manusia memiliki jutaan sel saraf yang dapat melakukan proses ini Informasi berupa sinyal listrik. Dendrit menerima informasi dari luar Neuron yang diproses di badan neuron dan diubah menjadi output ini ditransmisikan ke neuron lain melalui akson. Neuron berikutnya dapat ditentukan apakah sinyal diterima atau ditolak.

Jaringan saraf tiruan menggunakan persamaan untuk meniru cara kerja otak manusia secara matematis. Informasi dari luar diwakili oleh simbol xn, di mana n adalah Penomoran informasi. Semua informasi memiliki bobot dan diwakili oleh Wn. Ada juga input berupa nilai bias. semua informasi masukan dan dapat dijumlahkan dan diproses dalam neuron buatan menggunakan rumus matematika fungsi aktivasi. Ada beberapa fungsi aktivasi yang tersedia seperti fungsi Sigmoid, Tanh, ReLU, dan beberapa lainnya. Fungsi aktivasi menentukan hubungan nonlinier antara input dan mengubahnya menjadi output. Hasil prosesnya terlihat seperti



ini:

*Gambar 2.1 Struktur neuron pada jaringan saraf tiruan*




*Gambar 2.2 Struktur neuron pada jaringan saraf tiruan*

Di otak manusia, jutaan neuron membentuk beberapa lapisan yang mewakili tahapan dalam proses pemrosesan sinyal. Dalam jaringan saraf tiruan, neuron buatan juga membentuk lapisan, yaitu lapisan input dan output, dengan kemungkinan menambahkan lapisan tambahan di antaranya, yaitu lapisan tersembunyi. Lapisan masukan menerima informasi dengan bobot tertentu dan diproses oleh lapisan tersembunyi untuk membentuk lapisan keluaran. Pada proses klasifikasi, keluarannya adalah prediksi klasifikasi dari sinyal informasi input.

## 2.2 *Backpropagation Method*

Metode backpropagation, atau *backpropagation*, adalah metode pelatihan jaringan saraf tiruan yang menyesuaikan bobot jaringan saraf tiruan berdasarkan output yang diinginkan dan tujuan yang diinginkan untuk meminimalkan nilai kesalahan. Untuk mendapatkan nilai kesalahan minimum, gunakan penurunan gradien untuk menemukan arah yang berlawanan dari turunan parsial dari fungsi kesalahan. Kemudian sesuaikan bobot antar layer hingga fungsi error menjadi titik minimum keseluruhan. Fungsi kesalahan yang digunakan pada umumnya adalah fungsi *quadratic error* yaitu:

$$E = \frac{1}{2}\sum_{i=1}^{n}(y_i - t_i)^2$$

### 2.2.1 Parameter Pembelajaran Metode Propagasi-balik

Terdapat beberapa parameter yang dapat mempengaruhi hasil pembelajaran propagasi-balik yaitu:

a. Bobot awal

Bobot jaringan syaraf tiruan harus ditentukan terlebih dahulu sebelum proses pelatihan. Anda dapat menentukan apakah jaringan saraf telah mencapai fungsi kesalahan minimum global atau lokal dan seberapa cepat jaringan saraf menyatu. Beberapa metode yang dapat digunakan untuk menentukan nilai bobot awal adalah:

1. Random/acak
2. Nguyen-Widrow

Metode Nguyen-Widrow merupakan modifikasi metode acak dimana bobot dimodifikasi terlebih dahulu sebelum proses pelatihan. Algoritma dari metode Nguyen-Widrow adalah sebagai berikut:

- Inisialisasi dengan bilangan acak antara -0,5 sampai 0,5 untuk bias dan bobot antara neuron di lapisan tersembunyi dengan neuron di lapisan keluaran.
- Untuk inisialisasi bias dan bobot antara neuron di lapisan masukan dengan lapisan tersembunyi:
    - Tentukan faktor skala $\beta = 0{,}7P^{1/N}$, dengan P adalah ukuran lapisan tersembunyi dan N adalah ukuran lapisan masukan.
    - Inisialisasi bobot $v_{ij}$ dengan bilangan acak antara -0,5 sampai 0,5.
    - Hitung norma dari vektor bobot dengan $\|v_{ij}\| = \sqrt{\sum_{i=1}^{p} {v_{ij}}^2}$
    - Menyesuaikan nilai bobot dengan $v_{ij} = = \frac{\beta vij}{\|vij\|}$
    - Inisialisasi bias dengan bilangan acak antara -0,5 sampai 0,5.

b. Laju pembelajaran (α)

Laju pembelajaran, atau *learning Rate*, menentukan seberapa banyak penyesuaian bobot yang dilakukan. Nilai yang terlalu kecil untuk kecepatan

pembelajaran memperlambat proses konvergensi jaringan, dan nilai yang terlalu besar membuat jaringan tidak stabil.

c. Koefisien Momentum (μ)

Momentum merupakan nilai tambah pada penyesuaian bobot untuk mempercepat proses pembelajaran. Hal ini dilakukan dengan menambahkan arah penyesuaian bobot dari iterasi sebelumnya. Besarnya efek momentum ditentukan oleh nilai koefisien momentum. Persamaan penyesuaian bobot dengan momentum adalah:

$$\Delta w_{jk}(\mathrm{t}) = \alpha\delta_k z_j + \mu\Delta w_{jk}(\mathrm{t}-1)$$

### 2.2.2 Algoritma Metode Propagasi-balik

Untuk mengimplementasikan metode backpropagation, terlebih dahulu dilakukan normalisasi data yang digunakan agar range datanya sama dan semua input memiliki efek yang terukur. Salah satu metode normalisasi data adalah min-max. Ini adalah normalisasi di mana data akan dikecilkan sesuai perbandingan terhadap nilai terbesar yang menghasilkan angka diantara 0 - 1. Nilai *min-max normalization* dihitung dengan persamaan

$$x(k) = \frac{x(k) - x_{min}}{x(k) - x_{max}}$$

Dengan x adalah variabel acak, μ adalah mean, dan σ adalah standar deviasi. Setelah dinormalisasi, data dibagi jadi 2 yaitu data untuk pelatihan dan data untuk pengujian. Selanjutnya dilakukan inisialisasi bobot, kemudian ke proses pelatihan yang terdiri dari proses *feedforward* dan *backpropagation of error*. Kedua proses dilakukan secara kontinu selama kondisi *stopping* bernilai *False*.

a. Proses *Feedforward*

1. Komputasi lapisan masukan. Untuk setiap masukan $x_i$, i=1, 2…, n :
   - Menerima input $x_n$
   - Mengirim ke lapisan tersembunyi setelahnya

2. Komputasi lapisan tersembunyi. Untuk setiap unit neuron tersembunyi $z_j$, j=1, 2…p:
   - Menghitung sinyal masukan beserta bobotnya dengan persamaan $z_{in_j=v_{oj}+\sum x_i v_{ij}}$
   - Menghitung nilai aktivasi setiap unit sebagai keluarannya dengan persamaan $z_j = f(z_{in_j})$
   - Mengirimkan nilai aktivasi ke neuron pada lapisan selanjutnya
3. Komputasi lapisan keluaran. Untuk setiap keluaran $y_k$, k=1,2,…,m:
   - Menghitung sinyal masukan beserta bobotnya dengan persamaan $y_{in_{k=w_{ok}+\sum z_j w_{jk}}}$
   - Menghitung nilai aktivasi setiap unit sebagai keluarannya dengan persamaan $y_k = f(y_{in_k})$
   - Mengkuantisasi nilai keluaran dengan persamaan

$$y_k = \begin{cases} 0 \le y_k < 0.3; y_k = 0 \\ 0.7 < y_k \le 1; y_k = 1 \\ else; y_k = y_k \end{cases}$$

   -

b. Proses *Backpropagation of Error*

1. Komputasi error di lapisan keluaran. Untuk setiap keluaran $y_k$, k=1,2,…,m:
   - Menerima target yang bersesuaian dengan masukan
   - Menghitung nilai galat dengan persamaan $\delta_k = (t_k - y_k) * f'(y\_in_k)$
   - Menghitung besar koreksi bobot unit keluaran dengan persamaan $\Delta w_{jk} = \alpha \frac{\delta E(w_j k_j)}{\delta w_{jk}} = \alpha \delta_k z_j$
   - Mengirimkan $\delta_k$ ke unit di lapisan sebelumnya
2. Komputasi error di lapisan tersembunyi. Untuk setiap unit tersembunyi $z_j$, j=1, 2,…,p:
   - Menghitung semua koreksi error dengan persamaan $\delta_{in_j} = \sum \delta_k w_{jk}$
   - Menghitung nilai aktivasi koreksi error dengan persamaan $\delta_j = \delta_{in_j} * f'(z_{in_j})$

- Menghitung besar koreksi bobot unit tersembunyi dengan persamaan $\Delta v_{ij} = \alpha \delta_j x_j$
- Menghitung besar koreksi bias unit tersembunyi dengan persamaan $\Delta v_{oj} = \alpha \delta_j$

3. Memperbarui bobot dan bias
   - Untuk setiap bobot dan bias dari setiap unit keluaran $y_k$, k=1,2,…,m:
     - $w_{jk}(baru) = w_{jk}(lama) + \Delta w_{jk}(baru) + \mu \Delta w_{jk}(lama)$
     - $w_{ok}(baru) = w_{ok}(lama) + \Delta w_{ok}(baru) + \mu \Delta w_{ok}(lama)$
   - Untuk setiap bobot dan bias dari setiap unit tersembunyi $z_j$, j=1,2,…,p:
     - $v_{ij}(baru) = v_{ij}(lama) + \Delta v_{ij}(baru) + \mu \Delta v_{ij}(lama)$
     - $v_{oj}(baru) = v_{oj}(lama) + \Delta v_{oj}(baru) + \mu \Delta v_{oj}(lama)$
4. Menghitung error dengan fungsi error kuadratis. Jika error belum memenuhi ketentuan maka proses pelatihan dilakukan lagi dari proses *feedforward*. Jika memenuhi ketentuan maka proses pelatihan selesai.

Setelah proses pelatihan selesai, maka proses pengujian akan berjalan. Untuk menguji jaringan saraf tiruan, jalankan proses feedforward pada data uji dan bandingkan dengan target. Benar dan salahnya antara prediksi dengan target dilabel kemudian dihitung kualitas model jaringan dengan parameter nilai MSE/RMSE.

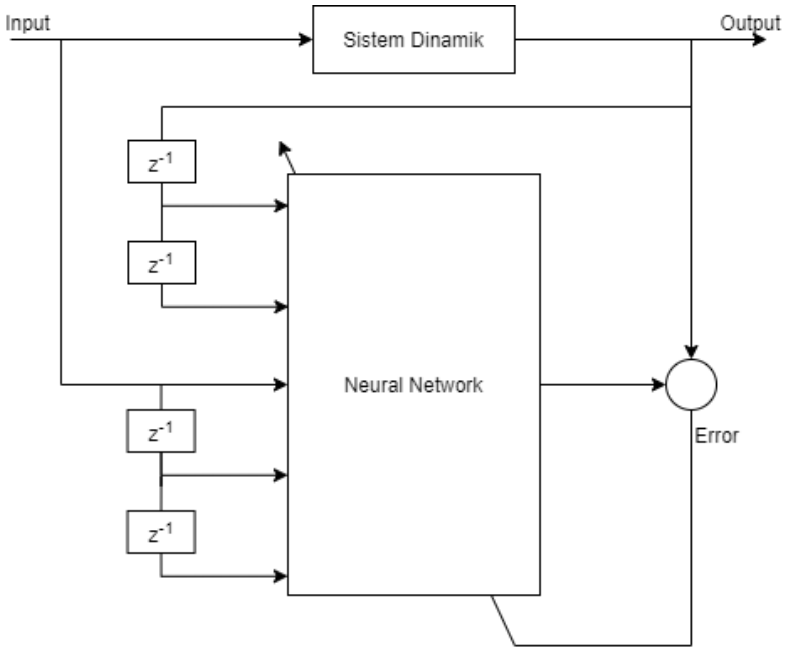
## 2.3 Jaringan Saraf Tiruan (*Neural Network*) untuk Sistem Kendali

Jaringan saraf tiruan untuk sistem kendali dibuat tanpa menggunakan sebuah model matematis dari sebuah dataset. Jaringan saraf tiruan akan bertindak sebagai penganalisa dari dataset untuk menciptakan sebuah model matematis tiruan yang akan terus berubah sesuai dengan setiap input (adaptif). Terdapat beberapa perbedaan yang dapat dilihat antara jaringan saraf tiruan untuk sistem kendali dengan *pattern recognition*. Berikut adalah beberapa perbedaannya:

| | **Pattern recognition** | **Sistem Kendali** |
|---|---|---|
| Ukuran sukses | Recognition rate | MSE/RMSE |

| | | |
|---|---|---|
| Dimensi input | Relatif besar | Relatif kecil |
| Normalisasi | Berkisar dari 0-1 | Berkisar dari -1 – 1 |
| Dimensi input | Memiliki range sama | Memiliki range berbeda |

Jaringan saraf tiruan pada sistem kendali sendiri memiliki proses pembuatan dengan identifikasi yang berbeda dari jaringan saraf untuk *pattern recognition* yaitu dengan blok diagram. Di bawah ini adalah neural network untuk sistem kendali apabila dibuat menjadi sebuah blok diagram.




## 2.4 *Fine-tuning*

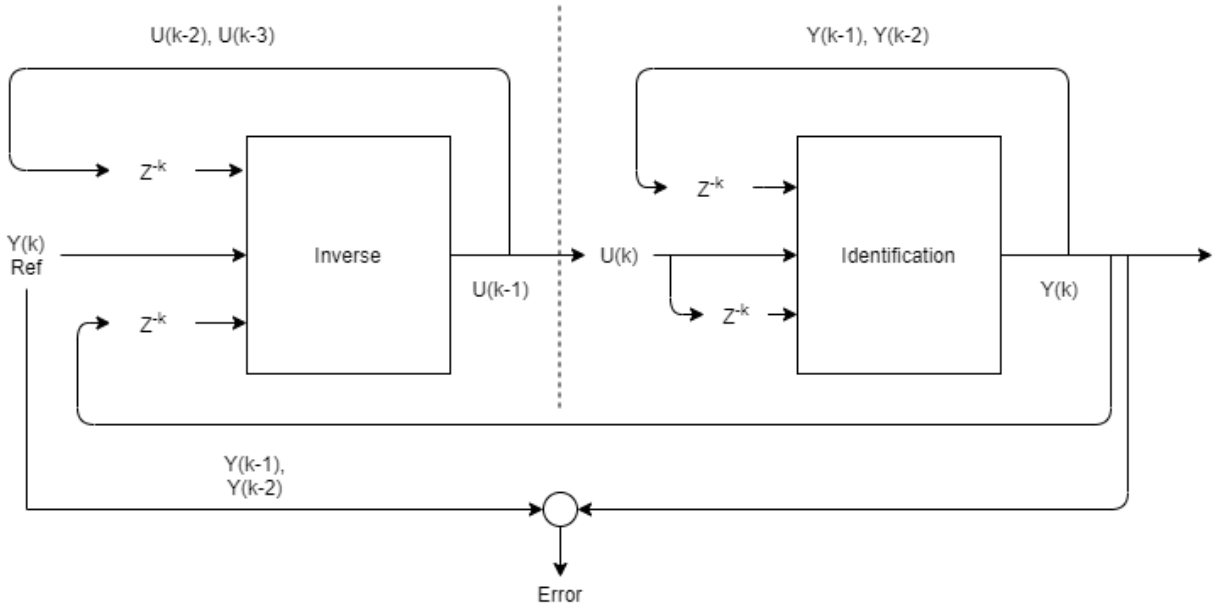
Fine-tuning merupakan proses yang menyesuaikan nilai parameter dari fungsi model matematika agar fit terhadap keluaran yang diinginkan. Hal ini dilakukan dengan memberlakukan model matematika dengan bobot dan bias yang sudah dilatih terhadap dataset yang baru. Hal ini akan menentukan nilai dari output di masing-masing row sehingga setelah fine tuning, nilai ketepatan antara data set dengan prediksi lebih tepat. Hal ini dilakukan saat proses terjadinya training dimana nilai masing-masing output dari feed-forward kemudian akan di pass ke nilai output pada kolom kedua dan ketiga.

## 2.4 *Jaringan Sistem Kendali Inverse (Direct Inverse Control)*

Sistem kendali inversi adalah salah satu teknik pengendali yang menggunakan fungsi invers dari plant untuk sistem pengendalinya secara open loop. Hal ini akan menyebabkan hasil output dari pengendali dijadikan sebagai fungsi input plant yang keluarannya akan sama dengan referensi masukan input. Sistem kendali inversi ini secara matematis digambarkan dalam rumus menggunakan ANN sebagai berikut:

$$u(k-1) = f(u(k-2), u(k-3), y(k), y(k-1))$$

Hal ini dapat direpresentasikan dalam blok diagram sesuai dengan kaidah sistem kendali yang sudah dijelaskan di bagian sebelumnya. Bentuk dari blok diagram akan seperti berikut:



# BAB III
# PEMBAHASAN

## 3.1. Deskripsi *Dataset*

*Dataset* menggunakan nilai variasi acak -1 sampai 1 pada input dan menggunakan komputasi tertentu pada output. *Dataset* ini yang nantinya akan digunakan sebagai dasar dari pelatihan dan pengujian mode *neural network*. Output atau keluaran bagian dari data non-liniear dari sistem awalnya direpresentasikan dengan persamaan yaitu:

$$y(k) = \frac{1}{1 + (y(k-1)^2)} + 0.25u(k) - 0.3u(k-1)$$

u(k) merupakan input atau masukan sementara y(k) adalah keluaran atau output. Dari situ kemudian dibutlah dataset yang berisi 6 kolom yang bernilai yaitu y(k), y(k-1), y(k-2), u(k), u(k-1), dan u(k-2). Sementara dengan data yang linier dapat dirumuskan outputnya dengan persamaan berikut:

$$y(k) = 0.04251.u(k-1) + 0.04044.u(k-2) + 1.778.y(k-1) - 0.8607.y(k-2)$$

Dengan prinsip linier dan non-linier ini kemudian akan diimplementasikan ke dalam beberapa jenis data yaitu data random, data step, dan data sine. Pada bagian dataset random linear digenerasi dengan program dibawah ini:

```
% Struktur data: y(k) | y(k-1) | y(k-2) | y(k-3) | u(k) | u(k-1)
| u(k-2) |
% u(k-3)

data = zeros(3010, 8);

% Random u(k-3)
data(:, 8) = rand(3010, 1);

% u(k-2), berdasarkan kolom u(k-3)
data(1:end-1, 7) = data(2:end, 8);
data(end, 7) = rand(1, 1);

% u(k-1), berdasarkan kolom u(k-2)
data(1:end-1, 6) = data(2:end, 7);
data(end, 6) = rand(1, 1);

% u(k), berdasarkan kolom u(k-1)
data(1:end-1, 5) = data(2:end, 6);
```

```
data(end, 5) = rand(1, 1);

% Random y(-2) untuk kolom y(k-3)
data(1, 4) = rand(1, 1);

% Random y(-1) untuk kolom y(k-2)
data(1, 3) = rand(1, 1);

% Random y(0) untuk kolom y(k-1)
data(1, 2) = rand(1, 1);

% y(1) untuk kolom y(k)
data(1, 1) = 0.04251*(data(1, 5)) + 0.04044*(data(1, 6)) +
1.778*(data(1, 2)) - 0.8607*(data(1, 3));

% Memindahkan y(k) ke y(k-1), y(k-2), dan y(k-3) yang sesuai,
serta generate y(k) baru
for k = 2:3010
    data(k, 4) = data(k-1, 3);
    data(k, 3) = data(k-1, 2);
    data(k, 2) = data(k-1, 1);
    data(k, 1) =  0.04251*(data(k, 5)) + 0.04044*(data(k, 6)) +
1.778*(data(k, 2)) - 0.8607*(data(k, 3));
end

save('data2_more.mat', 'data');
```

Sementara dataset random nonlinear akan digenerasi dengan program dibawah ini:

```
% Struktur data: y(k) | y(k-1) | y(k-2) | u(k) | u(k-1) | u(k-2)

data = zeros(3010, 6);

% Random u(k-2)
data(:, 6) = rand(3010, 1) * 20 - 10;

% u(k-1), berdasarkan kolom u(k-2)
data(1:end-1, 5) = data(2:end, 6);
data(end, 5) = rand(1, 1) * 20 - 10;

% u(k), berdasarkan kolom u(k-1)
data(1:end-1, 4) = data(2:end, 5);
data(end, 4) = rand(1, 1) * 20 - 10;

% Random y(-1) untuk kolom y(k-2)
data(1, 3) = 0;

% Random y(0) untuk kolom y(k-1)
data(1, 2) = 0;

% y(1) untuk kolom y(k)
```

```
data(1, 1) = (1/(1 + (data(1, 2)^2))) + (0.25*data(1, 4)) -
(0.3*data(1, 5));

% Memindahkan y(k) ke y(k-1) dan y(k-2) yang sesuai, serta
generate y(k) baru
for k = 2:3010
    data(k, 3) = data(k-1, 2);
    data(k, 2) = data(k-1, 1);
    data(k, 1) = (1/(1 + (data(k, 2)^2))) + (0.25*data(k, 4)) -
(0.3*data(k, 5));
end

save('data1.mat', 'data');
```

Pada dataset ketiga kemudian menggunakan input atau masukan berupa sinyal sinusoidal. Data input sinusoidal berasal dari rumus yaitu:

$$u(k) = A.\sin(\omega k)$$

Program yang akan mengenerasi data sin linear adalah program dibawah ini:

```
% Struktur data: y(k) | y(k-1) | y(k-2) | y(k-3) | u(k) | u(k-1)
| u(k-2) |
% u(k-3)

data = zeros(3010, 8);
x = linspace(1, 3013/100, 3013);

% Random u(k-3)
data(:, 8) = sin(x(1:3010));

% Random u(k-2)
data(:, 7) = sin(x(2:3011));

% u(k-1), berdasarkan kolom u(k-2)
data(:, 6) = sin(x(3:3012));

% u(k), berdasarkan kolom u(k-1)
data(:, 5) = sin(x(4:3013));

% Output feedback dibuat 0 terlebih dahulu karena belum ada
output pada
% detik 0
data(1, 4) = 0;
data(1, 3) = 0;
data(1, 2) = 0;

% y(1) untuk kolom y(k)
```

```
data(1, 1) = 0.04251*(data(1, 5)) + 0.04044*(data(1, 6)) +
1.778*(data(1, 2)) - 0.8607*(data(1, 3));

% Memindahkan y(k) ke y(k-1) dan y(k-2) yang sesuai, serta
generate y(k) baru
for k = 2:3010
    data(k, 4) = data(k-1, 3);
    data(k, 3) = data(k-1, 2);
    data(k, 2) = data(k-1, 1);
    data(k, 1) = 0.04251*(data(k, 5)) + 0.04044*(data(k, 6)) +
1.778*(data(k, 2)) - 0.8607*(data(k, 3));
end

save('sine2.mat', 'data');
```

Program yang akan mengenerasi data sin nonlinear adalah program dibawah ini:

```
% Struktur data: y(k) | y(k-1) | y(k-2) | u(k) | u(k-1) | u(k-2)

data = zeros(3010, 6);
x = linspace(1, 1000, 3012);

% Random u(k-2)
data(:, 6) = sin(x(1:3010));

% u(k-1), berdasarkan kolom u(k-2)
data(:, 5) = sin(x(2:3011));

% u(k), berdasarkan kolom u(k-1)
data(:, 4) = sin(x(3:3012));

% Output feedback dibuat 0 terlebih dahulu karena belum ada
output pada
% detik 0
data(1, 3) = 0;
data(1, 2) = 0;

% y(1) untuk kolom y(k)
data(1, 1) = (1/(1 + (data(1, 2)^2))) + (0.25*data(1, 4)) -
(0.3*data(1, 5));

% Memindahkan y(k) ke y(k-1) dan y(k-2) yang sesuai, serta
generate y(k) baru
for k = 2:3010
    data(k, 3) = data(k-1, 2);
    data(k, 2) = data(k-1, 1);
    data(k, 1) = (1/(1 + (data(k, 2)^2))) + (0.25*data(k, 4)) -
(0.3*data(k, 5));
end

save('sine.mat', 'data');
```

Pada bagian dataset step, maka semua inputnya diberikan nilai 1 sehingga pada rumusnya yaitu:

$$u(k) = 1$$

Dari rumus step kemudian dibuat program dibawah ini yang mengenerasi data step input linear:

```
% Struktur data: y(k) | y(k-1) | y(k-2) | y(k-3) | u(k) | u(k-1)
| u(k-2) |
% u(k-3)

data = zeros(3010, 8);
x = [zeros(1, 13) ones(1, 3010)];

% Random u(k-3)
data(:, 8) = x(1:3010);

% Random u(k-2)
data(:, 7) = x(2:3011);

% u(k-1), berdasarkan kolom u(k-2)
data(:, 6) = x(3:3012);

% u(k), berdasarkan kolom u(k-1)
data(:, 5) = x(4:3013);

% Output feedback dibuat 0 terlebih dahulu karena belum ada
output pada
% detik 0
data(1, 4) = 0;
data(1, 3) = 0;
data(1, 2) = 0;

% y(1) untuk kolom y(k)
data(1, 1) = 0.04251*(data(1, 5)) + 0.04044*(data(1, 6)) +
1.778*(data(1, 2)) - 0.8607*(data(1, 3));

% Memindahkan y(k) ke y(k-1) dan y(k-2) yang sesuai, serta
generate y(k) baru
for k = 2:3010
    data(k, 4) = data(k-1, 3);
    data(k, 3) = data(k-1, 2);
    data(k, 2) = data(k-1, 1);
```

```
    data(k, 1) = 0.04251*(data(k, 5)) + 0.04044*(data(k, 6)) +
1.778*(data(k, 2)) - 0.8607*(data(k, 3));
end

save('step2.mat', 'data');
```

Dari rumus step kemudian dibuat program dibawah ini yang mengenerasi data step input nonlinear:

```
% Struktur data: y(k) | y(k-1) | y(k-2) | u(k) | u(k-1) | u(k-2)

data = zeros(3010, 6);
x = [zeros(1, 12) ones(1, 3010)];

% Random u(k-2)
data(:, 6) = x(1:3010);

% u(k-1), berdasarkan kolom u(k-2)
data(:, 5) = x(2:3011);

% u(k), berdasarkan kolom u(k-1)
data(:, 4) = x(3:3012);

% Output feedback dibuat 0 terlebih dahulu karena belum ada
output pada
% detik 0
data(1, 3) = 0;
data(1, 2) = 0;

% y(1) untuk kolom y(k)
data(1, 1) = (1/(1 + (data(1, 2)^2))) + (0.25*data(1, 4)) -
(0.3*data(1, 5));

% Memindahkan y(k) ke y(k-1) dan y(k-2) yang sesuai, serta
generate y(k) baru
for k = 2:3010
    data(k, 3) = data(k-1, 2);
    data(k, 2) = data(k-1, 1);
    data(k, 1) = (1/(1 + (data(k, 2)^2))) + (0.25*data(k, 4)) -
(0.3*data(k, 5));
end

save('step.mat', 'data');
```

Dataset digunakan untuk identifikasi sistem dan membuat *inverse controller*. Tiap identifikasi sistem dan pembuatan *inverse controller* dilakukan pelatihan dan pengujian.

Pada bagian pelatihan, dilakukan proses feedforward dan backpropagation of error untuk menyesuaikan bobot yang terus diulang hingga hasil pelatihan kurang dari error atau melebihi epoch maksimum. Fungsi aktivasi yang digunakan adalah tanh pada saat input layer ke hidden layer supaya dapat mencakup output dari -1 hingga 1 dan linear pada saat hidden layer ke ouput layer agar output yang dihasilkan tidak dibatasi.

Pada bagian pengujian, dilakukan proses feedforward untuk mengetahui performa model yang telah dibuat. Parameter fungsi kesalahan yang digunakan adalah MSE dan RMSE.

Setelah dilakukan keduanya, selanjutnya adalah penggabungan keduanya agar mendapatkan Direct Inverse Control (DIC). Output yang dihasilkan dari plant yaitu y(k) akan dijadikan *setpoint* yang digunakan pada sistem DIC.

## 3.2. Implementasi Algoritma pada MATLAB

Algoritma *training* dan *testing* dilakukan dalam bentuk program yang berisikan kode yang dapat melakukan simulasi *Neural Network* pada MATLAB. Kode akan terlampir di bagian akhir dari laporan.

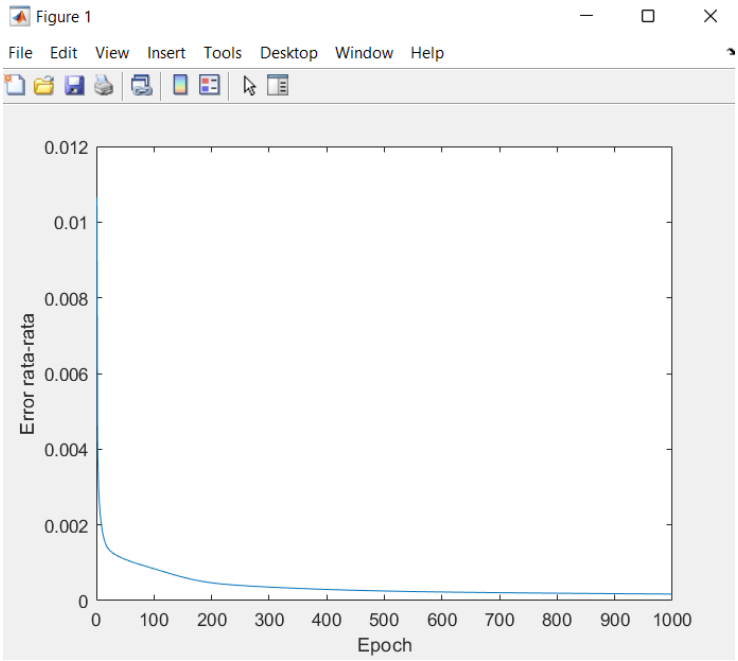
## 3.3. Hasil Pelatihan dan Pengujian

Kode MATLAB akan melakukan simulasi jaringan saraf tiruan. Data masukan dan keluaran untuk pelatihan akan digunakan untuk membentuk jaringan saraf tiruan kemudian dilakukan pengujian dengan data masukan dan keluaran untuk pelatihan. Hasil yang diamati berupa *mean squared error* (MSE) saat pengujian dengan tambahan grafik plot error terakhir per epoch saat pelatihan.

### 3.3.1. Dataset 1: Data random non-linier

#### 3.3.1.1. Data non-linear dengan metode training backpropagation stage 1 dan testing stage 1

Saat pelatihan jaringan saraf tiruan sebanyak 1000 epoch, didapatkan hasil sebagai berikut:

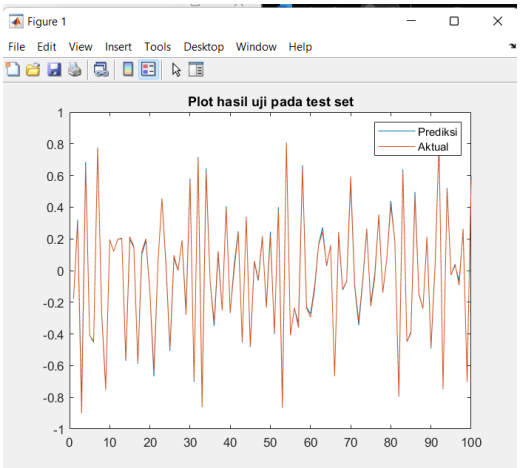



```
Epoch 900:
Error rata-rata: 1.824308e-04
Epoch 950:
Error rata-rata: 1.769330e-04
Epoch 1000:
Error rata-rata: 1.716616e-04
Proses training selesai!
```

*Gambar 3.3.1.1.1 Hasil plot error saat training pada dataset 1*

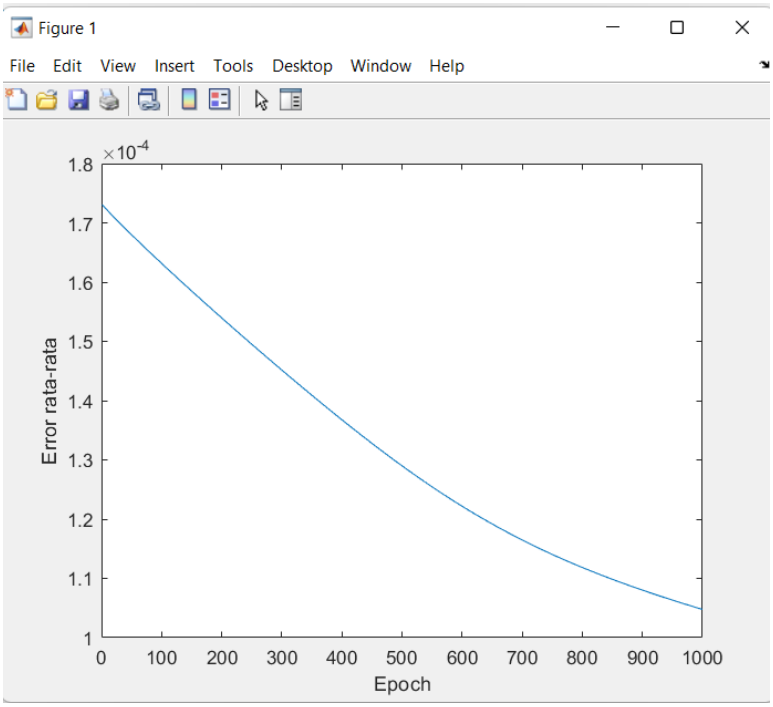
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
    - MSE: 0.00017




*Gambar hasil pengujian MSE pada dataset*

3.3.1.2. Data non-linear dengan metode training backpropagation stage 2 dan testing stage 2




*Gambar 3.3.1 Hasil plot error saat training pada dataset 1*

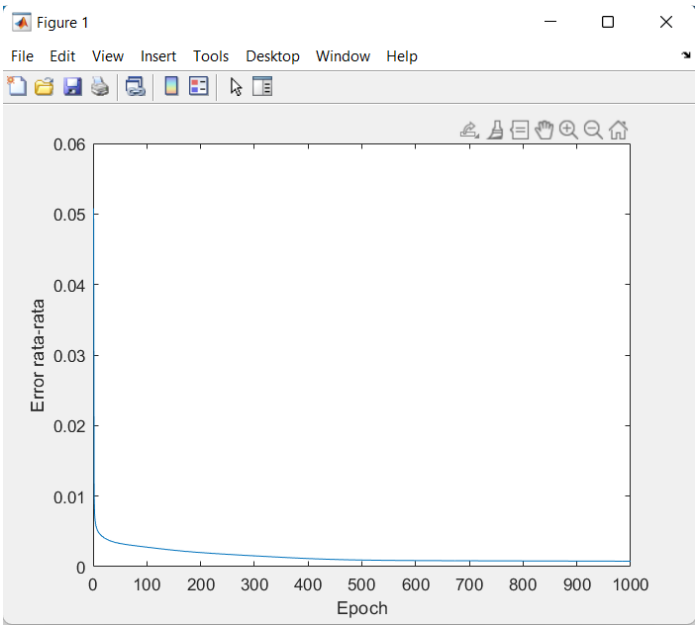
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
    - MSE: 0.00011

```
>> test_stage2_nonlinear('data1')
Pengujian terhadap test set
    MSE: 0.00011
```

*Gambar 3.2 Hasil pengujian MSE pada dataset*

3.3.1.3. Data non-linear dengan metode training inverse stage 1 dan testing inverse stage 1




*Gambar hasil plot error saat training pada dataset*

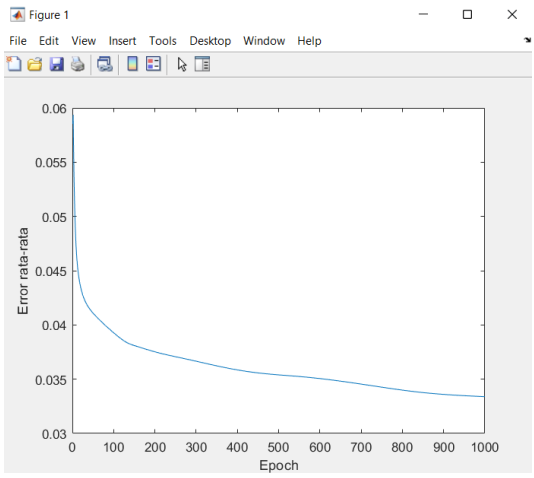
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.00083

```
>> test_inverse_nonlinear('data1')
Pengujian terhadap test set
    MSE: 0.00083
```

*Gambar hasil pengujian MSE pada dataset*

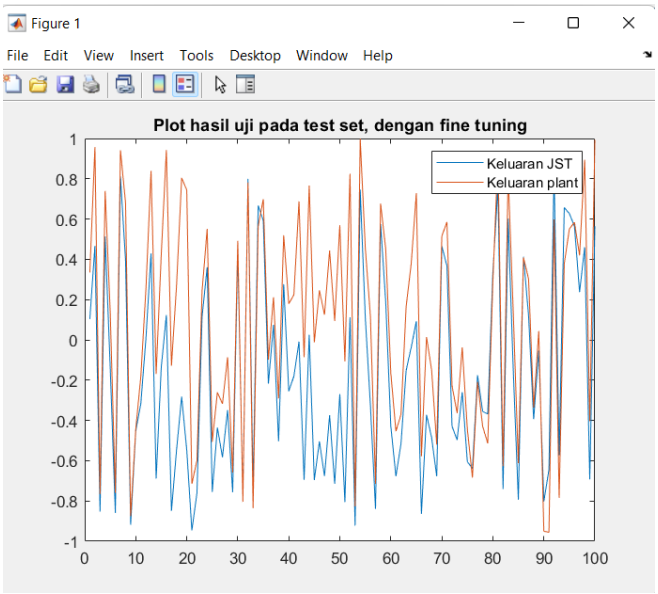
3.3.1.4. Data non-linear dengan metode training inverse stage 2 dan testing inverse stage 2




*Gambar hasil plot error saat training pada dataset*

Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.06551

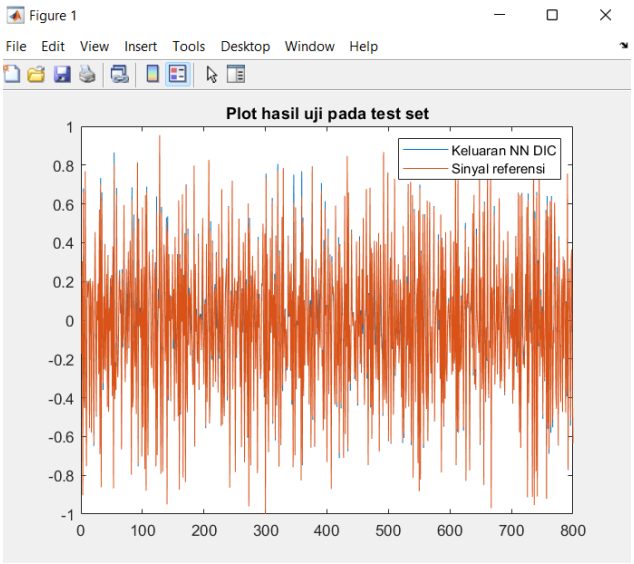



*Gambar hasil pengujian MSE pada dataset*

3.3.1.5. Data non-linear dengan Direct Inverse Control testing pada bagian pertama

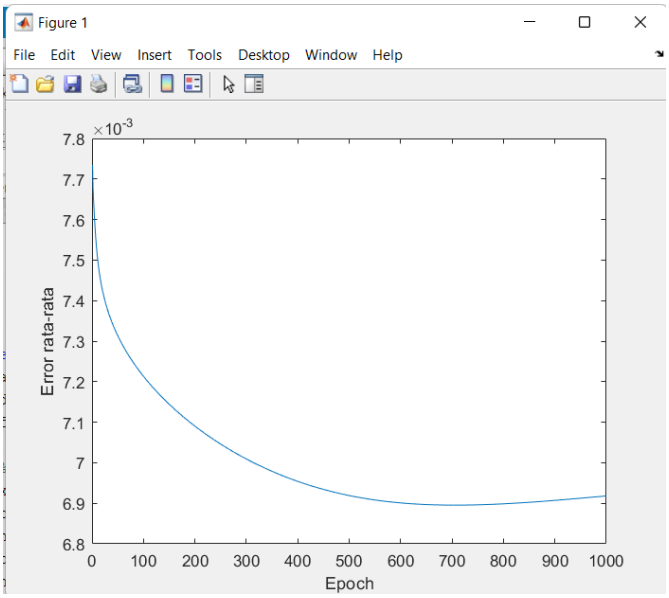
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.00415



*Gambar hasil pengujian MSE pada dataset*

3.3.1.6. Data non-linear dengan Direct Inverse Control training

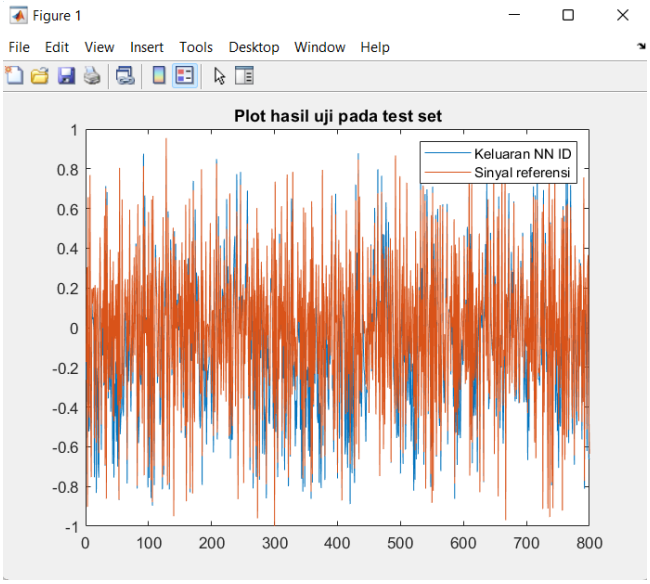



*Gambar hasil plot error saat training pada dataset*

3.3.1.7. Data non-linear dengan Direct Inverse Control testing hasil retrain

Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

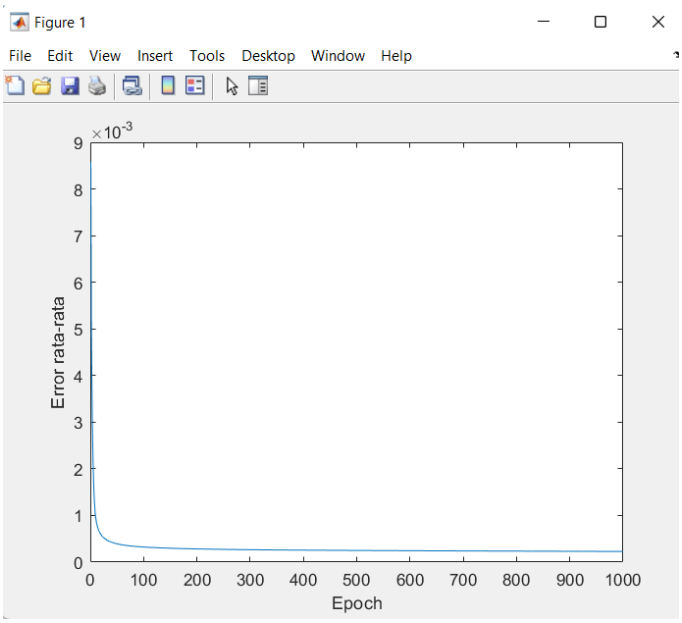
- Terhadap test set
  - MSE: 0.01299




*Gambar hasil pengujian MSE pada dataset*

### 3.3.2. Dataset 2: Data random linier

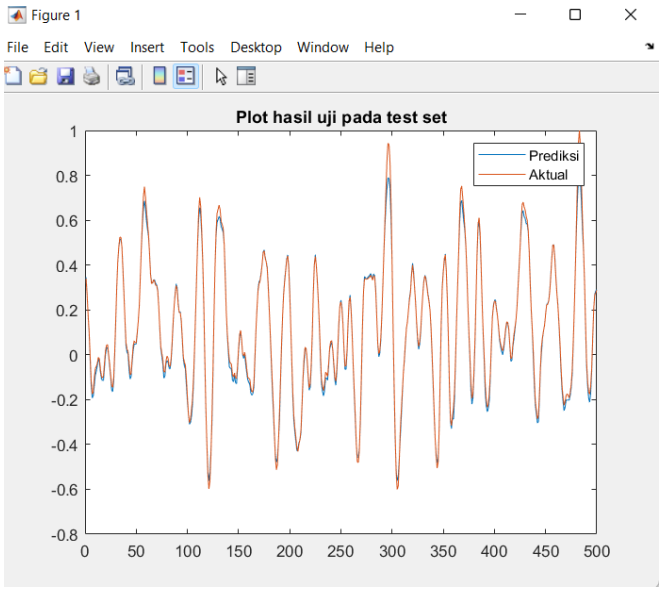
#### 3.3.2.1. Data linear dengan metode training backpropagation stage 1 dan testing stage 1




*Gambar hasil plot error saat training pada dataset*

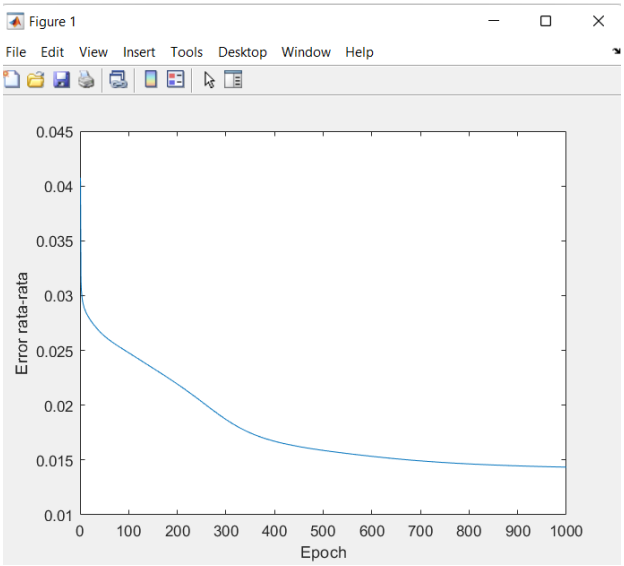
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.00024




*Gambar hasil pengujian MSE pada dataset*

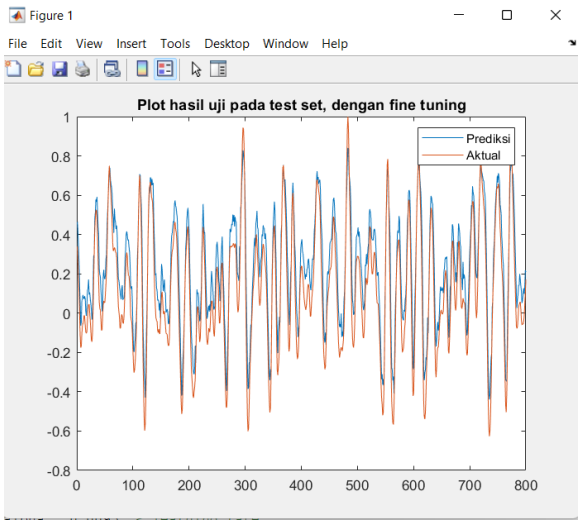
3.3.2.2. Data linear dengan metode training backpropagation stage 2 dan testing stage 2




*Gambar hasil plot error saat training pada dataset*

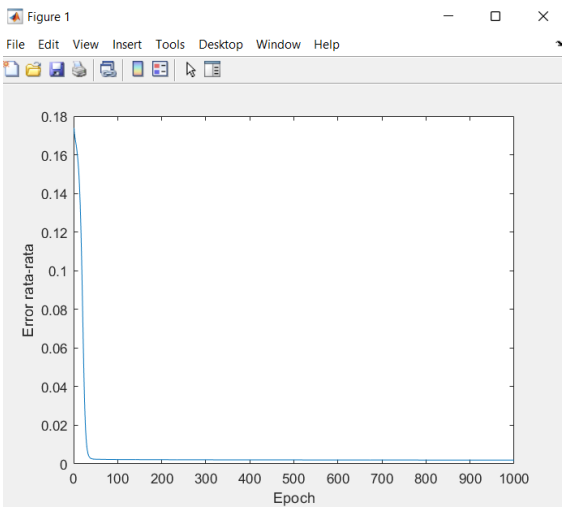
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
    - MSE: 0.00871




*Gambar hasil pengujian MSE pada dataset*

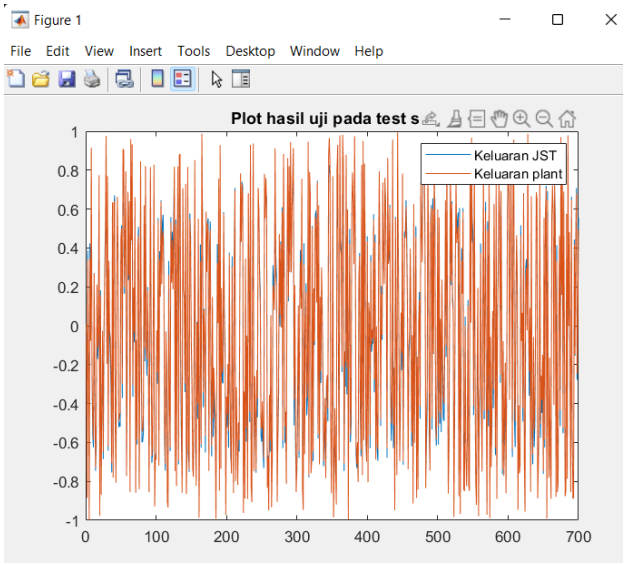
3.3.2.3. Data linear dengan metode training inverse stage 1 dan testing inverse stage 1




*Gambar hasil plot error saat training pada dataset*

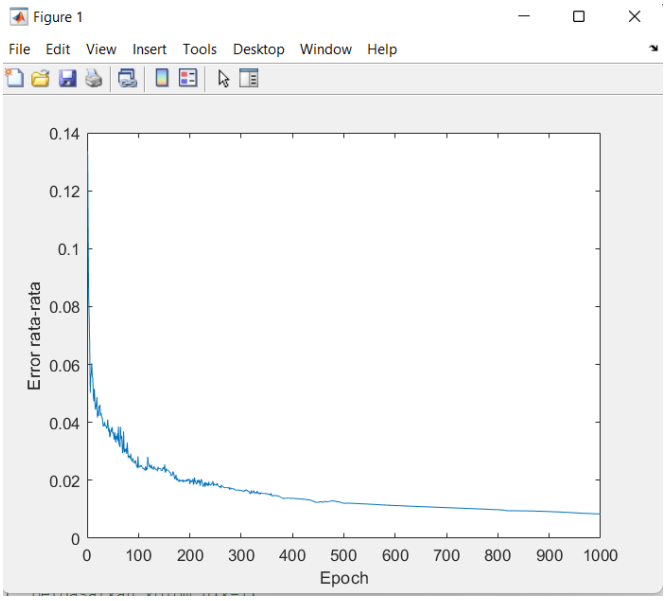
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
    - MSE: 0.00234




*Gambar hasil pengujian MSE pada dataset*

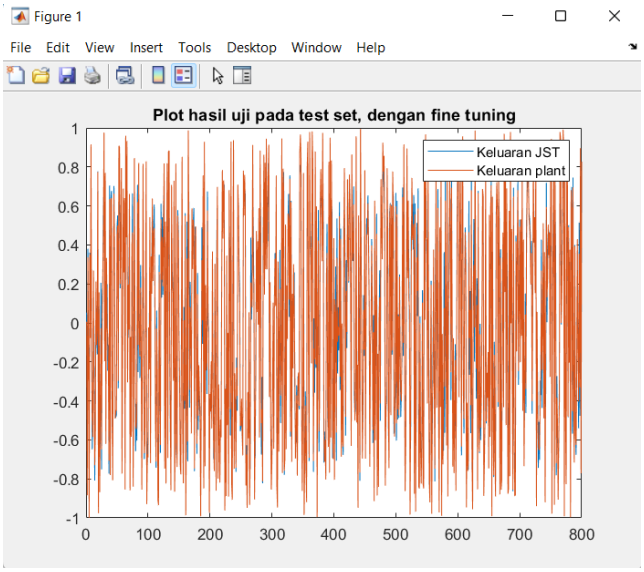
3.3.2.4. Data linear dengan metode training inverse stage 2 dan testing inverse stage 2




*Gambar hasil plot error saat training pada dataset*

Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.00614



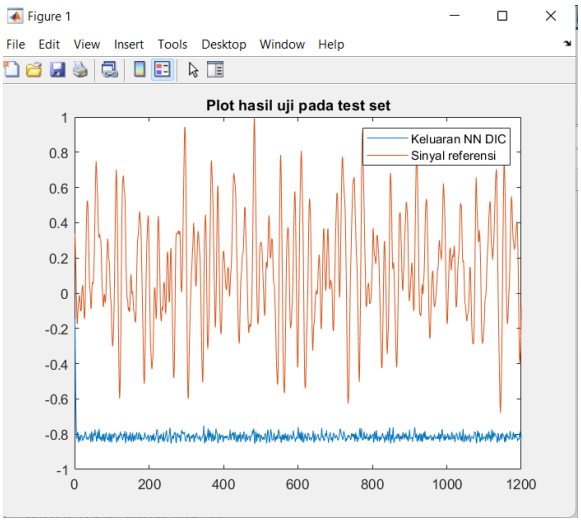

*Gambar hasil pengujian MSE pada dataset*

3.3.2.5. Data linear dengan Direct Inverse Control testing pada bagian pertama

Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

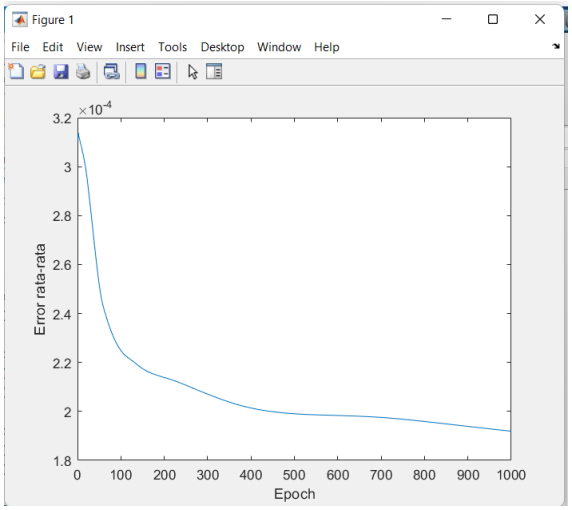
- Terhadap test set

- MSE: 0.49491




*Gambar hasil pengujian MSE pada dataset*

3.3.2.6. Data linear dengan Direct Inverse Control training

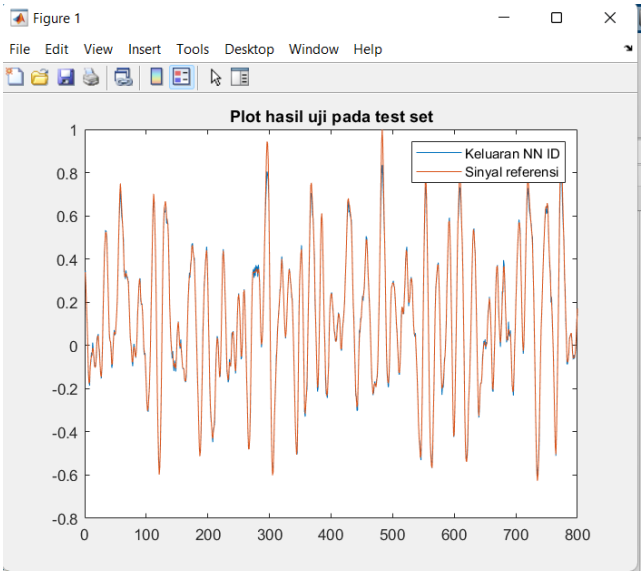



*Gambar hasil plot error saat training pada dataset*

3.3.2.7. Data linear dengan Direct Inverse Control testing hasil retrain

Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

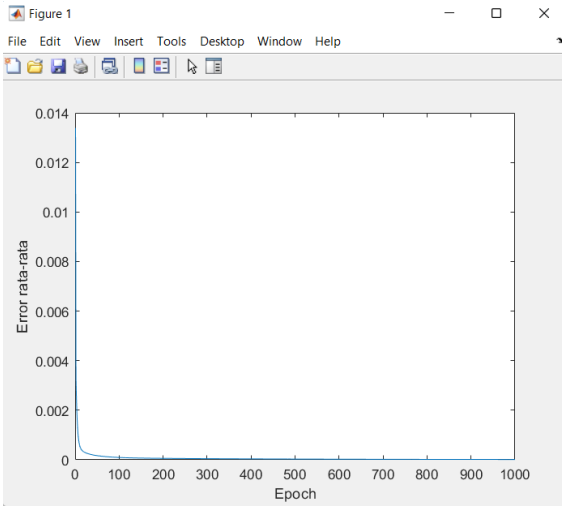
- Terhadap test set
  - MSE: 0.00017




*Gambar hasil pengujian MSE pada dataset*

### 3.3.3. Dataset 3: Data sine non-linier

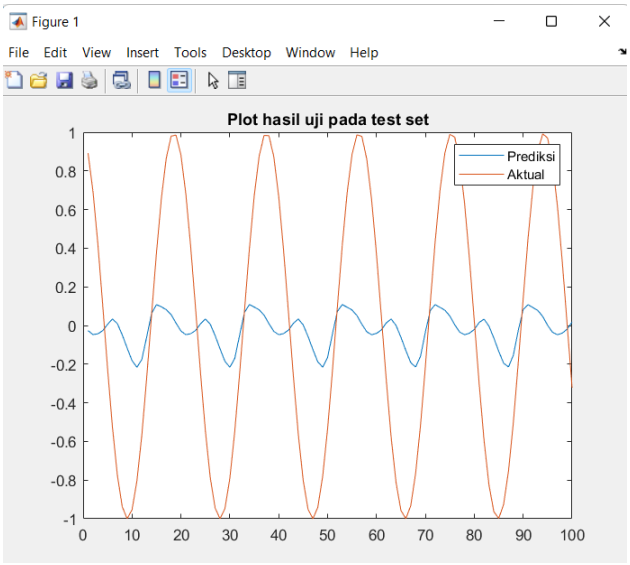
3.3.3.1. Data non-linear dengan metode training backpropagation stage 1 dan testing stage 1




*Gambar hasil plot error saat training pada dataset*

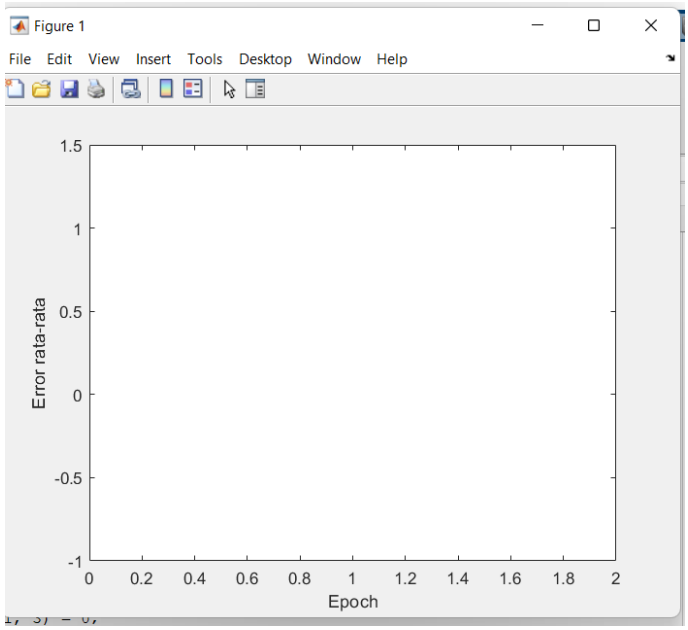
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
    - MSE: 0.21306




*Gambar hasil pengujian MSE pada dataset*

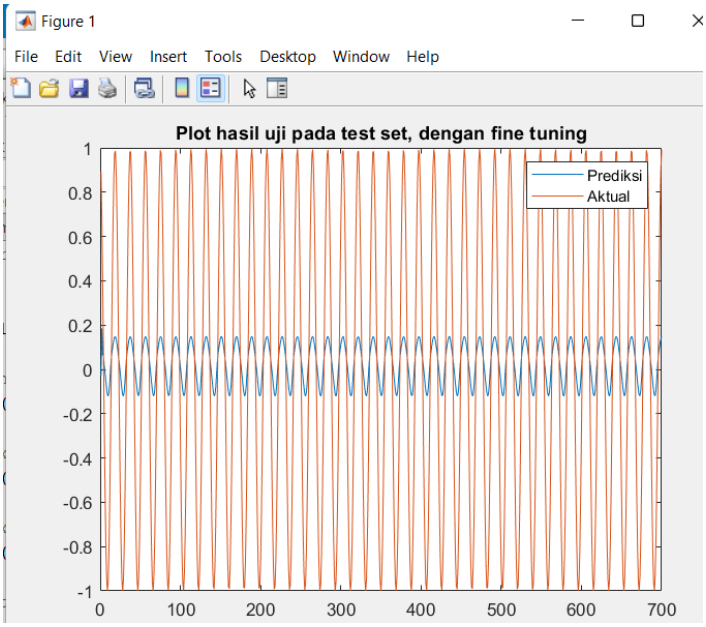
3.3.3.2. Data non-linear dengan metode training backpropagation stage 2 dan testing stage 2




*Gambar hasil plot error saat training pada dataset*

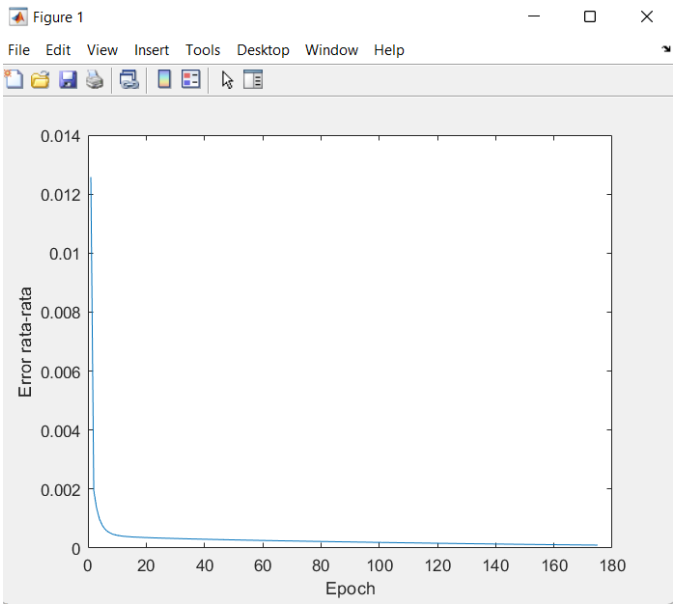
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.18969




*Gambar hasil pengujian MSE pada dataset*

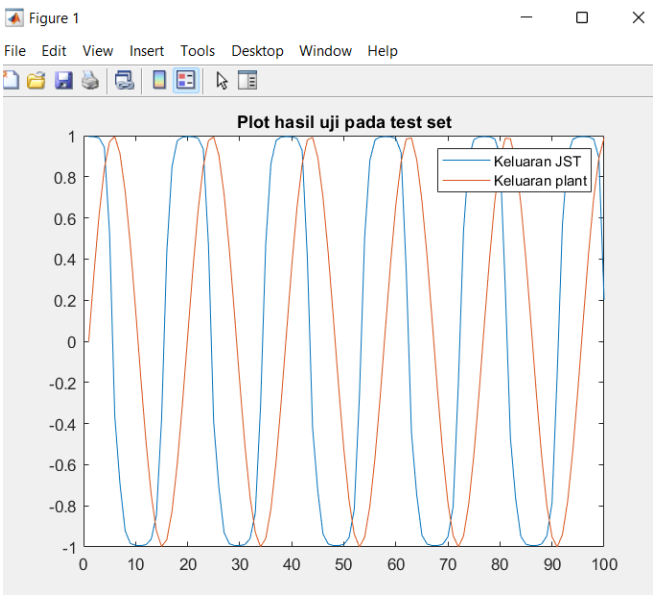
3.3.3.3. Data non-linear dengan metode training inverse stage 1 dan testing inverse stage 1




*Gambar hasil plot error saat training pada dataset*

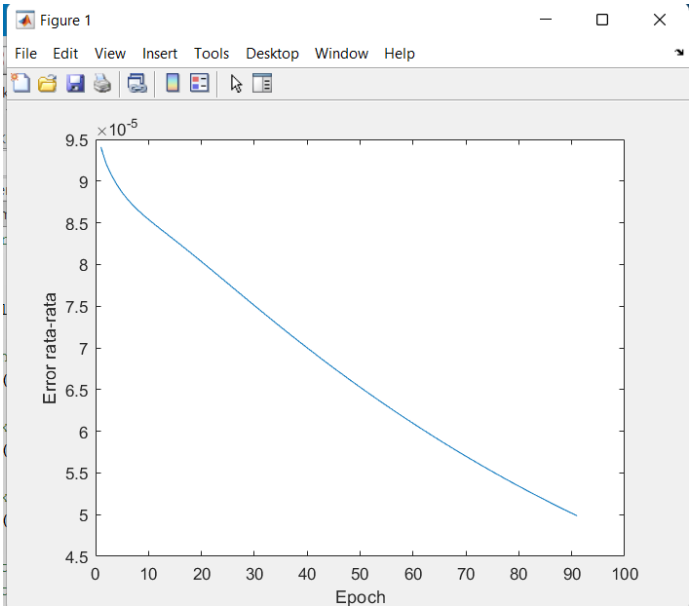
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.59491




*Gambar hasil pengujian MSE pada dataset*

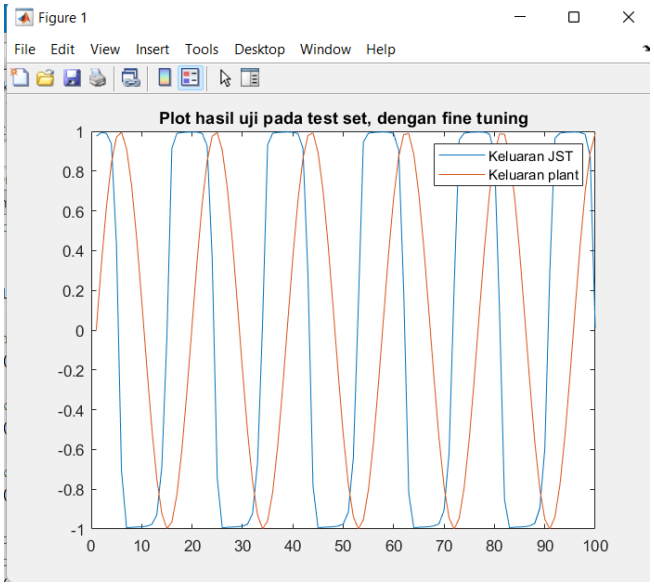
3.3.3.4. Data non-linear dengan metode training inverse stage 2 dan testing inverse stage 2




*Gambar hasil plot error saat training pada dataset*

Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.73328

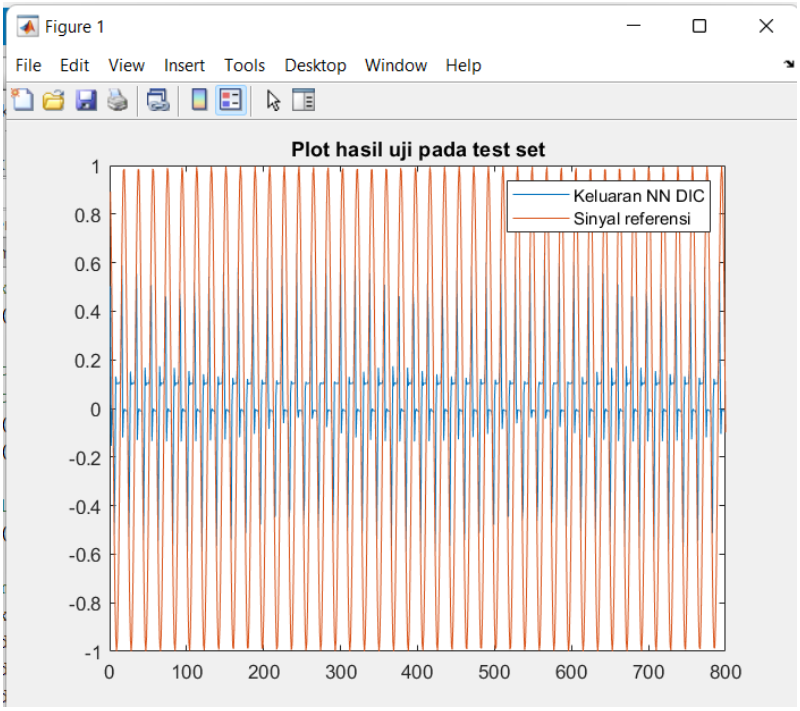



*Gambar hasil pengujian MSE pada dataset*

3.3.3.5. Data non-linear dengan Direct Inverse Control testing pada bagian pertama

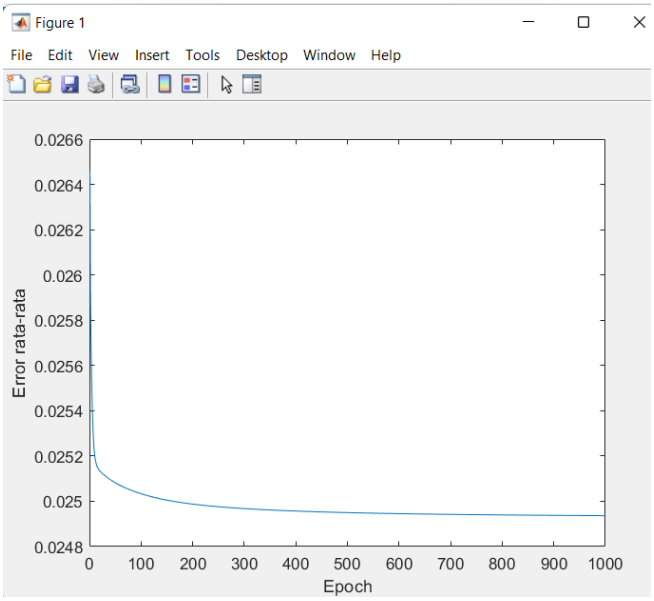
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.25207




*Gambar hasil pengujian MSE pada dataset*

3.3.3.6. Data non-linear dengan Direct Inverse Control training

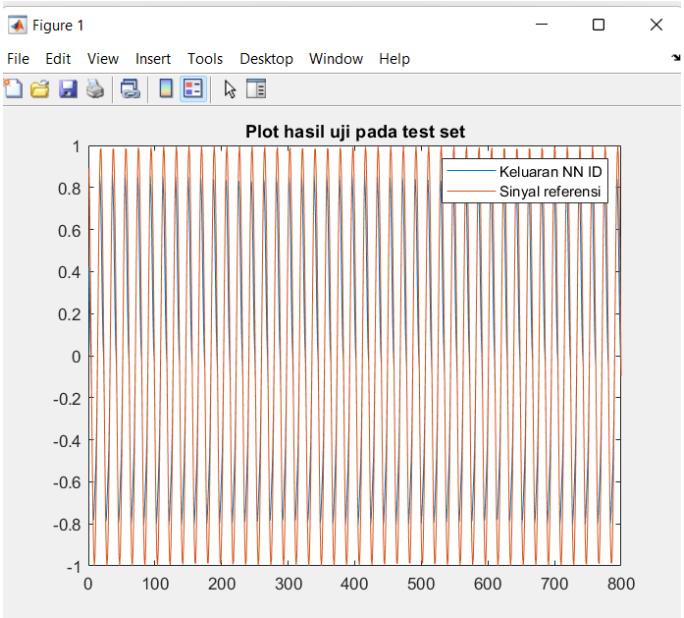



*Gambar hasil plot error saat training pada dataset*

3.3.3.7. Data non-linear dengan Direct Inverse Control testing hasil retrain

Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

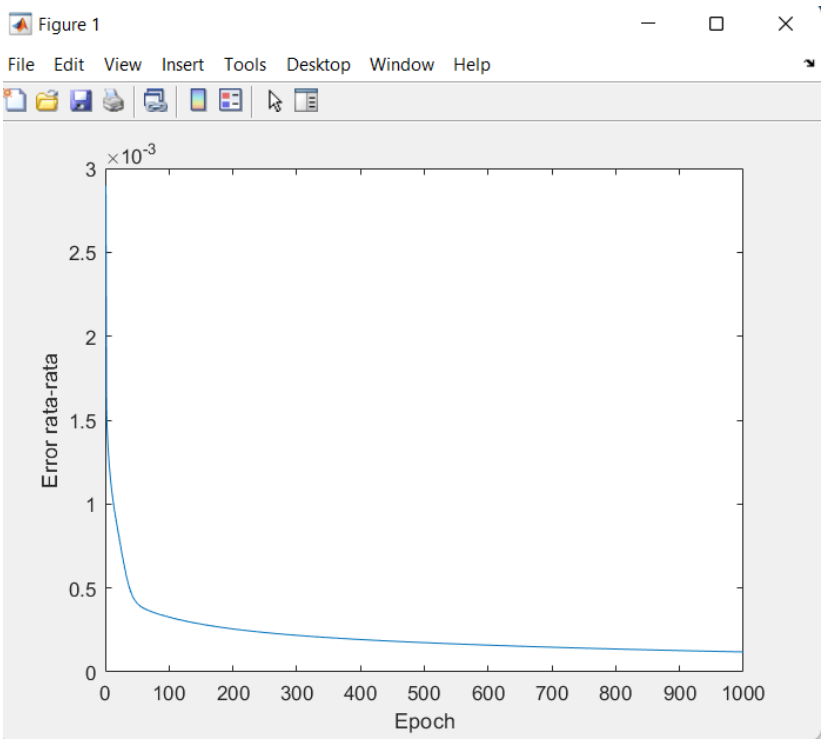
- Terhadap test set
  - MSE: 0.02508




*Gambar hasil pengujian MSE pada dataset*

### 3.3.4. Dataset 4: Data sine linier

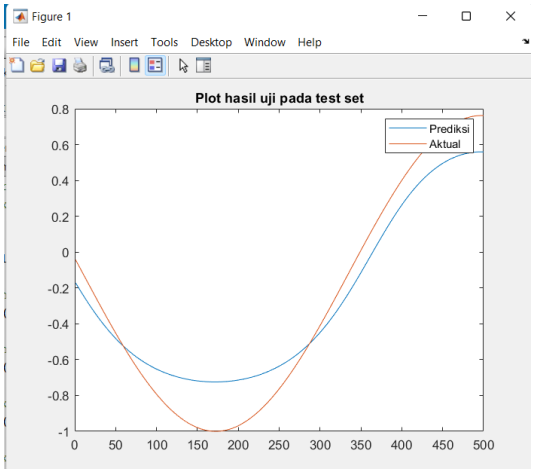
#### 3.3.4.1. Data linear dengan metode training backpropagation stage 1 dan testing stage 1




*Gambar hasil plot error saat training pada dataset*

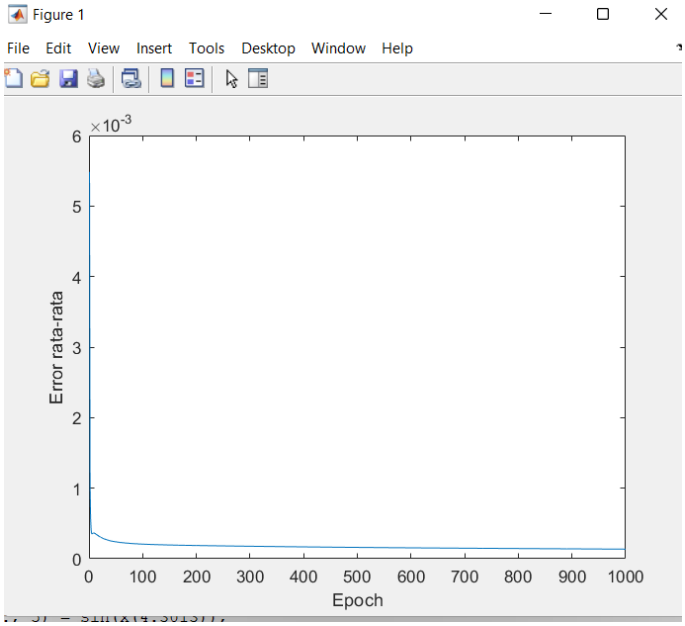
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.01370




*Gambar hasil pengujian MSE pada dataset*

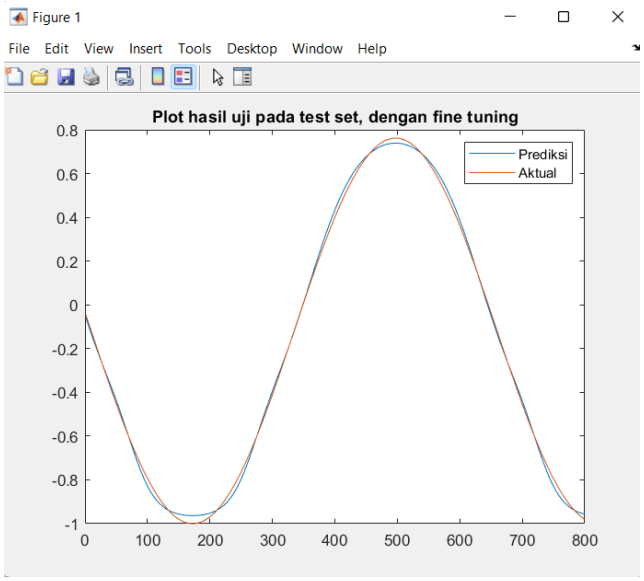
3.3.4.2. Data linear dengan metode training backpropagation stage 2 dan testing stage 2




*Gambar hasil plot error saat training pada dataset*

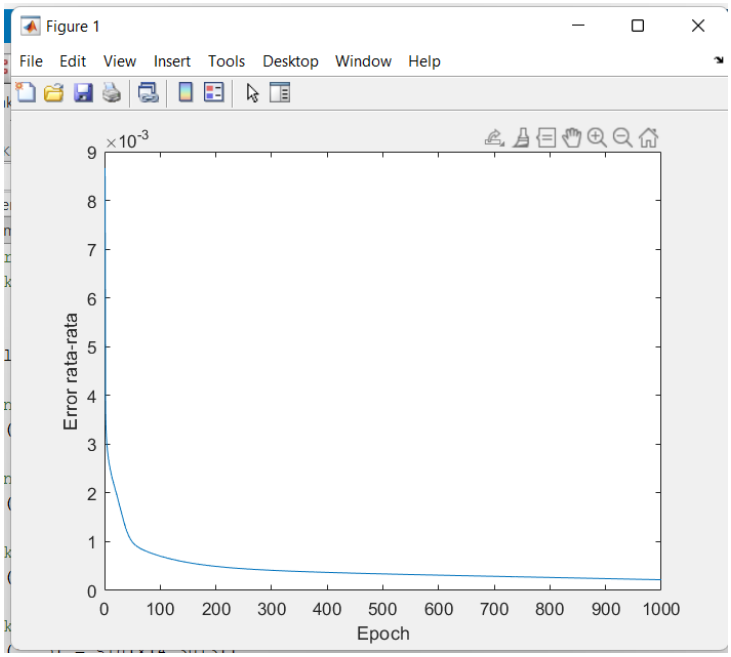
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.00021




*Gambar hasil pengujian MSE pada dataset*

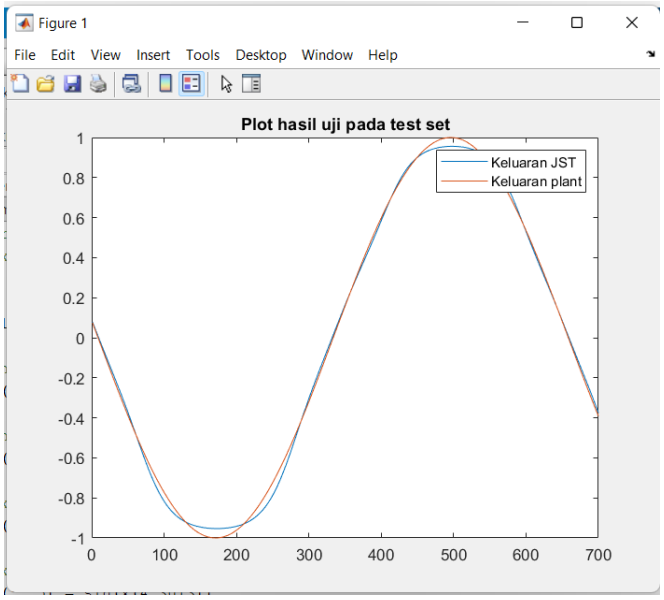
3.3.4.3. Data linear dengan metode training inverse stage 1 dan testing inverse stage 1




*Gambar hasil plot error saat training pada dataset*

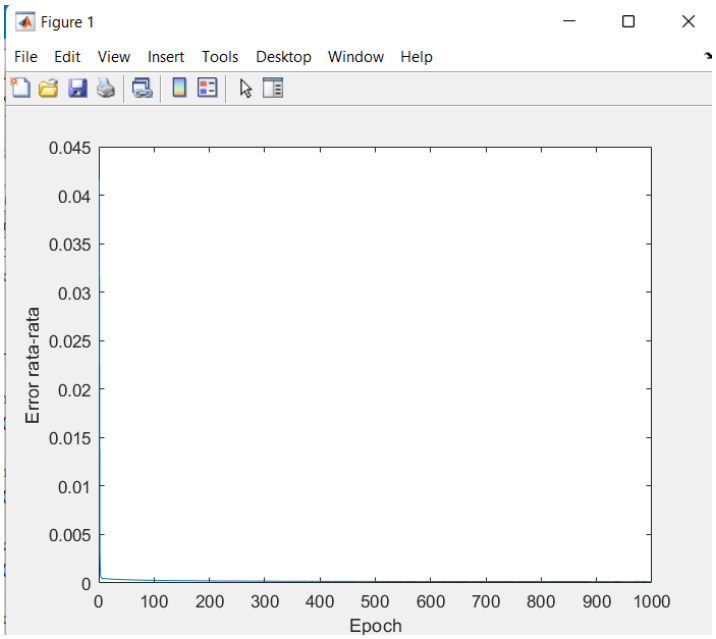
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.00037




*Gambar hasil pengujian MSE pada dataset*

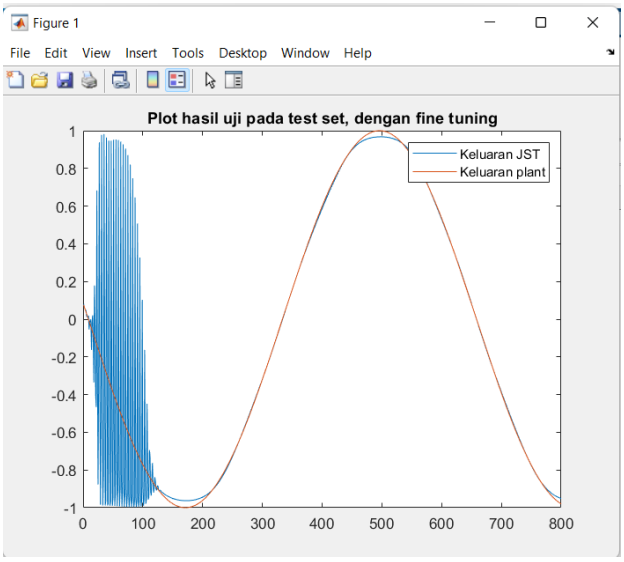
3.3.4.4. Data linear dengan metode training inverse stage 2 dan testing inverse stage 2




*Gambar hasil plot error saat training pada dataset*

Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.01495

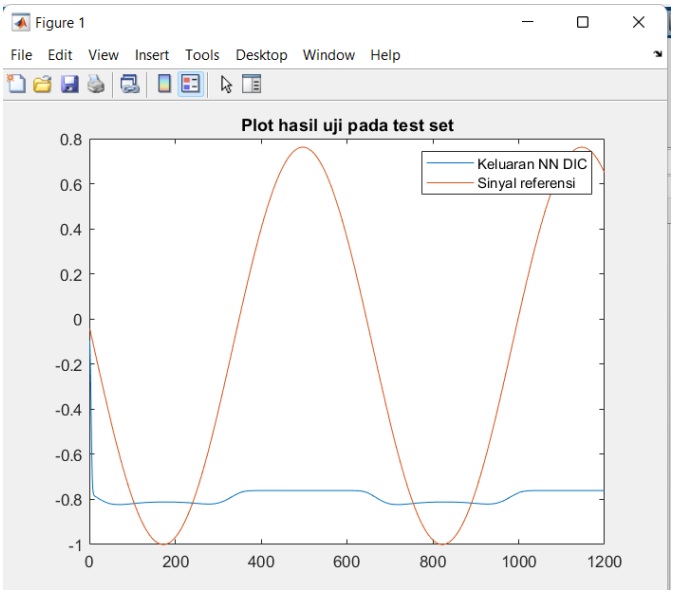



*Gambar hasil pengujian MSE pada dataset*

3.3.4.5. Data linear dengan Direct Inverse Control testing pada bagian pertama

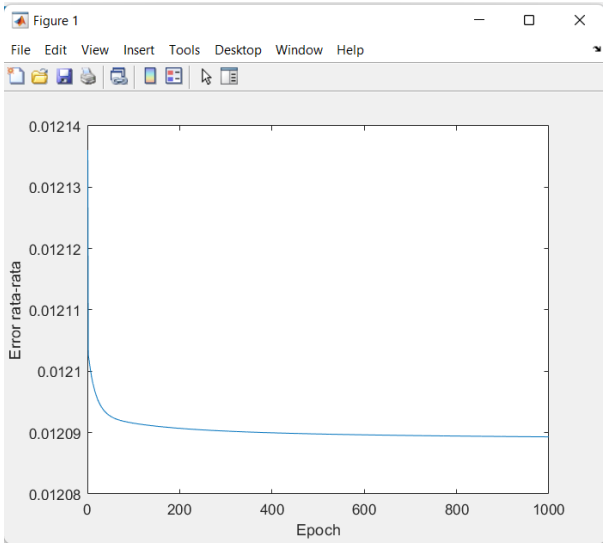
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.35501




*Gambar hasil pengujian MSE pada dataset*

3.3.4.6. Data linear dengan Direct Inverse Control training

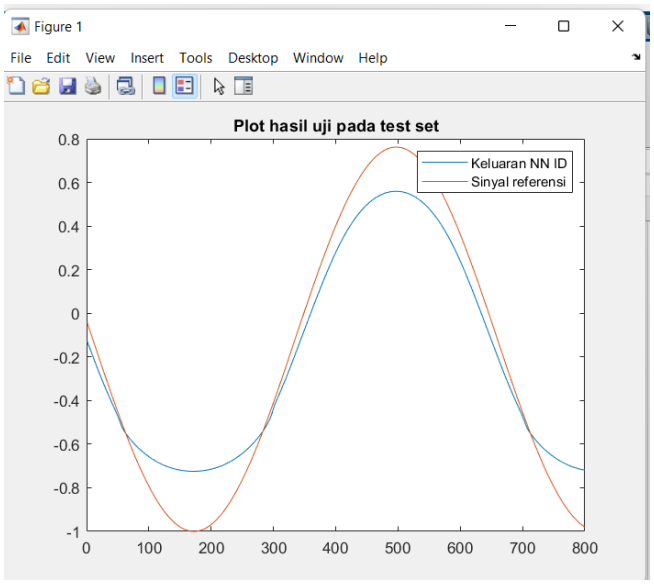



*Gambar hasil plot error saat training pada dataset*

3.3.4.7. Data linear dengan Direct Inverse Control testing hasil retrain

Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

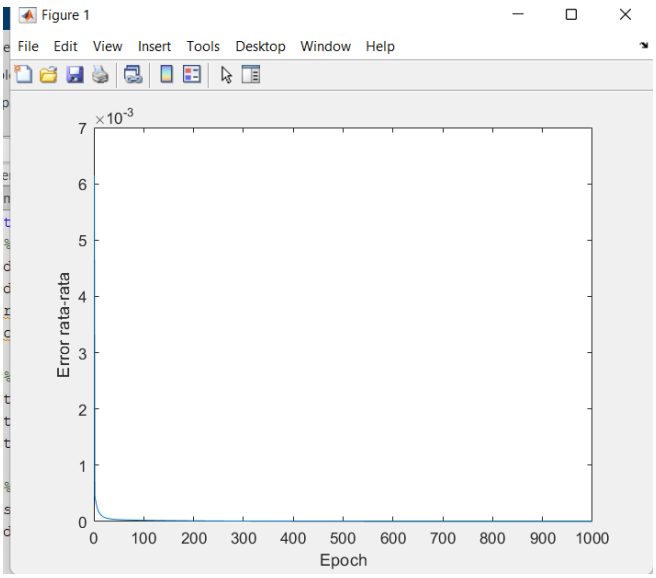
- Terhadap test set
  - MSE: 0.01245




*Gambar hasil pengujian MSE pada dataset*

### 3.3.5. Dataset 5: Data step non-linear

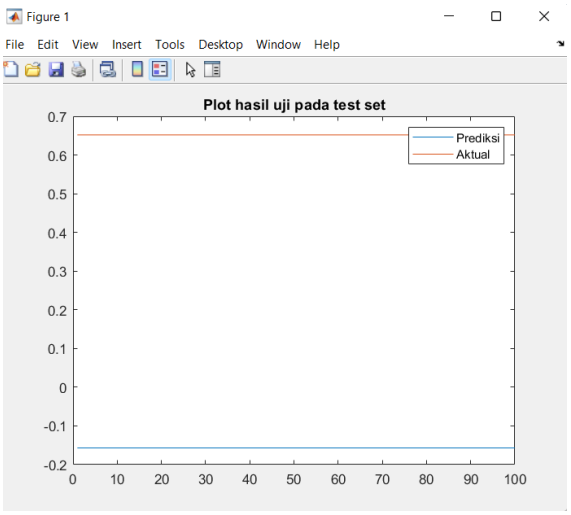
3.3.5.1. Data non-linear dengan metode training backpropagation stage 1 dan testing stage 1




*Gambar hasil plot error saat training pada dataset*

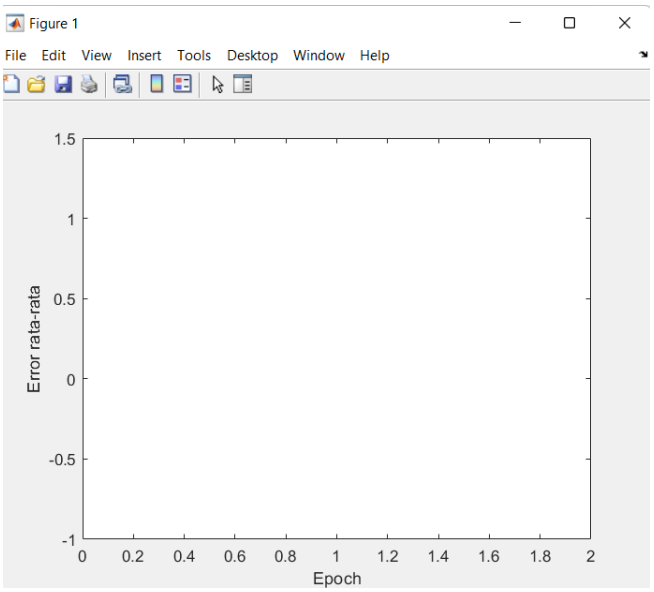
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
    - MSE: 0.32691




*Gambar hasil pengujian MSE pada dataset*

3.3.5.2. Data non-linear dengan metode training backpropagation stage 2 dan testing stage 2

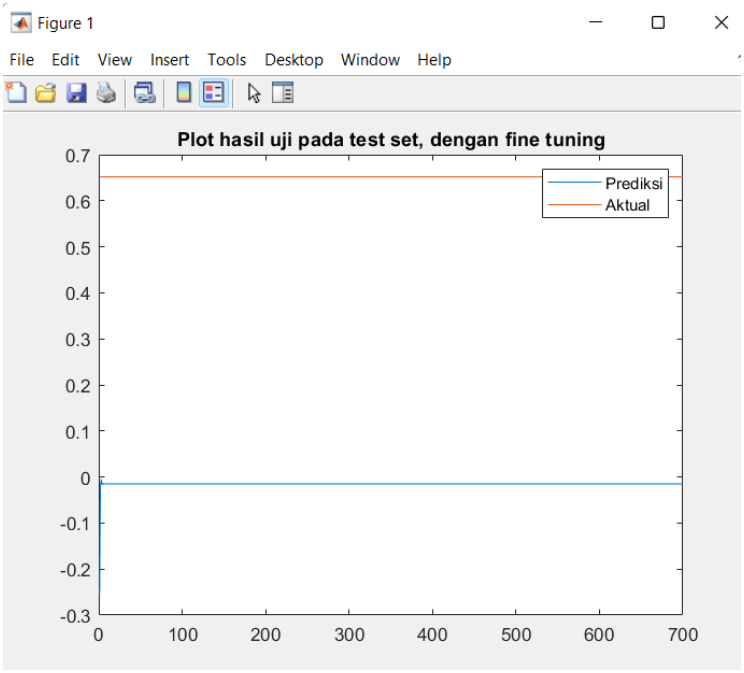



*Gambar hasil plot error saat training pada dataset*

Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

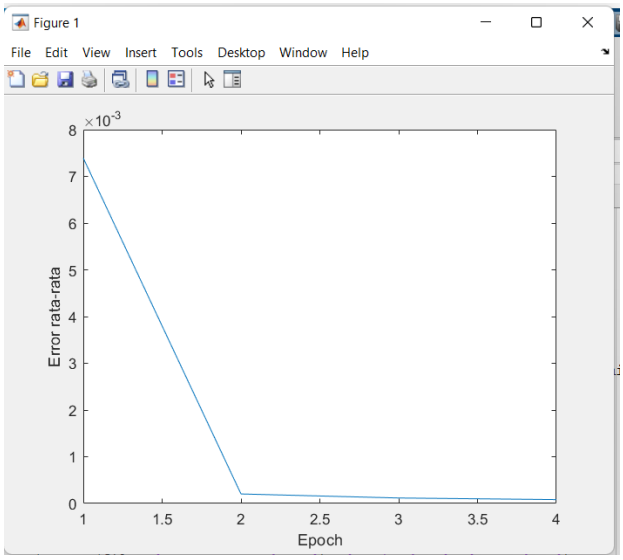
- Terhadap test set

- MSE: 0.22233




*Gambar hasil pengujian MSE pada dataset*

3.3.5.3. Data non-linear dengan metode training inverse stage 1 dan testing inverse stage 1

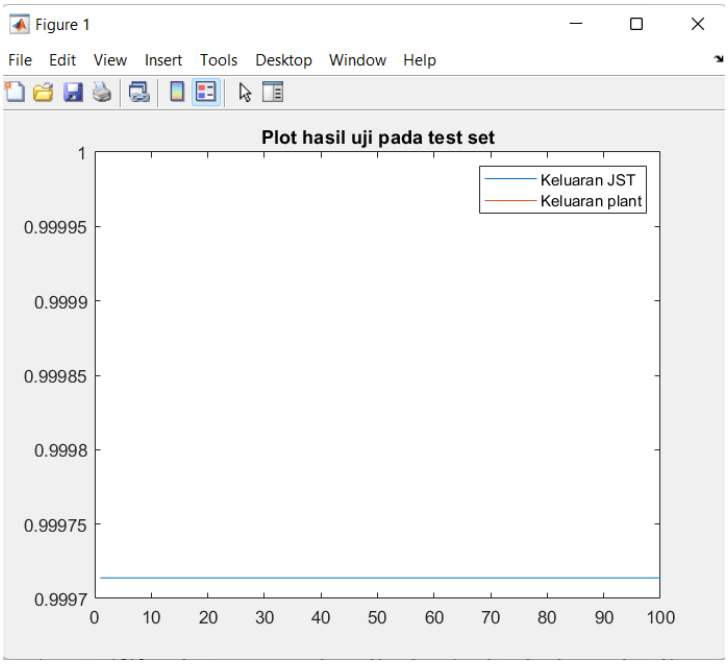



*Gambar hasil plot error saat training pada dataset*

Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

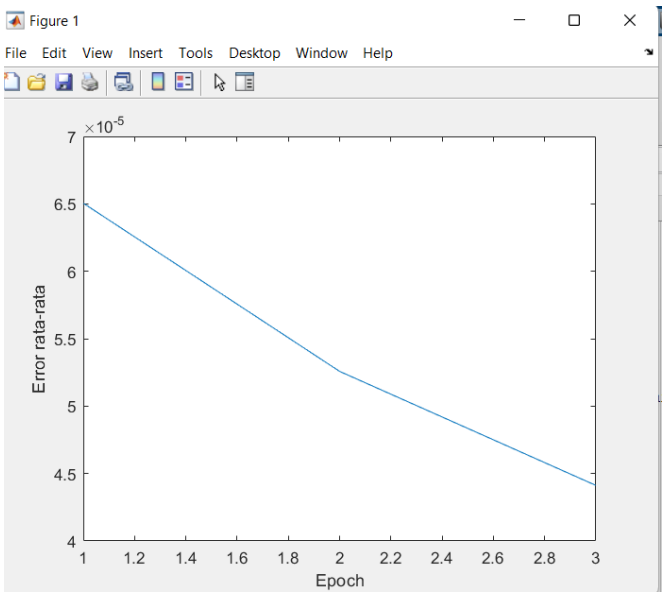
- Terhadap test set

- MSE: 0.0000




*Gambar hasil pengujian MSE pada dataset*

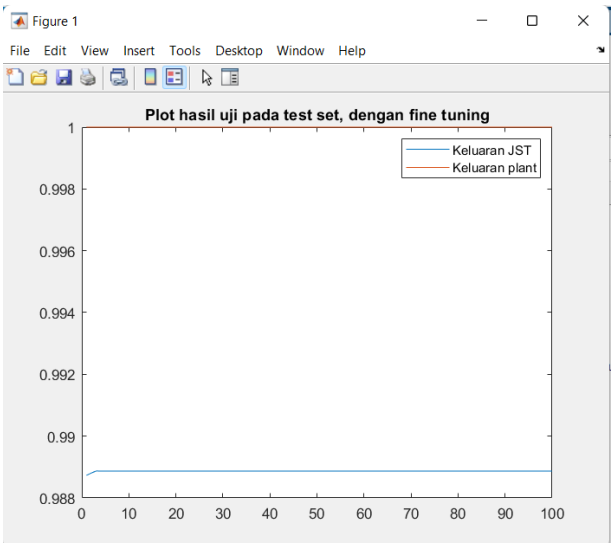
3.3.5.4. Data non-linear dengan metode training inverse stage 2 dan testing inverse stage 2




*Gambar hasil plot error saat training pada dataset*

Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.00006

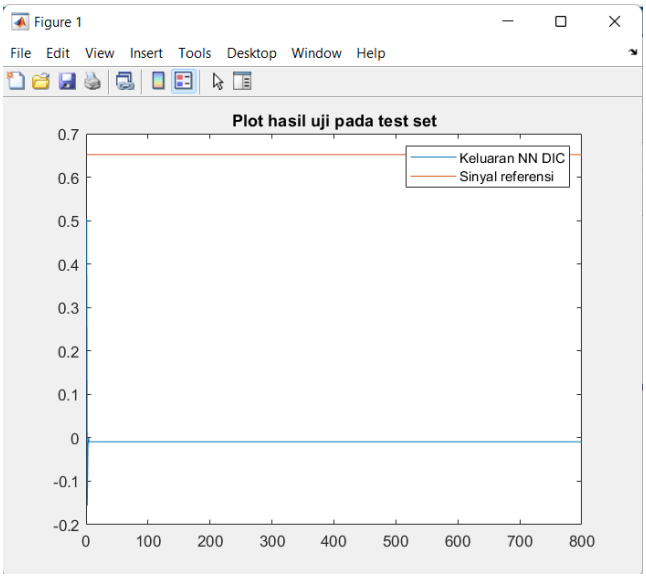



*Gambar hasil pengujian MSE pada dataset*

3.3.5.5. Data non-linear dengan Direct Inverse Control testing pada bagian pertama

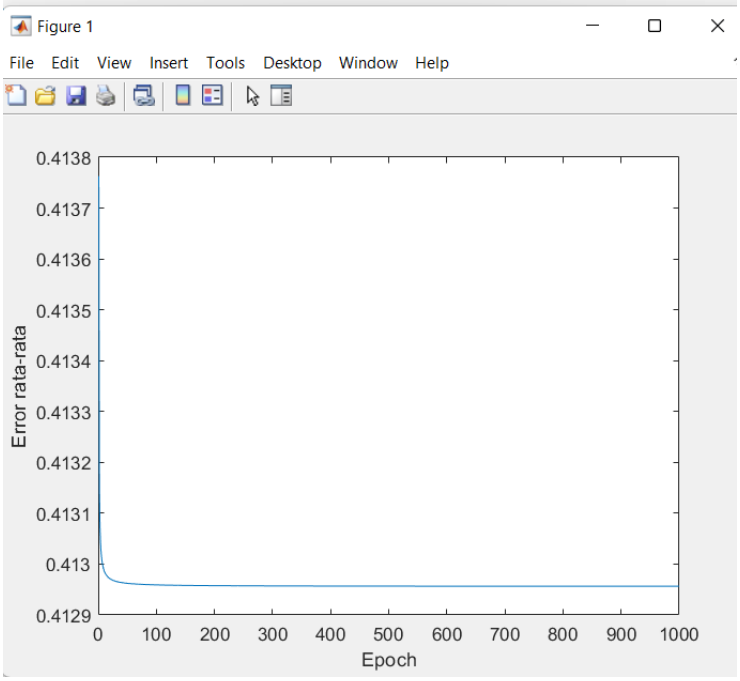
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.21842




*Gambar hasil pengujian MSE pada dataset*

3.3.5.6. Data non-linear dengan Direct Inverse Control training

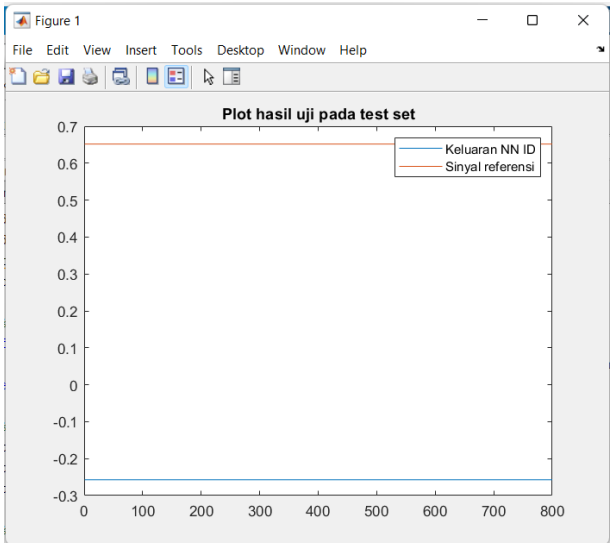



*Gambar hasil plot error saat training pada dataset*

3.3.5.7. Data non-linear dengan Direct Inverse Control testing hasil retrain

Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

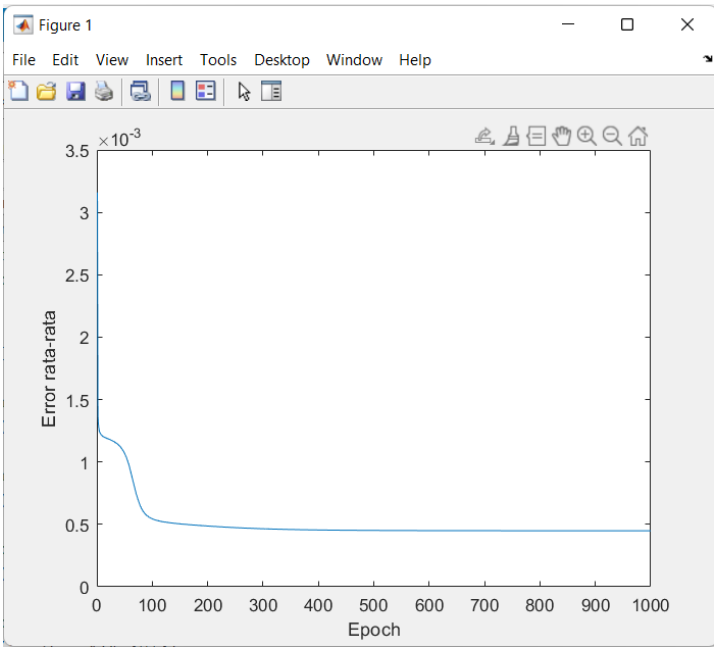
- Terhadap test set
  - MSE: 0.41318




*Gambar hasil pengujian MSE pada dataset*

### 3.3.6. Dataset 6: Data step linier

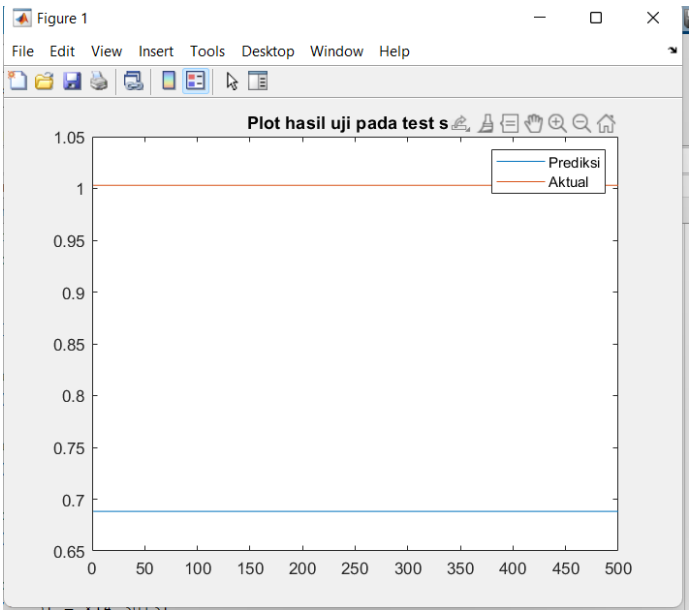
3.3.6.1. Data linear dengan metode training backpropagation stage 1 dan testing stage 1




*Gambar hasil plot error saat training pada dataset*

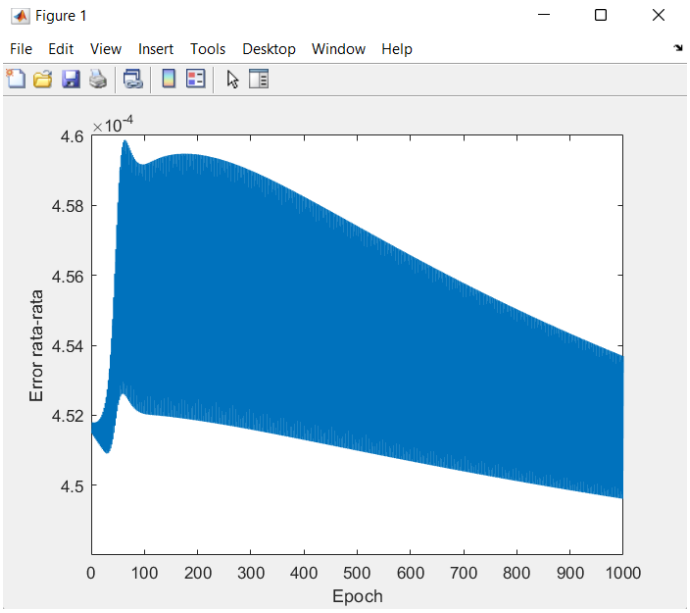
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.04955



*Gambar hasil pengujian MSE pada dataset*

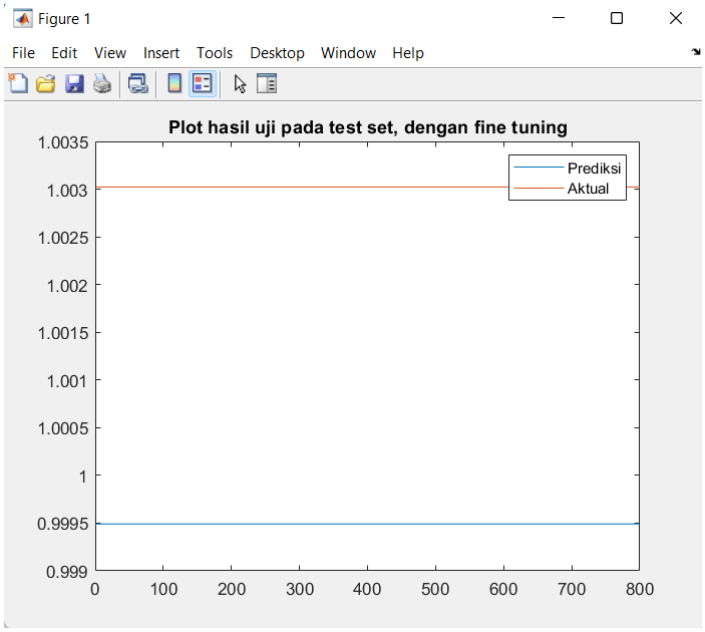
3.3.6.2. Data linear dengan metode training backpropagation stage 2 dan testing stage 2




*Gambar hasil plot error saat training pada dataset*

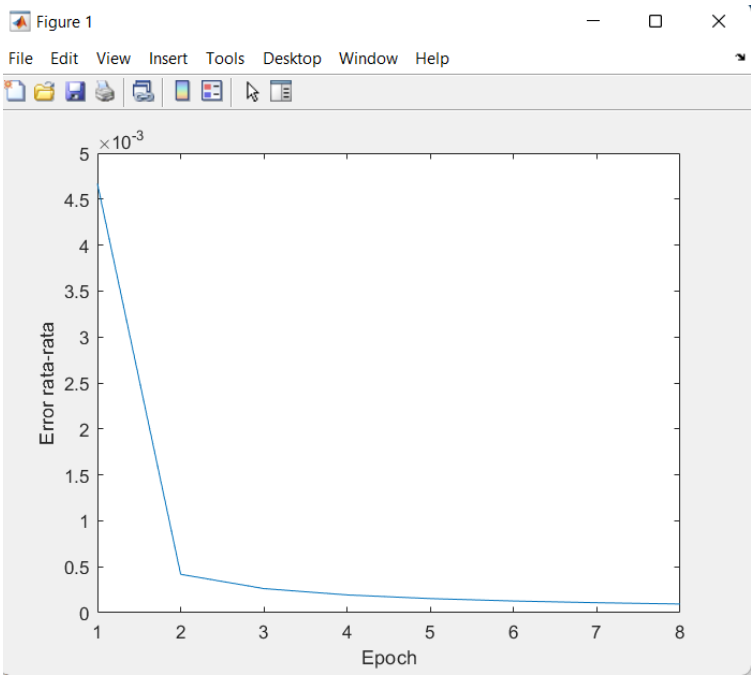
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.00001




*Gambar hasil pengujian MSE pada dataset*

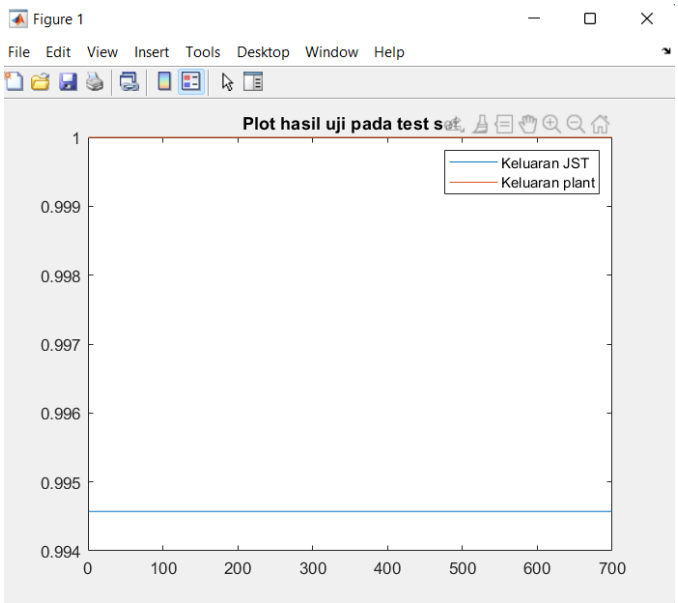
3.3.6.3. Data linear dengan metode training inverse stage 1 dan testing inverse stage 1




*Gambar hasil plot error saat training pada dataset*

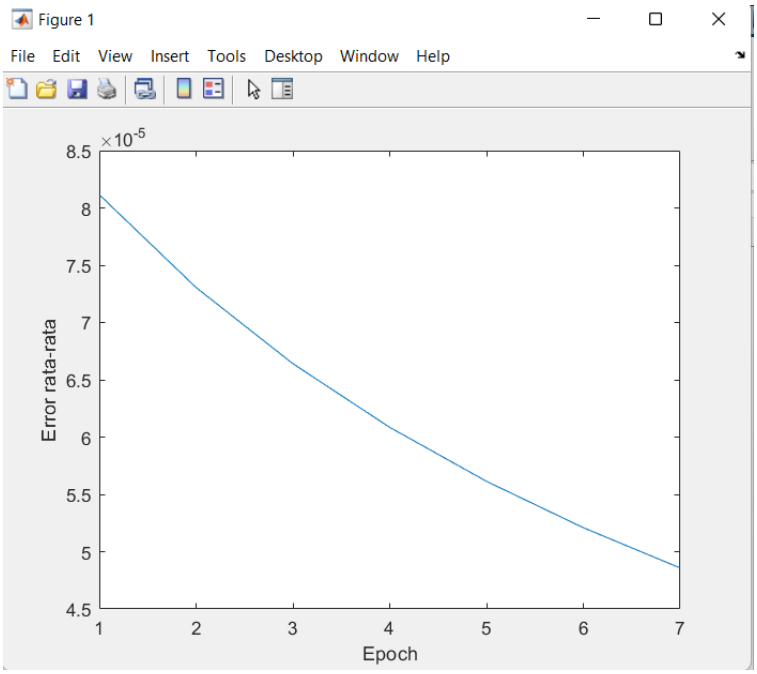
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.00001




*Gambar hasil pengujian MSE pada dataset*

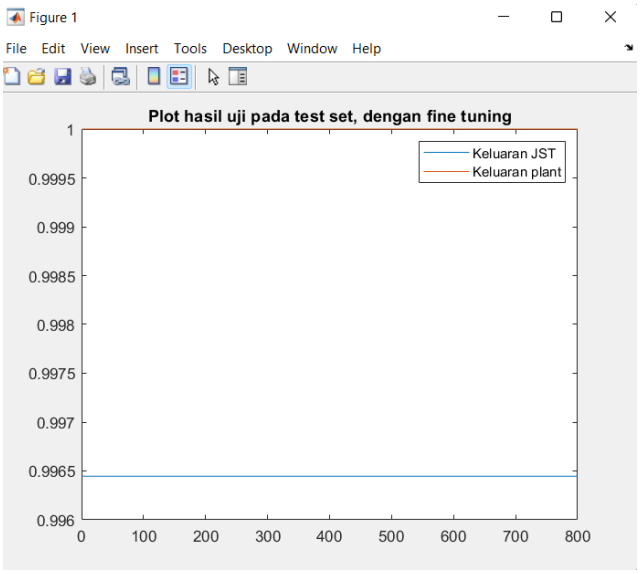
3.3.6.4. Data linear dengan metode training inverse stage 2 dan testing inverse stage 2




*Gambar hasil plot error saat training pada dataset*

Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.00001

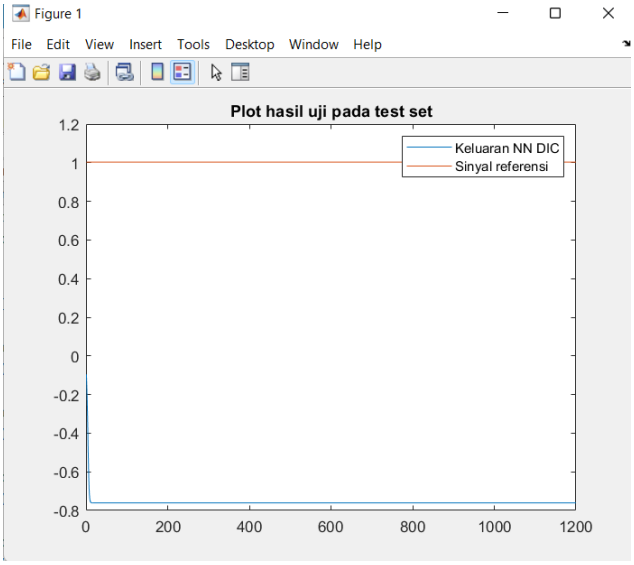



*Gambar hasil pengujian MSE pada dataset*

3.3.6.5. Data linear dengan Direct Inverse Control testing pada bagian pertama

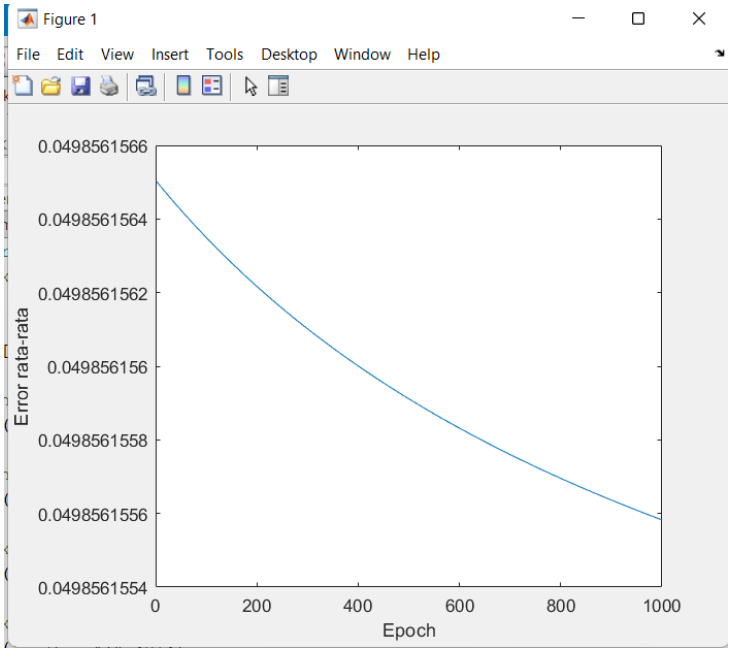
Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 1.55259




*Gambar hasil pengujian MSE pada dataset*

3.3.6.6. Data linear dengan Direct Inverse Control training

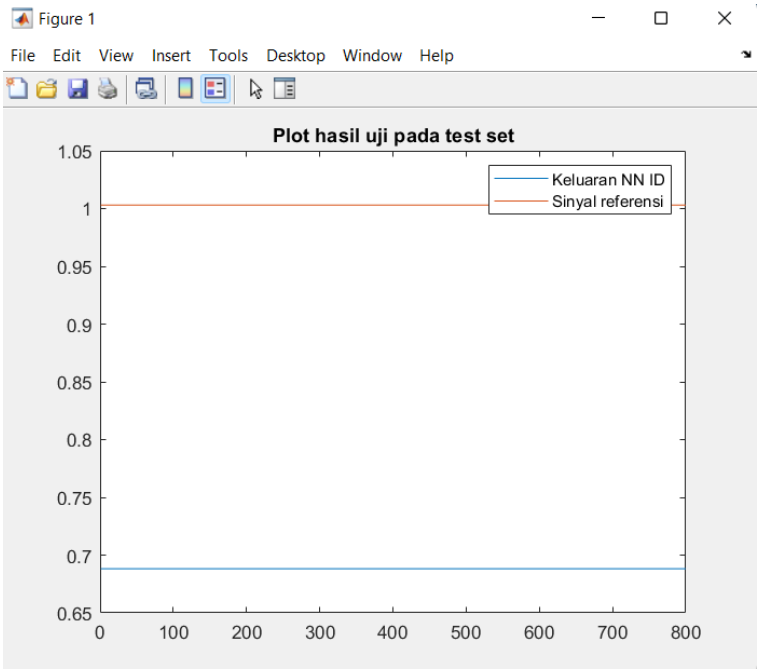



*Gambar hasil plot error saat training pada dataset*

3.3.6.7. Data linear dengan Direct Inverse Control testing hasil retrain

Saat pengujian jaringan saraf tiruan, didapatkan hasil sebagai berikut:

- Terhadap test set
  - MSE: 0.04955




*Gambar hasil pengujian MSE pada dataset*

## 3.4. Analisa Hasil

### 3.4.1 Bryan

mu = 0.65

alpha = 0.0095

hidden neuron = 16

epoch = 1000

Dari pengamatan hasil pengujian dan testing dari 6 jenis dataset yaitu Data random linear, data random nonlinear, data sin linear, data sin nonlinear, data step linear, dan data step nonlinear. Dapat dilihat bahwa dijalankan dengan beberapa metode yang berbeda mulai dari JST sampai dengan metode Direct Inverse Control (DIC). Pada dasaset pertama yaitu data random nonlinear dengan metode JST biasa tanpa fine tuning, metode training dan error MSE yang didapatkan terbilang sudah cukup kecil dan bagus, Dengan adanya fine-tuning, nilai MSE semakin mengecil sehingga fine-tuning cocok diaplikasikan ke dalam dataset random yang nonlinear. Namun apabila menggunakan metode Inverse dan DIC, dapat terlihat bahwa sepertinya kurang cocok diimplementasikan pada dataset dengan fine-tuning maupun tanpa fine-tuning, dimana terlihat MSEnya naik cukup jauh dibandingkan dengan metode JST biasa.

Lalu dari dataset kedua yaitu data random linear dapat dilihat pada metode JST biasa didapatkan hasil yang cukup baik dan sesuai dengan ekspektasi. Namun justru dengan adanya fine-tuning, nilai dari MSE justru akan semakin kecil pada data random linear dan tidak sesuai dengan ekspektasi. Lalu selain itu, penerapan JST dengan metode inverse juga justru akan memperburuk prediksi dari JST. Terdapat 1 metode yang dirasa cukup tepat dalam menangani data random linear ialah dengan metode DIC yang sudah di retrain sehingga didapatkan MSE yang lebih baik dari metode backpropagation.

Lalu pada dataset setelahnya yaitu dataset sin yang nonlinear, dapat terlihat bahwa dengan metode JST backpropagation, nilai MSE yang dihasilkan masih

belum cukup sesuai dengan ekspektasi. Namun dengan memakai fine tuning, MSE berubah menjadi jauh lebih kecil. Namun apabila masuk ke dalam metode sistem inverse maka nilai error per epoch juga nilai MSEnya akan semakin naik. Apabila dilihat secara langsung dari grafik, dapat terlihat bahwa bentuk prediksi sudah hampir menyamai bentuk dari keluaran aslinya, hanya saja seperti terdapat delay yang dialami salah satu pihal. Apabila diberikan metode Direct Inverse Control bentuk grafik mengalami perbaikan seperti halnya MSE. Dengan adanya retrain dan ditest kembali, prediksi sistem mengalami kemajuan pesat dimana MSE yang dihasilkan sangat kecil disbanding dengan sebelumnya.

Lalu pada dataset sin linear, pengaplikasian JST terlihat masih kurang dengan memakai backpropagation sehingga diperlukan finetuning yang membuat MSE jauh lebih baik. Selain itu, dengan memakai metode sistem inverse juga akan membuat hasil MSE lebih hasil hanya saja dengan fine tuning pada inverse terjadi abnormalitas prediksi dari JST dibandingkan dengan yang normal. Dengan memakai metode DIC, terlihat bahwa MSE kurang baik namun dari sisi grafik mengalami perbaikan setelah mengalami retrain pada metode DICnya.

Secara keseluruhan JST pada input step ini, masih terbilang cukup baik dengan segala metode. namun pada input step nonlinear, terlihat bahwa metode terbaik ialah dengan menggunakan metode invers tanpa adanya fine-tuning yang menghasilkan MSE yang bernilai 0 yang artinya betul-betul akurat sempurna. Begitu pula pada dataset step yang linear memiliki kecendrungan menghasilkan MSE yang sangat baik. Hal ini terlihat pada metode backprop biasa dengan fine tuning, metode invers dan invers dengan fine-tuning.

Secara keseluruhan, pembuatan program JST sudah dapat terbilang cukup baik dimana masing-masing input data memiliki metode masing-masing yang cocok dengan karakteristiknya,

### 3.4.2 Adrian

Parameter:

Identifikasi sistem:

Pelatihan biasa:

- Jumlah neuron tersembunyi: 32 buah
- Laju pembelajaran $\alpha$: 0.01
- Konstanta momentum $\mu$: 0.7
- Epoch maksimal 1000 kali atau error minimal $10^{-7}$

Pelatihan ulang:

- Jumlah neuron tersembunyi: 32 buah
- Laju pembelajaran $\alpha$: 0.005
- Konstanta momentum $\mu$: 0.5
- Epoch maksimal 1000 kali atau error minimal $10^{-4}$

Inversi:

Pelatihan biasa:

- Jumlah neuron tersembunyi: 32 buah
- Laju pembelajaran $\alpha$: 0.01
- Konstanta momentum $\mu$: 0.7
- Epoch maksimal 1000 kali atau error minimal $10^{-4}$

Pelatihan ulang:

- Jumlah neuron tersembunyi: 32 buah
- Laju pembelajaran $\alpha$: 0.005
- Konstanta momentum $\mu$: 0.5
- Epoch maksimal 1000 kali atau error minimal $5 \times 10^{-5}$

Pelatihan ulang DIC:

- Jumlah neuron tersembunyi: 32 buah
- Laju pembelajaran $\alpha$: 0.01
- Konstanta momentum $\mu$: 0.7
- Epoch maksimal 2000 kali atau error minimal $10^{-4}$

SNAPID:

$$\boldsymbol{w} = \begin{bmatrix} 5 \\ 0.1 \\ 0.9 \end{bmatrix}, \boldsymbol{\eta} = \begin{bmatrix} 2 \\ 0.01 \\ 0.06 \end{bmatrix}, \text{dan } K = 8.5$$

### 1.1 Analisis kinerja JST sebagai identifikasi sistem

Secara umum, kinerja JST untuk memodelkan *plant* cukup baik, baik *plant* nonlinear maupun linear. Hal ini terlihat dari JST yang konvergen dalam nilai MSE yang kecil dan dalam epoch yang terbilang kecil.

Dari hasil yang didapatkan, JST lebih baik dalam memodelkan *plant* nonlinear daripada linear, meskipun perbedaannya juga tidak signifikan. Hal ini disebabkan karena *plant* linear yang digunakan mempunyai suku yang lebih banyak daripada *plant* nonlinear, sehingga secara matematis lebih kompleks. Nonlinearitas dari *plant* pertama tidak begitu berpengaruh karena JST mampu untuk memodelkan sistem nonlinear.

Dengan *fine-tuning*, terutama untuk *plant* linear, hasilnya terlihat kurang baik dibandingkan sebelum dilakukan *tuning* untuk input acak. Akan tetapi, untuk input sinusoidal dan input step, setelah *tuning*, hasilnya justru membaik. Hal ini disebabkan karena JST sebelum *tuning* masih mempunyai ketidaksempurnaan meskipun sudah dilatih. Ketidaksempurnaan ini menghasilkan error pada keluaran yang dihasilkan.

### 1.2 Analisis kinerja JST sebagai sistem kendali berbasis inversi

Permasalahan utama dari sistem inversi ialah apakah konfigurasi inversi ini secara matematis eksis. Apabila beberapa input dapat menghasilkan output yang sama, maka inversi tidak memungkinkan untuk eksis, terlebih lagi direalisasikan.

Mengingat kembali, untuk *plant* nonlinear, persamaan yang digunakan adalah

$$y(k) = \frac{1}{1 + y(k-1)^2} + 0.25u(k) - 0.3u(k-1)$$

Apabila persamaan tersebut dimanipulasi sedemikian rupa sehingga dibentuk inversnya secara matematis, maka hal ini memungkinkan, yaitu seperti berikut ini.

$$u(k) = 4y(k) - \frac{4}{1 + y(k-1)^2} + 0.8333u(k-1)$$

Terlihat bahwa untuk *plant* nonlinear, fungsi invers eksis.

Sedangkan, untuk *plant* linear, persamaan yang digunakan adalah

$$y(k) = 0.04251u(k-1) + 0.04044u(k-2) + 1.778y(k-1) - 0.8607y(k-2)$$

Terlihat disini bahwa **tidak ada suku u(k)**. Sehingga, fungsi invers tidak eksis. Apabila "dipaksakan" untuk mengambil fungsi invers dari suku yang lain, misalnya seperti berikut.

$$u(k-1) = 23.5239y(k) - 41.82545y(k-1) + 20.247y(k-2) - 0.9513u(k-2)$$
$$\Leftrightarrow u(k) = 23.5239y(k+1) - 41.82545y(k) + 20.247y(k-1) - 0.9513u(k-1)$$

Maka, terlihat bahwa input u(k) saat tertentu bergantung pada output di masa datang, yang mana hal ini tidak memungkinkan. Dengan demikian, **fungsi invers untuk *plant* linear tidak eksis**.

Secara umum, JST sebagai sistem kendali berbasis inversi masih dicoba untuk direalisasikan. Akan tetapi, untuk *plant* linear, JST konvergen pada nilai error yang lebih tinggi daripada *plant* nonlinear. Hal ini dapat terjadi karena kesulitan JST untuk menemukan fungsi matematis yang ekuivalen dengan invers dari *plant*.

Oleh karena itu, hasil pengujian juga selaras dengan hasil pelatihan. Pengujian untuk *plant* nonlinear memberikan hasil yang lebih baik daripada *plant* linear.

Dalam proses *fine-tuning*, parameter pelatihan sengaja dipilih dengan laju pembelajaran dan konstanta momentum yang lebih kecil agar pelatihan bisa berjalan lebih lambat, sehingga jaringan bisa menemukan pola atau fungsi matematis secara lebih akurat dari data dengan input acak yang diberikan, mengingat bahwa jaringan juga pada hakikatnya sudah terlatih. Bagaimanapun, dengan *fine-tuning*, secara umum, hasil pelatihan dan pengujian memberikan hasil yang kurang baik dibandingkan dengan sebelum dilakukan *tuning*. Hal ini dapat dijelaskan dengan alasan yang sama dengan kasus yang terjadi pada pengembangan JST sebagai identifikasi sistem, yaitu bahwa masih terdapat ketidaksempurnaan pada jaringan setelah dilakukan pelatihan biasa. Karena sistem inversi konvergen di nilai error yang lebih tinggi daripada JST untuk identifikasi sistem, ketidaksempurnaan pada JST sebagai sistem inversi juga lebih besar. Keluaran yang kurang baik tersebut diumpan balik ke input JST menghasilkan keluaran lain, sehingga dengan begitu, berpotensi akan menghasilkan keluaran yang kurang baik kembali.

### 1.3 Analisis kinerja JST sebagai sistem *direct inverse control*

Setelah masing-masing JST identifikasi sistem dan JST sistem inversi dilatih secara terpisah, kedua JST digabungkan sehingga membentuk sistem *direct inverse control*. Dengan adanya kesalahan kecil dari masing-masing JST, secara teoritis, hasil keluaran dari sistem *direct inverse control* akan menghasilkan kesalahan pula. Sehingga, dalam laporan ini, dilakukan pelatihan ulang pula untuk JST sistem inversi.

Seperti dipaparkan sebelumnya, untuk *plant* linear, sistem inversi tidak dapat dibentuk dengan baik, sehingga hasilnya juga tidak baik. Sedangkan, untuk *plant* nonlinear, sistem inversi bisa dibentuk. Sehingga, hasil pengujian terhadap sistem DIC untuk *plant* nonlinear terbilang baik.

Bagaimanapun, pelatihan ulang serta pengujian terhadap DIC yang sudah di-*tuning* justru memberikan hasil yang kurang baik untuk *plant* nonlinear. Sedangkan, untuk *plant* linear, hasilnya justru membaik dengan signifikan. Hasil yang kurang baik untuk *plant* nonlinear dapat disebabkan karena *plant* yang digunakan memetakan input acak menjadi output yang cenderung lebih sering berubah, dibandingkan dengan *plant* linear (dapat terlihat pada "Gambar 13. MSE hasil pengujian NN ID untuk plant nonlinear dan grafik hasil pengujian NN ID dengan plant nonlinear, prediksi vs aktual" dan pada "Gambar 16. MSE hasil pengujian NN ID untuk plant linear dan grafik hasil pengujian NN ID dengan plant linear, prediksi vs aktual"). Apabila keduanya dibandingkan, terlihat untuk input acak yang sama, output pada *plant* nonlinear lebih rapat (lebih sering berubah), sedangkan output pada *plant* linear lebih renggang (lebih konstan), sehingga JST lebih mudah mendapatkan pola.

### 1.4 Analisis konvergensi JST untuk *plant* nonlinear

Persoalan mengenai konvergensi dari *plant* nonlinear terlihat tidak masalah dalam laporan ini.

Ditinjau sebagai sistem, *plant* nonlinear yang digunakan terbilang stabil, terlihat dari hasil masukan dan keluaran yang disimulasikan untuk input acak, sinusoidal, dan step. Sedangkan, ditinjau dari sisi pelatihan dan pengembangan JST untuk *plant* nonlinear, didapatkan error yang terbilang kecil (dalam orde  hingga ), sehingga dapat dikatakan konvergen.

### 1.5 Analisis kinerja *single-neuron adaptive PID*

*Plant* yang digunakan untuk simulasi *single-neuron adaptive PID* (SNAPID) merupakan *plant* yang berbeda dari kedua *plant* yang digunakan di atas. *Plant* yang

digunakan tidak stabil, terlihat dari keluaran yang divergen (membesar) ketika diberikan referensi apapun.

Setelah diberikan SNAPID dengan parameter-parameter yang digunakan di atas, terlihat bahwa simulasi sistem bisa menjadi stabil untuk referensi sinusoidal dan referensi step. Untuk referensi acak, SNAPID tidak mampu memberikan hasil yang baik. Hal ini disebabkan karena referensi yang acak tersebut. SNAPID berusaha untuk mengompensasi, tetapi karena *plant* yang digunakan tidak stabil, kemampuan SNAPID untuk mengompensasi sangat terbatas, sehingga ketidakstabilan *plant* mengambil alih dan menyebabkan divergensi. Akan tetapi, mengingat referensi acak tidak umum digunakan (karena referensi digunakan agar sistem menghasilkan output tertentu yang mendekati referensi/*setpoint* tersebut), hal ini bukan menjadi masalah yang besar.

### 3.4.3 Evan

Menggunakan Parameter Pembelajaran Yakni:

- hidden neuron: 32
- alpha: 0.01
- miu: 0.02
- epoch: 1000

Untuk SNAPID:

$$\boldsymbol{w} = \begin{bmatrix} 5 \\ 0.1 \\ 0.9 \end{bmatrix}, \boldsymbol{\eta} = \begin{bmatrix} 2 \\ 0.01 \\ 0.06 \end{bmatrix}, \text{dan } K = 8.5$$

### 3.5.1 Analisis Jaringan Syaraf Tiruan

Penggunaan metode BPNN dalam plant tersebuyt baik secara plant linear maupun linear dapat terlihat menghasilkan data yang cenderung bagus. Hal ini dibuktikan dengan nilai MSE yang sangat kecil baik pada data training maupun dilihat dari data testing. Penggunaan plant juga membuktikan bahwa plant bersifat konvergen dengan nilai MSE yang kecil dan juga epoch yang sedikit.

Selain itu dengan pengunaan Jaringan Syaraf Tiruan, ditemukan bahwa dengan penggunaan plant nonlinear akan memberikan hasil model yang lebih baik apabila

dibandingkan dengan penggunaan plant linear. Hal ini dikarenakan dengan penggunaan plant linear cenderung memiliki suku yang lebih banyak dibanding dengan plant nonlinear sehingga lebih kompleks secara matematis. Namun perbedaan hasil ini tidak terlalu signifikan

Setelah dilakukan proses retrain atau fine tuning guna menghilangkan overfitting dan memperbaiki hasil. Dalam hasil dengna penggunaan fine tuning, ditemukan bahwa dengan adanya tuning pada data acak tidak terlalu berpengaruh. Hal ini dikarenakan nilai error pada data acak telah diperoleh nilai erro yang kecil.

Namun apabila dilihat menggunakan dataset dari sinus dan step maka dapat terlihat hasilnya membaik secara signifikan.

Fenomena ini terjadi yakni pada sinus dan step lebih baik namun pada data acak cenderung kurang karena. Padafine tuning memiliki kelemahan seperti menggunakan keluaran yang berpojtensi menyebabkan kesalahan.  Hal ini dikarenakan Keluaran ini lalu dibuat sebagai umpan balik ke masukan JST. Masukan JST ini akan menghasilkan keluaran lagi yang berpotensi menyebabkan kesalahan kembali. Dengan menggunakan input acak, pola dari konvergensi jaringan akan lebih sulit untuk dipelajari JST, sedangkan dengan jaringan yang sama, untuk input sinusoidal dan input step, pola lebih mudah ditangkap oleh JST, sehingga memberikan hasil pengujian yang lebih baik.

### 3.5.2 Analisis Sistem berbasis Inversi

Pada sistem inversi dilakukan guna mengcari konfigurasi inversi apakah secara matematis eksis. Dimana apabila terdapat beberapa input yang menghasilkan output yang sama maka inversi tidak mungkin untik eksis.

Dalam mencari persamaan invers maka dapat dilakukan pendekatan menginvers persamaan yang ada pada plant linear dan plant non linear. Dimana apabila dilihat pada plant non linear terlihat bahwa persamaan plant non linear dapat diinversi. Atau dapat disebutkan sebagai fungsi invers eksis.

Namun sebaliknya pada persamaan plant linear ditemukan bahwa fungsi plant linear tidak dapat diinvers karena persamaannya yang tidak memungkinkan.

Sehingga dengan penggunaan JST berbasis inversi ditemukan bahwa nilai error menjadi jauh lebih tinggi apabila dibandingkan dengan palnt non linear karena jaringan

yang dimiliki tidak dapat menemukan fungsi matematis yang ekuivalen dengan invers plant. Oleh karena itu pula juga terbukti bedasarkan pengamatan bahwa dengan penggunaan invers maka akan memberikan pengujian yang lebih baik pada plant non linear dibanding dengan plant linear.

Analisis DIC

Setelah dilakukan membuat pengendali dan plant yang ingin dikendalikan secara baik. Hal ini dilakukan dengan menggunakan bobot bobot yang telah didapatkan dari proses pelatihan pengendali dan identifikasi sistem dan Menyusun program forward pass dimulai dari pengendali kemudian masuk ke plant yang telah diidentifikasi untuk menghasilkan luaran dari plant. Pada dasarnya, DIC dimulai dengan memberikan output reference kepada model inversi. Selanjutnya model inversi akan menghasilkan input yang sesuai dengan output tersebut untuk diberikan kepada hasil sistem identifikasi. Seelanjutnya sistem identifikasi akan memproses input tersebut untuk menghasilkan output dari plant. Output actual inilah yang akan dibandingkan dengan output reference apakah sesuai atau tidak.

Pada data hasil nilai error yang diperoleh terbukti bahwa tidak sedikit. Namun, hasil ini adalah hasil terbaik yang dapat diperoleh setelah beberapa upaya dengan mengubah penundaan input dan output. Nilai MSE dari DIC yang tidak terlalu besar dapat disebabkan oleh delay yang tidak tepat atau arsitektur yang tidak tepat.

### 3.5.3 Analisis Plant Non Linear

Dalam penggunaan plant non linear memiliki tingkat error rate yang jauh lebih kecil dibandingkan dengan plant linear. Sehingga dapat dibuktikan bahwa penggunaan plant non linear jauh lebih baik.

Namun tingkat error cenderung kecil dan tidak terlalu berpengaruh sehingga plant nonlinear dapat disebut lebih stabil dan masukan dan keluaran disimulasikan dengan input

acak sinusoidal dan step. Juga apabila ditinjau dari sisi pelatihan dan pengembangan JST maka diperoleh error yang kecil sehingga konvergen

### 3.5.4 Analisis Single Neuron Adaptive PID

Berbeda dengan dua plant yang disebutkan sebelumnya, plant yang berbeda digunakan untuk simulasi PID (SNAPID) adaptif neuron tunggal. plant yang digunakan tidak stabil, yang dibuktikan dengan output yang menyimpang (memperbesar) ketika ada referensi yang diberikan.

Dapat diamati bahwa simulasi sistem dapat stabil untuk referensi sinusoidal dan referensi langkah setelah memberikan SNAPID dengan nilai yang diberikan di atas. SNAPID tidak dapat memberikan hasil yang memuaskan untuk referensi acak. Referensi acak yang harus disalahkan untuk ini. Meskipun SNAPID berusaha untuk menebus plant yang tidak stabil, kemampuan SNAPID untuk melakukannya sangat dibatasi, dan akibatnya, ketidakstabilan plant mengambil kendali dan menyebabkan divergensi. Namun, karena referensi acak jarang digunakan karena referensi adalah yang memungkinkan sistem menghasilkan output tertentu yang mendekati referensi/*setpoint* tersebut

# BAB IV
# KESIMPULAN

## 4.1. Kesimpulan

Berdasarkan pembahasan, dapat simpulkan bahwa:

- Jaringan saraf tiruan adalah algoritma adaptif yang dapat digunakan untuk memodelkan masalah kompleks dengan sistem dinamik memakai prinsip dari jaringan otak manusia dalam mengenal pola
- Propagasi-balik adalah metode pembelajaran jaringan saraf tiruan dengan menyesuaikan bobot antar-neuron
- Jaringan saraf tiruan dapat dilatih untuk memprediksi keluaran dari suatu sistem yang direpresentasikan dengan fungsi matematis
- Parameter yang digunakan dari sistem jaringan dapat mempengaruhi hasil dari prediksi atau MSE dari jaringan
- Dengan menggunakan fungsi sigmoid sebagai fungsi aktivasi, data masukan analog (berupa sinyal sinusoid) lebih baik dibandingkan menggunakan data diskrit yang acak
- Fine-tuning merupakan suatu proses penggunaan fungsi dari hasil feedforward yang dipakai untuk feedback pada kondisi dari output sebelumnya
- Implementasi dari *fine tuning* pada data acak lebih baik dengan implementasi *fine-tuning* pada data sinewave
- Jaringan saraf tiruan dapat digunakan untuk memodelkan sistem pengendalian suatu *plant* dengan menggunakan metode sistem inversi.
- Masing-masing data memiliki metode masing-masing yang paling cocok untuk diimplementasikan.
- Metode Direct Inverse Control adalah salah satu teknik pengendali yang menggunakan fungsi invers dari plant untuk sistem pengendalinya secara open loop

**LAMPIRAN PROGRAM**

```
function train(file, mu_init)
    % Import
    train_data = load(strcat(file, '_preprocessed.mat')).train_data;

    % Pemisahan input dengan target
    input = train_data(:, 2:end);
    target = train_data(:, 1);
    [train_size, col_input] = size(input);

    % Inisialisasi jumlah neuron pada lapisan tersembunyi
    num_of_hidden = 16;

    % Inisialisasi parameter
    alpha = 0.0095; % learning rate
    mu = mu_init; % konstanta momentum
    stop_flag = 0;
    stop_crit = 1e-7;
    max_epoch = 1000;
    E = zeros(1, train_size);

    % Inisialisasi bobot menggunakan metode Nguyen-Widrow
    Vij = rand(col_input, num_of_hidden) - 0.5;
    Wjk = rand(num_of_hidden, 1) - 0.5;
    Vj_mag = sqrt(sum(Vij .^ 2));
    Wk_mag = sqrt(sum(Wjk .^ 2));
    beta1 = 0.7 * (num_of_hidden) ^ (1/col_input);
    beta2 = 0.7 * (1 ^ (1/num_of_hidden));
    for i = 1:col_input
        Vij(i, :) = beta1 * Vij(i, : ) ./ Vj_mag;
    end
    for i = 1:num_of_hidden
        Wjk(i, :) = beta2 * Wjk(i, : ) ./ Wk_mag;
    end

    % Inisialisasi bias menggunakan metode Nguyen-Widrow
    V0j = (2*beta1).*rand(1, num_of_hidden) - beta1;
    W0k = (2*beta2).*rand(1, 1) - beta2;

    % Inisialisasi gradien bobot
    deltaVij_old = zeros(col_input, num_of_hidden);
    deltaWjk_old = zeros(num_of_hidden, 1); % 1 = output hanya satu
jenis (y)

    % Training
    epoch = 1;
    while ~stop_flag && epoch <= max_epoch
        % Inisialisasi error dan batch
        n = 1;
        E(1, n) = 0;
```

```
        X = zeros(1, col_input);
        y = zeros(1, 1);
        % Untuk setiap baris pada data training
        for n = 1:train_size
            % Feedforward
            X(n, :) = input(n, :);
            y(n, :) = target(n, :);
            Z_inj = V0j + X(n, :) * Vij;
            Zj = 2 ./ (1 + exp(-2*Z_inj)) - 1; % fungsi aktivasi tanh
            Y_ink = W0k + Zj * Wjk;
            Yk = 2 ./ (1 + exp(-2*Y_ink)) - 1; % fungsi aktivasi tanh

            % Backpropagation
            E(1, n) = 0.5*sum((y(n, :) - Yk).^2); % MSE loss function
            do_k = (y(n, :) - Yk) .* (1 - Yk.^2);
            deltaWjk = alpha * Zj' * do_k;
            deltaW0k = alpha * do_k;
            do_j = (do_k * Wjk') .* (1 - Zj.^2);
            deltaVij = alpha * X(n, :)' * do_j;
            deltaV0j = alpha * do_j;

            % Update bobot, bias, dan gradien (delta) menggunakan
momentum
            Wjk = Wjk + deltaWjk + (mu * deltaWjk_old);
            W0k = W0k + deltaW0k;
            Vij = Vij + deltaVij + (mu * deltaVij_old);
            V0j = V0j + deltaV0j;
            deltaWjk_old = deltaWjk;
            deltaVij_old = deltaVij;
        end
        error(1, epoch) = sum(E)/train_size;
        if mod(epoch, 50) == 0
            fprintf('Epoch %d: \n', epoch);
            fprintf('Error rata-rata: %d\n', error(1, epoch));
        end
        if error(1, epoch) < stop_crit
            stop_flag = 1;
        end
        epoch = epoch + 1;
    end

    % Plot loss terhadap epoch
    plot(1:size(error, 2), error)
    xlabel('Epoch')
    ylabel('Error rata-rata')

    % Menyimpan bobot
    save(strcat('weights_', file, '.mat'), 'V0j', 'Vij', 'W0k',
'Wjk');
    disp('Proses training selesai!');
end
```

```
function test_linear(file)
    % Import data
    test_data = load(strcat(file, '_preprocessed.mat')).test_data;

    % Pemisahan input dengan target
    test_input = test_data(:, 2:end);
    test_target = test_data(:, 1);
    [test_size, col_test] = size(test_input);

    % Import bobot
    V0j = load(strcat('weights_data2_more.mat')).V0j;
    Vij = load(strcat('weights_data2_more.mat')).Vij;
    W0k = load(strcat('weights_data2_more.mat')).W0k;
    Wjk = load(strcat('weights_data2_more.mat')).Wjk;

    % Hanya feedforward
    X = zeros(1, col_test);
    y = zeros(1, 1);
    yhat = zeros(1500, 1); % variabel untuk menyimpan output
    for n = 1:test_size
        % Karena test, cukup feedforward
        X(n, :) = test_input(n, :);
        y(n, :) = test_target(n, :);
        Z_inj = V0j + X(n, :) * Vij;
        Zj = 2 ./ (1 + exp(-2*Z_inj)) - 1; % fungsi aktivasi tanh
        Y_ink = W0k + Zj * Wjk;
        Yk = 2 ./ (1 + exp(-2*Y_ink)) - 1; % fungsi aktivasi tanh

        yhat(n) = Yk; % menyimpan untuk diplot

        E(1, n) = 0.5*sum((y(n, :) - Yk).^2); % MSE loss function
    end
    avg_error = sum(E)/test_size;
    disp("Pengujian terhadap test set");
    fprintf("\tMSE: %.5f\n", avg_error);

    % Plot hasil
    x = 1:500;
    plot(x, yhat(x, :))
    hold on
    plot(x, test_target(x, :))
    hold off
    title('Plot hasil uji pada test set')
    legend('Prediksi', 'Aktual')
end

function test_nonlinear(file)
    % Import data
    test_data = load(strcat(file, '_preprocessed.mat')).test_data;

    % Pemisahan input dengan target
    test_input = test_data(:, 2:end);
```

```
    test_target = test_data(:, 1);
    [test_size, col_test] = size(test_input);

    % Import bobot
    V0j = load(strcat('weights_data1.mat')).V0j;
    Vij = load(strcat('weights_data1.mat')).Vij;
    W0k = load(strcat('weights_data1.mat')).W0k;
    Wjk = load(strcat('weights_data1.mat')).Wjk;

    % Hanya feedforward
    X = zeros(1, col_test);
    y = zeros(1, 1);
    yhat = zeros(1500, 1); % variabel untuk menyimpan output
    for n = 1:test_size
        % Karena test, cukup feedforward
        X(n, :) = test_input(n, :);
        y(n, :) = test_target(n, :);
        Z_inj = V0j + X(n, :) * Vij;
        Zj = 2 ./ (1 + exp(-2*Z_inj)) - 1; % fungsi aktivasi tanh
        Y_ink = W0k + Zj * Wjk;
        Yk = 2 ./ (1 + exp(-2*Y_ink)) - 1; % fungsi aktivasi tanh

        yhat(n) = Yk; % menyimpan untuk diplot

        E(1, n) = 0.5*sum((y(n, :) - Yk).^2); % MSE loss function
    end
    avg_error = sum(E)/test_size;
    disp("Pengujian terhadap test set");
    fprintf("\tMSE: %.5f\n", avg_error);

    % Plot hasil
    x = 1:100;
    plot(x, yhat(x, :))
    hold on
    plot(x, test_target(x, :))
    hold off
    title('Plot hasil uji pada test set')
    legend('Prediksi', 'Aktual')
end

function train_stage2_nonlinear(file, mu_init)
    % Import
    train_data = load(strcat(file, '_preprocessed.mat')).train_data;

    % Pemisahan input dengan target
    input = train_data(:, 2:end);
    target = train_data(:, 1);
    [train_size, col_input] = size(input);

    % Mengosongkan input feedback untuk proses fine tuning
    input(:, 1:2) = zeros(train_size, 2);
```

```
    % Inisialisasi jumlah neuron pada lapisan tersembunyi
    num_of_hidden = 16;

    % Inisialisasi parameter
    alpha = 0.0095; % learning rate
    mu = mu_init; % konstanta momentum
    stop_flag = 0;
    stop_crit = 1e-4;
    max_epoch = 1000;
    E = zeros(1, train_size);

    % Import bobot
    V0j = load(strcat('weights_', file, '.mat')).V0j;
    Vij = load(strcat('weights_', file, '.mat')).Vij;
    W0k = load(strcat('weights_', file, '.mat')).W0k;
    Wjk = load(strcat('weights_', file, '.mat')).Wjk;

    % Inisialisasi gradien bobot
    deltaVij_old = zeros(col_input, num_of_hidden);
    deltaWjk_old = zeros(num_of_hidden, 1); % 1 = output hanya satu
jenis (y)

    % Training
    epoch = 1;
    while ~stop_flag && epoch <= max_epoch
        % Inisialisasi error dan batch
        n = 1;
        E(1, n) = 0;
        X = zeros(1, col_input);
        y = zeros(1, 1);
        % Untuk setiap baris pada data training
        for n = 1:train_size
            % Feedforward
            X(n, :) = input(n, :);
            y(n, :) = target(n, :);
            Z_inj = V0j + X(n, :) * Vij;
            Zj = 2 ./ (1 + exp(-2*Z_inj)) - 1; % fungsi aktivasi tanh
            Y_ink = W0k + Zj * Wjk;
            Yk = 2 ./ (1 + exp(-2*Y_ink)) - 1; % fungsi aktivasi tanh

            % FEEDBACK
            input(n+1, 2) = X(n, 1); % y(k-1) di-pass ke y(k-2)
            input(n+1, 1) = Yk; % y(k) di-pass ke y(k-1)

            % Backpropagation
            E(1, n) = 0.5*sum((y(n, :) - Yk).^2); % MSE loss function
            do_k = (y(n, :) - Yk) .* (1 - Yk.^2);
            deltaWjk = alpha * Zj' * do_k;
            deltaW0k = alpha * do_k;
            do_j = (do_k * Wjk') .* (1 - Zj.^2);
            deltaVij = alpha * X(n, :)' * do_j;
            deltaV0j = alpha * do_j;
```

```
            % Update bobot, bias, dan gradien (delta) menggunakan
momentum
            Wjk = Wjk + deltaWjk + (mu * deltaWjk_old);
            W0k = W0k + deltaW0k;
            Vij = Vij + deltaVij + (mu * deltaVij_old);
            V0j = V0j + deltaV0j;
            deltaWjk_old = deltaWjk;
            deltaVij_old = deltaVij;
        end
        error(1, epoch) = sum(E)/train_size;
        if mod(epoch, 50) == 0
            fprintf('Epoch %d: \n', epoch);
            fprintf('Error rata-rata: %d\n', error(1, epoch));
        end
        if error(1, epoch) < stop_crit
            stop_flag = 1;
        end
        epoch = epoch + 1;
    end
    % Koreksi pada matriks input (row berlebih)
    input(end, :) = [];

    % Plot loss terhadap epoch
    plot(1:size(error, 2), error)
    xlabel('Epoch')
    ylabel('Error rata-rata')

    % Menyimpan bobot
    save(strcat('weights_', file, '_stage2', '.mat'), 'V0j', 'Vij',
'W0k', 'Wjk');

    % Menyimpan data tahap 2
    train_data = [target input];
    test_data = load(strcat(file, '_preprocessed.mat')).test_data;
    save(strcat(file, '_stage2', '_preprocessed'), 'train_data',
'test_data');
    disp('Proses training selesai!');
end
function train_stage2_linear(file, mu_init)
    % Import
    train_data = load(strcat(file, '_preprocessed.mat')).train_data;

    % Pemisahan input dengan target
    input = train_data(:, 2:end);
    target = train_data(:, 1);
    [train_size, col_input] = size(input);

    % Mengosongkan input feedback untuk proses fine tuning
    input(:, 1:2) = zeros(train_size, 2);

    % Inisialisasi jumlah neuron pada lapisan tersembunyi
```

```
    num_of_hidden = 16;

    % Inisialisasi parameter
    alpha = 0.0095; % learning rate
    mu = mu_init; % konstanta momentum
    stop_flag = 0;
    stop_crit = 1e-4;
    max_epoch = 1000;
    E = zeros(1, train_size);

    % Import bobot
    V0j = load(strcat('weights_', file, '.mat')).V0j;
    Vij = load(strcat('weights_', file, '.mat')).Vij;
    W0k = load(strcat('weights_', file, '.mat')).W0k;
    Wjk = load(strcat('weights_', file, '.mat')).Wjk;

    % Inisialisasi gradien bobot
    deltaVij_old = zeros(col_input, num_of_hidden);
    deltaWjk_old = zeros(num_of_hidden, 1); % 1 = output hanya satu
jenis (y)

    % Training
    epoch = 1;
    while ~stop_flag && epoch <= max_epoch
        % Inisialisasi error dan batch
        n = 1;
        E(1, n) = 0;
        X = zeros(1, col_input);
        y = zeros(1, 1);
        % Untuk setiap baris pada data training
        for n = 1:train_size
            % Feedforward
            X(n, :) = input(n, :);
            y(n, :) = target(n, :);
            Z_inj = V0j + X(n, :) * Vij;
            Zj = 2 ./ (1 + exp(-2*Z_inj)) - 1; % fungsi aktivasi tanh
            Y_ink = W0k + Zj * Wjk;
            Yk = 2 ./ (1 + exp(-2*Y_ink)) - 1; % fungsi aktivasi tanh

            % FEEDBACK
            input(n+1, 3) = X(n, 2); % y(k-2) di-pass ke y(k-3)
            input(n+1, 2) = X(n, 1); % y(k-1) di-pass ke y(k-2)
            input(n+1, 1) = Yk; % y(k) di-pass ke y(k-1)

            % Backpropagation
            E(1, n) = 0.5*sum((y(n, :) - Yk).^2); % MSE loss function
            do_k = (y(n, :) - Yk) .* (1 - Yk.^2);
            deltaWjk = alpha * Zj' * do_k;
            deltaW0k = alpha * do_k;
            do_j = (do_k * Wjk') .* (1 - Zj.^2);
            deltaVij = alpha * X(n, :)' * do_j;
            deltaV0j = alpha * do_j;
```

```
            % Update bobot, bias, dan gradien (delta) menggunakan
momentum
            Wjk = Wjk + deltaWjk + (mu * deltaWjk_old);
            W0k = W0k + deltaW0k;
            Vij = Vij + deltaVij + (mu * deltaVij_old);
            V0j = V0j + deltaV0j;
            deltaWjk_old = deltaWjk;
            deltaVij_old = deltaVij;
        end
        error(1, epoch) = sum(E)/train_size;
        if mod(epoch, 50) == 0
            fprintf('Epoch %d: \n', epoch);
            fprintf('Error rata-rata: %d\n', error(1, epoch));
        end
        if error(1, epoch) < stop_crit
            stop_flag = 1;
        end
        epoch = epoch + 1;
    end
    % Koreksi pada matriks input (row berlebih)
    input(end, :) = [];

    % Plot loss terhadap epoch
    plot(1:size(error, 2), error)
    xlabel('Epoch')
    ylabel('Error rata-rata')

    % Menyimpan bobot
    save(strcat('weights_', file, '_stage2', '.mat'), 'V0j', 'Vij',
'W0k', 'Wjk');

    % Menyimpan data tahap 2
    train_data = [target input];
    test_data = load(strcat(file, '_preprocessed.mat')).test_data;
    save(strcat(file, '_stage2', '_preprocessed'), 'train_data',
'test_data');
    disp('Proses training selesai!');
end
function test_stage2_nonlinear(file)
    % Import data
    test_data = load(strcat(file,
'_stage2_preprocessed.mat')).test_data;

    % Pemisahan input dengan target
    test_input = test_data(:, 2:end);
    test_target = test_data(:, 1);
    [test_size, col_test] = size(test_input);

    % Import bobot
    V0j = load(strcat('weights_data1_stage2.mat')).V0j;
    Vij = load(strcat('weights_data1_stage2.mat')).Vij;
```

```
    W0k = load(strcat('weights_data1_stage2.mat')).W0k;
    Wjk = load(strcat('weights_data1_stage2.mat')).Wjk;

    % Hanya feedforward
    X = zeros(1, col_test);
    y = zeros(1, 1);
    yhat = zeros(1500, 1); % variabel untuk menyimpan output
    for n = 1:test_size
        % Karena test, cukup feedforward
        X(n, :) = test_input(n, :);
        y(n, :) = test_target(n, :);
        Z_inj = V0j + X(n, :) * Vij;
        Zj = 2 ./ (1 + exp(-2*Z_inj)) - 1; % fungsi aktivasi tanh
        Y_ink = W0k + Zj * Wjk;
        Yk = 2 ./ (1 + exp(-2*Y_ink)) - 1; % fungsi aktivasi tanh

        yhat(n) = Yk; % menyimpan untuk diplot

        % FEEDBACK
        test_input(n+1, 2) = X(n, 1); % y(k-1) di-pass ke y(k-2)
        test_input(n+1, 1) = Yk; % y(k) di-pass ke y(k-1)

        E(1, n) = 0.5*sum((y(n, :) - Yk).^2); % MSE loss function
    end
    avg_error = sum(E)/test_size;
    disp("Pengujian terhadap test set");
    fprintf("\tMSE: %.5f\n", avg_error);

    % Plot hasil
    x = 1:700;
    plot(x, yhat(x, :))
    hold on
    plot(x, test_target(x, :))
    hold off
    title('Plot hasil uji pada test set, dengan fine tuning')
    legend('Prediksi', 'Aktual')
end
function test_stage2_linear(file)
    % Import data
    test_data = load(strcat(file, '_preprocessed.mat')).test_data;

    % Pemisahan input dengan target
    test_input = test_data(:, 2:end);
    test_target = test_data(:, 1);
    [test_size, col_test] = size(test_input);

    % Import bobot
%     V0j = load(strcat('weights_data2_more_stage2.mat')).V0j;
%     Vij = load(strcat('weights_data2_more_stage2.mat')).Vij;
%     W0k = load(strcat('weights_data2_more_stage2.mat')).W0k;
%     Wjk = load(strcat('weights_data2_more_stage2.mat')).Wjk;
    weights = load(strcat('weights_', file, '_stage2.mat'));
```

```
    V0j = weights.V0j;
    Vij = weights.Vij;
    W0k = weights.W0k;
    Wjk = weights.Wjk;

    % Hanya feedforward
    X = zeros(1, col_test);
    y = zeros(1, 1);
    yhat = zeros(1500, 1); % variabel untuk menyimpan output
    for n = 1:test_size
        % Karena test, cukup feedforward
        X(n, :) = test_input(n, :);
        y(n, :) = test_target(n, :);
        Z_inj = V0j + X(n, :) * Vij;
        Zj = 2 ./ (1 + exp(-2*Z_inj)) - 1; % fungsi aktivasi tanh
        Y_ink = W0k + Zj * Wjk;
        Yk = 2 ./ (1 + exp(-2*Y_ink)) - 1; % fungsi aktivasi tanh

        yhat(n) = Yk; % menyimpan untuk diplot

        E(1, n) = 0.5*sum((y(n, :) - Yk).^2); % MSE loss function
    end
    avg_error = sum(E)/test_size;
    disp("Pengujian terhadap test set");
    fprintf("\tMSE: %.5f\n", avg_error);

    % Plot hasil
    x = 1:800;
    plot(x, yhat(x, :))
    hold on
    plot(x, test_target(x, :))
    hold off
    title('Plot hasil uji pada test set, dengan fine tuning')
    legend('Prediksi', 'Aktual')
end
function train_inverse_linear(file, HIDDEN, ALPHA, MU, EPOCH)
    % Import
    % Struktur data: y(k) | y(k-1) | y(k-2) | u(k) | u(k-1) | u(k-2)
    train_data = load(strcat(file, '_preprocessed.mat')).train_data;

    % Pemisahan input dengan target
    input = train_data(:, [1 2 3 4 6 7 8]);
    target = train_data(:, 5);
    [train_size, col_input] = size(input);

    % Inisialisasi jumlah neuron pada lapisan tersembunyi
    num_of_hidden = HIDDEN;

    % Inisialisasi parameter
    alpha = ALPHA; % learning rate
    mu = MU; % konstanta momentum
    stop_flag = 0;
```

```
    stop_crit = 1e-4;
    max_epoch = EPOCH;
    E = zeros(1, train_size);

    % Inisialisasi bobot menggunakan metode Nguyen-Widrow
    Vij = rand(col_input, num_of_hidden) - 0.5;
    Wjk = rand(num_of_hidden, 1) - 0.5;
    Vj_mag = sqrt(sum(Vij .^ 2));
    Wk_mag = sqrt(sum(Wjk .^ 2));
    beta1 = 0.7 * (num_of_hidden) ^ (1/col_input);
    beta2 = 0.7 * (1) ^ (1/num_of_hidden);
    for i = 1:col_input
        Vij(i, :) = beta1 * Vij(i, : ) ./ Vj_mag;
    end
    for i = 1:num_of_hidden
        Wjk(i, :) = beta2 * Wjk(i, : ) ./ Wk_mag;
    end

    % Inisialisasi bias menggunakan metode Nguyen-Widrow
    V0j = (2*beta1).*rand(1, num_of_hidden) - beta1;
    W0k = (2*beta2).*rand(1, 1) - beta2;

    % Inisialisasi gradien bobot
    deltaVij_old = zeros(col_input, num_of_hidden);
    deltaWjk_old = zeros(num_of_hidden, 1); % 1 = output hanya satu
jenis (y)

    % Training
    epoch = 1;
    while ~stop_flag && epoch <= max_epoch
        % Inisialisasi error dan batch
        n = 1;
        E(1, n) = 0;
        X = zeros(1, col_input);
        y = zeros(1, 1);
        % Untuk setiap baris pada data training
        for n = 1:train_size
            % Feedforward
            X(n, :) = input(n, :);
            y(n, :) = target(n, :);
            Z_inj = V0j + X(n, :) * Vij;
            Zj = 2 ./ (1 + exp(-2*Z_inj)) - 1; % fungsi aktivasi tanh
            Y_ink = W0k + Zj * Wjk;
            Yk = 2 ./ (1 + exp(-2*Y_ink)) - 1; % fungsi aktivasi tanh

            % Backpropagation
            E(1, n) = 0.5 * sum((y(n, :) - Yk).^2); % MSE loss
function
            do_k = (y(n, :) - Yk) .* (1 - Yk.^2);
            deltaWjk = alpha * Zj' * do_k;
            deltaW0k = alpha * do_k;
            do_j = (do_k * Wjk') .* (1 - Zj.^2);
```

```
            deltaVij = alpha * X(n, :)' * do_j;
            deltaV0j = alpha * do_j;

            % Update bobot, bias, dan gradien (delta) menggunakan
momentum
            Wjk = Wjk + deltaWjk + (mu * deltaWjk_old);
            W0k = W0k + deltaW0k;
            Vij = Vij + deltaVij + (mu * deltaVij_old);
            V0j = V0j + deltaV0j;
            deltaWjk_old = deltaWjk;
            deltaVij_old = deltaVij;
        end
        error(1, epoch) = sum(E)/train_size;
        if mod(epoch, 50) == 0 % Print setiap 50 epoch
            fprintf('Epoch %d: \n', epoch);
            fprintf('Error rata-rata: %d\n', error(1, epoch));
        end
        if error(1, epoch) < stop_crit
            stop_flag = 1;
        end
        epoch = epoch + 1;
    end

    % Plot error terhadap epoch
    plot(1:size(error, 2), error)
    xlabel('Epoch')
    ylabel('Error rata-rata')

    % Menyimpan bobot
    save(strcat('weights_', file, '_inverse', '.mat'), 'V0j', 'Vij',
'W0k', 'Wjk');
    disp('Proses training selesai!');
end
function train_inverse_nonlinear(file, HIDDEN, ALPHA, MU, EPOCH)
    % Import
    % Struktur data: y(k) | y(k-1) | y(k-2) | u(k) | u(k-1) | u(k-2)
    train_data = load(strcat(file, '_preprocessed.mat')).train_data;

    % Pemisahan input dengan target
    input = train_data(:, [1 2 3 5 6]);
    target = train_data(:, 4);
    [train_size, col_input] = size(input);

    % Inisialisasi jumlah neuron pada lapisan tersembunyi
    num_of_hidden = HIDDEN;

    % Inisialisasi parameter
    alpha = ALPHA; % learning rate
    mu = MU; % konstanta momentum
    stop_flag = 0;
    stop_crit = 1e-4;
    max_epoch = EPOCH;
```

```
    E = zeros(1, train_size);

    % Inisialisasi bobot menggunakan metode Nguyen-Widrow
    Vij = rand(col_input, num_of_hidden) - 0.5;
    Wjk = rand(num_of_hidden, 1) - 0.5;
    Vj_mag = sqrt(sum(Vij .^ 2));
    Wk_mag = sqrt(sum(Wjk .^ 2));
    beta1 = 0.7 * (num_of_hidden) ^ (1/col_input);
    beta2 = 0.7 * (1) ^ (1/num_of_hidden);
    for i = 1:col_input
        Vij(i, :) = beta1 * Vij(i, : ) ./ Vj_mag;
    end
    for i = 1:num_of_hidden
        Wjk(i, :) = beta2 * Wjk(i, : ) ./ Wk_mag;
    end

    % Inisialisasi bias menggunakan metode Nguyen-Widrow
    V0j = (2*beta1).*rand(1, num_of_hidden) - beta1;
    W0k = (2*beta2).*rand(1, 1) - beta2;

    % Inisialisasi gradien bobot
    deltaVij_old = zeros(col_input, num_of_hidden);
    deltaWjk_old = zeros(num_of_hidden, 1); % 1 = output hanya satu
jenis (y)

    % Training
    epoch = 1;
    while ~stop_flag && epoch <= max_epoch
        % Inisialisasi error dan batch
        n = 1;
        E(1, n) = 0;
        X = zeros(1, col_input);
        y = zeros(1, 1);
        % Untuk setiap baris pada data training
        for n = 1:train_size
            % Feedforward
            X(n, :) = input(n, :);
            y(n, :) = target(n, :);
            Z_inj = V0j + X(n, :) * Vij;
            Zj = 2 ./ (1 + exp(-2*Z_inj)) - 1; % fungsi aktivasi tanh
            Y_ink = W0k + Zj * Wjk;
            Yk = 2 ./ (1 + exp(-2*Y_ink)) - 1; % fungsi aktivasi tanh

            % Backpropagation
            E(1, n) = 0.5 * sum((y(n, :) - Yk).^2); % MSE loss
function
            do_k = (y(n, :) - Yk) .* (1 - Yk.^2);
            deltaWjk = alpha * Zj' * do_k;
            deltaW0k = alpha * do_k;
            do_j = (do_k * Wjk') .* (1 - Zj.^2);
            deltaVij = alpha * X(n, :)' * do_j;
            deltaV0j = alpha * do_j;
```

```
            % Update bobot, bias, dan gradien (delta) menggunakan
momentum
            Wjk = Wjk + deltaWjk + (mu * deltaWjk_old);
            W0k = W0k + deltaW0k;
            Vij = Vij + deltaVij + (mu * deltaVij_old);
            V0j = V0j + deltaV0j;
            deltaWjk_old = deltaWjk;
            deltaVij_old = deltaVij;
        end
        error(1, epoch) = sum(E)/train_size;
        if mod(epoch, 50) == 0 % Print setiap 50 epoch
            fprintf('Epoch %d: \n', epoch);
            fprintf('Error rata-rata: %d\n', error(1, epoch));
        end
        if error(1, epoch) < stop_crit
            stop_flag = 1;
        end
        epoch = epoch + 1;
    end

    % Plot error terhadap epoch
    plot(1:size(error, 2), error)
    xlabel('Epoch')
    ylabel('Error rata-rata')

    % Menyimpan bobot
    save(strcat('weights_', file, '_inverse', '.mat'), 'V0j', 'Vij',
'W0k', 'Wjk');
    disp('Proses training selesai!');
end
function test_inverse_nonlinear(file)
    % Import data
    test_data = load(strcat(file, '_preprocessed.mat')).test_data;

    % Pemisahan input dengan target
    test_input = test_data(:, [1 2 3 5 6]);
    test_target = test_data(:, 4);
    [test_size, col_test] = size(test_input);

    % Import bobot
    V0j = load(strcat('weights_data1_inverse.mat')).V0j;
    Vij = load(strcat('weights_data1_inverse.mat')).Vij;
    W0k = load(strcat('weights_data1_inverse.mat')).W0k;
    Wjk = load(strcat('weights_data1_inverse.mat')).Wjk;

    % Hanya feedforward
    X = zeros(1, col_test);
    y = zeros(1, 1);
    yhat = zeros(1500, 1); % variabel untuk menyimpan output
    for n = 1:test_size
        % Karena test, cukup feedforward
```

```
        X(n, :) = test_input(n, :);
        y(n, :) = test_target(n, :);
        Z_inj = V0j + X(n, :) * Vij;
        Zj = 2 ./ (1 + exp(-2*Z_inj)) - 1; % fungsi aktivasi tanh
        Y_ink = W0k + Zj * Wjk;
        Yk = 2 ./ (1 + exp(-2*Y_ink)) - 1; % fungsi aktivasi tanh

        yhat(n) = Yk; % menyimpan untuk diplot

        E(1, n) = 0.5*sum((y(n, :) - Yk).^2); % MSE loss function
    end
    avg_error = sum(E)/test_size;
    disp("Pengujian terhadap test set");
    fprintf("\tMSE: %.5f\n", avg_error);

    % Plot hasil
    x = 1:100;
    plot(x, yhat(x, :))
    hold on
    plot(x, test_target(x, :))
    hold off
    title('Plot hasil uji pada test set')
    legend('Keluaran JST', 'Keluaran plant')
end
function test_inverse_linear(file)
    % Import data
    test_data = load(strcat(file, '_preprocessed.mat')).test_data;

    % Pemisahan input dengan target
    test_input = test_data(:, [1 2 3 4 6 7 8]);
    test_target = test_data(:, 5);
    [test_size, col_test] = size(test_input);

    % Import bobot
%     V0j = load(strcat('weights_data2_more_inverse.mat')).V0j;
%     Vij = load(strcat('weights_data2_more_inverse.mat')).Vij;
%     W0k = load(strcat('weights_data2_more_inverse.mat')).W0k;
%     Wjk = load(strcat('weights_data2_more_inverse.mat')).Wjk;
    weights = load(strcat('weights_', file, '_inverse.mat'));
    V0j = weights.V0j;
    Vij = weights.Vij;
    W0k = weights.W0k;
    Wjk = weights.Wjk;

    % Hanya feedforward
    X = zeros(1, col_test);
    y = zeros(1, 1);
    yhat = zeros(1500, 1); % variabel untuk menyimpan output
    for n = 1:test_size
        % Karena test, cukup feedforward
        X(n, :) = test_input(n, :);
        y(n, :) = test_target(n, :);
```

```
        Z_inj = V0j + X(n, :) * Vij;
        Zj = 2 ./ (1 + exp(-2*Z_inj)) - 1; % fungsi aktivasi tanh
        Y_ink = W0k + Zj * Wjk;
        Yk = 2 ./ (1 + exp(-2*Y_ink)) - 1; % fungsi aktivasi tanh

        yhat(n) = Yk; % menyimpan untuk diplot

        E(1, n) = 0.5*sum((y(n, :) - Yk).^2); % MSE loss function
    end
    avg_error = sum(E)/test_size;
    disp("Pengujian terhadap test set");
    fprintf("\tMSE: %.5f\n", avg_error);

    % Plot hasil
    x = 1:700;
    plot(x, yhat(x, :))
    hold on
    plot(x, test_target(x, :))
    hold off
    title('Plot hasil uji pada test set')
    legend('Keluaran JST', 'Keluaran plant')
end
function train_inverse_stage2_nonlinear(file, HIDDEN, ALPHA, MU,
EPOCH)
    % Import
    train_data = load(strcat(file, '_preprocessed.mat')).train_data;

    % Pemisahan input dengan target
    % Struktur input: y(k) | y(k-1) | y(k-2) | u(k-1) | u(k-2)
    % Struktur target: u(k)
    input = train_data(:, [1 2 3 5 6]);
    target = train_data(:, 4);
    [train_size, col_input] = size(input);

    % Inisialisasi jumlah neuron pada lapisan tersembunyi
    num_of_hidden = HIDDEN;

    % Inisialisasi parameter
    alpha = ALPHA; % learning rate
    mu = MU; % konstanta momentum
    stop_flag = 0;
    stop_crit = 5 * 1e-5;
    max_epoch = EPOCH;
    E = zeros(1, train_size);

    % Import bobot
    V0j = load(strcat('weights_', file, '_inverse', '.mat')).V0j;
    Vij = load(strcat('weights_', file, '_inverse', '.mat')).Vij;
    W0k = load(strcat('weights_', file, '_inverse', '.mat')).W0k;
    Wjk = load(strcat('weights_', file, '_inverse', '.mat')).Wjk;

    % Inisialisasi gradien bobot
```

```
    deltaVij_old = zeros(col_input, num_of_hidden);
    deltaWjk_old = zeros(num_of_hidden, 1); % 1 = output hanya satu
jenis (y)

    % Training
    epoch = 1;
    while ~stop_flag && epoch <= max_epoch
        % Inisialisasi error dan batch
        n = 1;
        E(1, n) = 0;
        X = zeros(1, col_input);
        y = zeros(1, 1);
        % Untuk setiap baris pada data training
        for n = 1:train_size
            % Feedforward
            X(n, :) = input(n, :);
            y(n, :) = target(n, :);
            Z_inj = V0j + X(n, :) * Vij;
            Zj = 2 ./ (1 + exp(-2*Z_inj)) - 1; % fungsi aktivasi tanh
            Y_ink = W0k + Zj * Wjk;
            % Yk = u_hat(k)
            Yk = 2 ./ (1 + exp(-2*Y_ink)) - 1; % fungsi aktivasi tanh

            % FEEDBACK
            input(n+1, 4) = Yk; % u(k) di-pass ke u(k-1)
            input(n+1, 5) = input(n, 4); % u(k-1) di-pass ke u(k-2)

            % Backpropagation
            E(1, n) = 0.5*sum((y(n, :) - Yk).^2); % MSE loss function
            do_k = (y(n, :) - Yk) .* (1 - Yk.^2);
            deltaWjk = alpha * Zj' * do_k;
            deltaW0k = alpha * do_k;
            do_j = (do_k * Wjk') .* (1 - Zj.^2);
            deltaVij = alpha * X(n, :)' * do_j;
            deltaV0j = alpha * do_j;

            % Update bobot, bias, dan gradien (delta) menggunakan
momentum
            Wjk = Wjk + deltaWjk + (mu * deltaWjk_old);
            W0k = W0k + deltaW0k;
            Vij = Vij + deltaVij + (mu * deltaVij_old);
            V0j = V0j + deltaV0j;
            deltaWjk_old = deltaWjk;
            deltaVij_old = deltaVij;
        end
        error(1, epoch) = sum(E)/train_size;
        if mod(epoch, 50) == 0
            fprintf('Epoch %d: \n', epoch);
            fprintf('Error rata-rata: %d\n', error(1, epoch));
        end
        if error(1, epoch) < stop_crit
            stop_flag = 1;
```

```
        end
        epoch = epoch + 1;
    end
    % Koreksi pada matriks input (row berlebih)
    input(end, :) = [];

    % Plot loss terhadap epoch
    plot(1:size(error, 2), error)
    xlabel('Epoch')
    ylabel('Error rata-rata')

    % Menyimpan bobot
    save(strcat('weights_', file, '_inverse_stage2', '.mat'), 'V0j',
'Vij', 'W0k', 'Wjk');

    % Menyimpan data tahap 2
    train_data = [input(:, 1) input(:, 2), target, input(:, 3),
input(:, 4)];
    test_data = load(strcat(file, '_preprocessed.mat')).test_data;
    save(strcat(file, '_inverse_stage2_preprocessed'), 'train_data',
'test_data');
    disp('Proses training selesai!');
end
function train_inverse_stage2_linear(file, HIDDEN, ALPHA, MU, EPOCH)
    % Import
    train_data = load(strcat(file, '_preprocessed.mat')).train_data;

    % Pemisahan input dengan target
    % Struktur input: y(k) | y(k-1) | y(k-2) | y(k-3) | u(k-1) | u(k-
2) |
    % u(k-3)
    % Struktur target: u(k)
    input = train_data(:, [1 2 3 4 6 7 8]);
    target = train_data(:, 5);
    [train_size, col_input] = size(input);

    % Inisialisasi jumlah neuron pada lapisan tersembunyi
    num_of_hidden = HIDDEN;

    % Inisialisasi parameter
    alpha = ALPHA; % learning rate
    mu = MU; % konstanta momentum
    stop_flag = 0;
    stop_crit = 5 * 1e-5;
    max_epoch = EPOCH;
    E = zeros(1, train_size);

    % Import bobot
    V0j = load(strcat('weights_', file, '_inverse', '.mat')).V0j;
    Vij = load(strcat('weights_', file, '_inverse', '.mat')).Vij;
    W0k = load(strcat('weights_', file, '_inverse', '.mat')).W0k;
    Wjk = load(strcat('weights_', file, '_inverse', '.mat')).Wjk;
```

```
    % Inisialisasi gradien bobot
    deltaVij_old = zeros(col_input, num_of_hidden);
    deltaWjk_old = zeros(num_of_hidden, 1); % 1 = output hanya satu
jenis (y)

    % Training
    epoch = 1;
    while ~stop_flag && epoch <= max_epoch
        % Inisialisasi error dan batch
        n = 1;
        E(1, n) = 0;
        X = zeros(1, col_input);
        y = zeros(1, 1);
        % Untuk setiap baris pada data training
        for n = 1:train_size
            % Feedforward
            X(n, :) = input(n, :);
            y(n, :) = target(n, :);
            Z_inj = V0j + X(n, :) * Vij;
            Zj = 2 ./ (1 + exp(-2*Z_inj)) - 1; % fungsi aktivasi tanh
            Y_ink = W0k + Zj * Wjk;
            % Yk = u_hat(k)
            Yk = 2 ./ (1 + exp(-2*Y_ink)) - 1; % fungsi aktivasi tanh

            % FEEDBACK
            input(n+1, 7) = input(n, 6); % u(k-2) di-pass ke u(k-3)
            input(n+1, 6) = input(n, 5); % u(k-1) di-pass ke u(k-2)
            input(n+1, 5) = Yk; % u(k) di-pass ke u(k-1)

            % Backpropagation
            E(1, n) = 0.5*sum((y(n, :) - Yk).^2); % MSE loss function
            do_k = (y(n, :) - Yk) .* (1 - Yk.^2);
            deltaWjk = alpha * Zj' * do_k;
            deltaW0k = alpha * do_k;
            do_j = (do_k * Wjk') .* (1 - Zj.^2);
            deltaVij = alpha * X(n, :)' * do_j;
            deltaV0j = alpha * do_j;

            % Update bobot, bias, dan gradien (delta) menggunakan
momentum
            Wjk = Wjk + deltaWjk + (mu * deltaWjk_old);
            W0k = W0k + deltaW0k;
            Vij = Vij + deltaVij + (mu * deltaVij_old);
            V0j = V0j + deltaV0j;
            deltaWjk_old = deltaWjk;
            deltaVij_old = deltaVij;
        end
        error(1, epoch) = sum(E)/train_size;
        if mod(epoch, 50) == 0
            fprintf('Epoch %d: \n', epoch);
            fprintf('Error rata-rata: %d\n', error(1, epoch));
```

```
        end
        if error(1, epoch) < stop_crit
            stop_flag = 1;
        end
        epoch = epoch + 1;
    end
    % Koreksi pada matriks input (row berlebih)
    input(end, :) = [];

    % Plot loss terhadap epoch
    plot(1:size(error, 2), error)
    xlabel('Epoch')
    ylabel('Error rata-rata')

    % Menyimpan bobot
    save(strcat('weights_', file, '_inverse_stage2', '.mat'), 'V0j',
'Vij', 'W0k', 'Wjk');

    % Menyimpan data tahap 2
    train_data = [input(:, 1) input(:, 2), target, input(:, 3),
input(:, 4)];
    test_data = load(strcat(file, '_preprocessed.mat')).test_data;
    save(strcat(file, '_inverse_stage2_preprocessed'), 'train_data',
'test_data');
    disp('Proses training selesai!');
end
function test_inverse_stage2_nonlinear(file)
    % Import data
    % Struktur data: y(k) | y(k-1) | y(k-2) | u(k) | u(k-1) | u(k-2)
    test_data = load(strcat(file, '_preprocessed.mat')).test_data;

    % Pemisahan input dengan target
    % Struktur input: y(k) | y(k-1) | y(k-2) | u(k-1) | u(k-2)
    % Struktur target: u(k)
    test_input = test_data(:, [1 2 3 5 6]);
    test_target = test_data(:, 4);
    [test_size, col_test] = size(test_input);

    % Import bobot
    V0j = load(strcat('weights_data1_inverse_stage2.mat')).V0j;
    Vij = load(strcat('weights_data1_inverse_stage2.mat')).Vij;
    W0k = load(strcat('weights_data1_inverse_stage2.mat')).W0k;
    Wjk = load(strcat('weights_data1_inverse_stage2.mat')).Wjk;

    % Hanya feedforward
    X = zeros(1, col_test);
    y = zeros(1, 1);
    yhat = zeros(1500, 1); % variabel untuk menyimpan output
    for n = 1:test_size
        % Karena test, cukup feedforward
        X(n, :) = test_input(n, :);
        y(n, :) = test_target(n, :);
```

```
        Z_inj = V0j + X(n, :) * Vij;
        Zj = 2 ./ (1 + exp(-2*Z_inj)) - 1; % fungsi aktivasi tanh
        Y_ink = W0k + Zj * Wjk;
        Yk = 2 ./ (1 + exp(-2*Y_ink)) - 1; % fungsi aktivasi tanh

        yhat(n) = Yk; % menyimpan untuk diplot

        % FEEDBACK
        test_input(n+1, 4) = Yk; % u(k) di-pass ke u(k-1)
        test_input(n+1, 5) = test_input(n, 4); % u(k-1) di-pass ke
u(k-2)

        E(1, n) = 0.5*sum((y(n, :) - Yk).^2); % MSE loss function
    end
    avg_error = sum(E)/test_size;
    disp("Pengujian terhadap test set");
    fprintf("\tMSE: %.5f\n", avg_error);

    % Plot hasil
    x = 1:100;
    plot(x, yhat(x, :))
    hold on
    plot(x, test_target(x, :))
    hold off
    title('Plot hasil uji pada test set, dengan fine tuning')
    legend('Keluaran JST', 'Keluaran plant')
end
function test_inverse_stage2_linear(file)
    % Import data
    % Struktur data: y(k) | y(k-1) | y(k-2) | u(k) | u(k-1) | u(k-2)
    test_data = load(strcat(file, '_preprocessed.mat')).test_data;

    % Pemisahan input dengan target
    % Struktur input: y(k) | y(k-1) | y(k-2) | y(k-3) | u(k-1) | u(k-
2) |
    % u(k-3)
    % Struktur target: u(k)
    test_input = test_data(:, [1 2 3 4 6 7 8]);
    test_target = test_data(:, 5);
    [test_size, col_test] = size(test_input);

    % Import bobot
%     V0j = load(strcat('weights_data1_inverse_stage2.mat')).V0j;
%     Vij = load(strcat('weights_data1_inverse_stage2.mat')).Vij;
%     W0k = load(strcat('weights_data1_inverse_stage2.mat')).W0k;
%     Wjk = load(strcat('weights_data1_inverse_stage2.mat')).Wjk;
weights = load(strcat('weights_', file, '_inverse_stage2.mat'));
    V0j = weights.V0j;
    Vij = weights.Vij;
    W0k = weights.W0k;
    Wjk = weights.Wjk;
```

```
    % Hanya feedforward
    X = zeros(1, col_test);
    y = zeros(1, 1);
    yhat = zeros(1500, 1); % variabel untuk menyimpan output
    for n = 1:test_size
        % Karena test, cukup feedforward
        X(n, :) = test_input(n, :);
        y(n, :) = test_target(n, :);
        Z_inj = V0j + X(n, :) * Vij;
        Zj = 2 ./ (1 + exp(-2*Z_inj)) - 1; % fungsi aktivasi tanh
        Y_ink = W0k + Zj * Wjk;
        Yk = 2 ./ (1 + exp(-2*Y_ink)) - 1; % fungsi aktivasi tanh

        yhat(n) = Yk; % menyimpan untuk diplot

        % FEEDBACK
        test_input(n+1, 7) = test_input(n, 6); % u(k-2) di-pass ke
u(k-3)
        test_input(n+1, 6) = test_input(n, 5); % u(k-1) di-pass ke
u(k-2)
        test_input(n+1, 5) = Yk; % u(k) di-pass ke u(k-1)

        E(1, n) = 0.5*sum((y(n, :) - Yk).^2); % MSE loss function
    end
    avg_error = sum(E)/test_size;
    disp("Pengujian terhadap test set");
    fprintf("\tMSE: %.5f\n", avg_error);

    % Plot hasil
    x = 1:800;
    plot(x, yhat(x, :))
    hold on
    plot(x, test_target(x, :))
    hold off
    title('Plot hasil uji pada test set, dengan fine tuning')
    legend('Keluaran JST', 'Keluaran plant')
end
function train_DIC_nonlinear(file, HIDDEN1, ALPHA, MU, EPOCH)
    % Import data
    % Struktur data: y(k) | y(k-1) | y(k-2) | u(k) | u(k-1) | u(k-2)
    train_data = load(strcat(file, '_preprocessed.mat')).train_data;

    % Pemisahan input dengan target
    % Struktur input: y(k) | y(k-1) | y(k-2) | u(k-1) | u(k-2)
    % Struktur output: u(k)
    input1 = train_data(:, [1 2 3 5 6]);
    target1 = train_data(:, 4);
    [train_size1, col_input1] = size(input1);
    input2 = train_data(:, 2:end);
    target2 = train_data(:, 1);

    % Inisialisasi jumlah neuron pada lapisan tersembunyi pada NN inv
```

```
    num_of_hidden1 = HIDDEN1;

    % Inisialisasi parameter
    alpha = ALPHA; % learning rate
    mu = MU; % konstanta momentum
    stop_flag = 0;
    stop_crit = 1e-4;
    max_epoch = EPOCH;
    E = zeros(1, train_size1);

    % Import bobot NN invers
    weights1 = load('weights_data1_inverse_stage2.mat');
    V0j1 = weights1.V0j;
    Vij1 = weights1.Vij;
    W0k1 = weights1.W0k;
    Wjk1 = weights1.Wjk;

    % Import bobot NN ID
    weights2 = load('weights_data1_stage2.mat');
    V0j2 = weights2.V0j;
    Vij2 = weights2.Vij;
    W0k2 = weights2.W0k;
    Wjk2 = weights2.Wjk;

    % Inisialisasi gradien bobot
    deltaVij1_old = zeros(col_input1, num_of_hidden1);
    deltaWjk1_old = zeros(num_of_hidden1, 1);

    % Training
    epoch = 1;
    while ~stop_flag && epoch <= max_epoch
        % Inisialisasi error dan batch
        n = 1;
        E(1, n) = 0;

        % Inisialisasi input/output NN inv
        X1 = zeros(1, col_input1);
        y1 = zeros(1, 1);

        % Inisialisasi input/output NN ID
        % NN ID menggunakan 5 input
        % Struktur input: y(k-1) | y(k-2) | u(k) | u(k-1) | u(k-2)
        X2 = zeros(1, 5);
        y2 = zeros(1, 1);

        for n = 1:train_size1
            % Feedforward NN inv
            X1(n, :) = input1(n, :);
            y1(n, :) = target1(n, :);
            Z_inj1 = V0j1 + X1(n, :) * Vij1;
            Zj1 = 2 ./ (1 + exp(-2*Z_inj1)) - 1; % fungsi aktivasi
tanh
```

```
            Y_ink1 = W0k1 + Zj1 * Wjk1;
            Yk1 = 2 ./ (1 + exp(-2*Y_ink1)) - 1; % fungsi aktivasi
tanh

            % FEEDBACK untuk NN inv
            % dengan Yk1 = u(k)
            input1(n+1, 5) = input1(n, 4); % u(k-1) di-pass ke u(k-2)
            input1(n+1, 4) = Yk1; % u(k) di-pass ke u(k-1)

            % Feedforward NN ID
            X2(n, :) = input2(n, :);
            X2(n, 3) = Yk1; % passing output dari NN inv ke neuron
u(k) NN ID
            y2(n, :) = target2(n, :); % target berasal dari input y(k)
NN inv
            Z_inj2 = V0j2 + X2(n, :) * Vij2;
            Zj2 = 2 ./ (1 + exp(-2*Z_inj2)) - 1; % fungsi aktivasi
tanh
            Y_ink2 = W0k2 + Zj2 * Wjk2;
            Yk2 = 2 ./ (1 + exp(-2*Y_ink2)) - 1; % fungsi aktivasi
tanh

            % Meneruskan input yang ter-delay
            X2(n+1, 5) = X2(n, 4); % u(k-1) di-pass ke u(k-2)
            X2(n+1, 4) = X2(n, 3); % u(k) di-pass ke u(k-1)

            % FEEDBACK untuk NN ID
            % dengan Yk2 = y(k)
            X2(n+1, 2) = X2(n, 1); % y(k-1) di-pass ke y(k-2)
            X2(n+1, 1) = Yk2; % y(k) di-pass ke y(k-1)

            % Backpropagation (NN ID tidak di-update, hanya NN inv)
            E(1, n) = 0.5 * sum((y2(n, :) - Yk2).^2); % MSE loss
function
            do_k1 = (y2(n, :) - Yk2) .* (1 - Yk1.^2);
            deltaWjk1 = alpha * Zj1' * do_k1;
            deltaW0k1 = alpha * do_k1;
            do_j1 = (do_k1 * Wjk1') .* (1 - Zj1.^2);
            deltaVij1 = alpha * X1(n, :)' * do_j1;
            deltaV0j1 = alpha * do_j1;

            % Update bobot, bias, dan gradien (delta) menggunakan
momentum
            Wjk1 = Wjk1 + deltaWjk1 + (mu * deltaWjk1_old);
            W0k1 = W0k1 + deltaW0k1;
            Vij1 = Vij1 + deltaVij1 + (mu * deltaVij1_old);
            V0j1 = V0j1 + deltaV0j1;
            deltaWjk1_old = deltaWjk1;
            deltaVij1_old = deltaVij1;
        end
        error(1, epoch) = sum(E)/train_size1;
        if mod(epoch, 50) == 0 % Print setiap 50 epoch
```

```
            fprintf('Epoch %d: \n', epoch);
            fprintf('Error rata-rata: %d\n', error(1, epoch));
        end
        if error(1, epoch) < stop_crit
            stop_flag = 1;
        end
        epoch = epoch + 1;
    end

    % Plot error terhadap epoch
    plot(1:size(error, 2), error)
    xlabel('Epoch')
    ylabel('Error rata-rata')

    % Menyimpan bobot
    save(strcat('weights_', file, '_DIC', '.mat'), 'V0j1', 'Vij1',
'W0k1', 'Wjk1', 'V0j2', 'Vij2', 'W0k2', 'Wjk2');
    disp('Proses training selesai!');
end
function train_DIC_linear(file, HIDDEN1, ALPHA, MU, EPOCH)
    % Import data
    % Struktur data: y(k) | y(k-1) | y(k-2) | y(k-3) | u(k) | u(k-1) |
    % u(k-2) | u(k-3)
    train_data = load(strcat(file, '_preprocessed.mat')).train_data;

    % Pemisahan input dengan target
    % Struktur input: y(k) | y(k-1) | y(k-2) | y(k-3) | u(k-1) | u(k-
2) | u(k-3)
    % Struktur output: u(k)
    input1 = train_data(:, [1 2 3 4 6 7 8]);
    target1 = train_data(:, 5);
    [train_size1, col_input1] = size(input1);
    input2 = train_data(:, 2:end);
    target2 = train_data(:, 1);

    % Inisialisasi jumlah neuron pada lapisan tersembunyi pada NN inv
    num_of_hidden1 = HIDDEN1;

    % Inisialisasi parameter
    alpha = ALPHA; % learning rate
    mu = MU; % konstanta momentum
    stop_flag = 0;
    stop_crit = 1e-4;
    max_epoch = EPOCH;
    E = zeros(1, train_size1);

    % Import bobot NN invers
    weights1 = load('weights_data2_more_inverse.mat');
    V0j1 = weights1.V0j;
    Vij1 = weights1.Vij;
    W0k1 = weights1.W0k;
    Wjk1 = weights1.Wjk;
```

```
    % Import bobot NN ID
    weights2 = load('weights_data2_more.mat');
    V0j2 = weights2.V0j;
    Vij2 = weights2.Vij;
    W0k2 = weights2.W0k;
    Wjk2 = weights2.Wjk;

    % Inisialisasi gradien bobot
    deltaVij1_old = zeros(col_input1, num_of_hidden1);
    deltaWjk1_old = zeros(num_of_hidden1, 1);

    % Training
    epoch = 1;
    while ~stop_flag && epoch <= max_epoch
        % Inisialisasi error dan batch
        n = 1;
        E(1, n) = 0;

        % Inisialisasi input/output NN inv
        X1 = zeros(1, col_input1);
        y1 = zeros(1, 1);

        % Inisialisasi input/output NN ID
        % NN ID menggunakan 7 input
        % Struktur input: y(k-1) | y(k-2) | y(k-3) | u(k) | u(k-1) |
u(k-2)
        % | u(k-3)
        X2 = zeros(1, 7);
        y2 = zeros(1, 1);

        for n = 1:train_size1
            % Feedforward NN inv
            X1(n, :) = input1(n, :);
            y1(n, :) = target1(n, :);
            Z_inj1 = V0j1 + X1(n, :) * Vij1;
            Zj1 = 2 ./ (1 + exp(-2*Z_inj1)) - 1; % fungsi aktivasi
tanh
            Y_ink1 = W0k1 + Zj1 * Wjk1;
            Yk1 = 2 ./ (1 + exp(-2*Y_ink1)) - 1; % fungsi aktivasi
tanh

            % FEEDBACK untuk NN inv
            % dengan Yk1 = u(k)
            input1(n+1, 7) = input1(n, 6); % u(k-2) di-pass ke u(k-3)
            input1(n+1, 6) = input1(n, 5); % u(k-1) di-pass ke u(k-2)
            input1(n+1, 5) = Yk1; % u(k) di-pass ke u(k-1)

            % Feedforward NN ID
            X2(n, :) = input2(n, :);
            X2(n, 4) = Yk1; % passing output dari NN inv ke neuron
u(k) NN ID
```

```
            y2(n, :) = target2(n, :); % target berasal dari input y(k)
NN inv
            Z_inj2 = V0j2 + X2(n, :) * Vij2;
            Zj2 = 2 ./ (1 + exp(-2*Z_inj2)) - 1; % fungsi aktivasi
tanh
            Y_ink2 = W0k2 + Zj2 * Wjk2;
            Yk2 = 2 ./ (1 + exp(-2*Y_ink2)) - 1; % fungsi aktivasi
tanh

            % Meneruskan input yang ter-delay
            X2(n+1, 7) = X2(n, 6); % u(k-2) di-pass ke u(k-3)
            X2(n+1, 6) = X2(n, 5); % u(k-1) di-pass ke u(k-2)
            X2(n+1, 5) = X2(n, 4); % u(k) di-pass ke u(k-1)

            % FEEDBACK untuk NN ID
            % dengan Yk2 = y(k)
            X2(n+1, 2) = X2(n, 1); % y(k-1) di-pass ke y(k-2)
            X2(n+1, 1) = Yk2; % y(k) di-pass ke y(k-1)

            % Backpropagation (NN ID tidak di-update, hanya NN inv)
            E(1, n) = 0.5 * sum((y2(n, :) - Yk2).^2); % MSE loss
function
            do_k1 = (y2(n, :) - Yk2) .* (1 - Yk1.^2);
            deltaWjk1 = alpha * Zj1' * do_k1;
            deltaW0k1 = alpha * do_k1;
            do_j1 = (do_k1 * Wjk1') .* (1 - Zj1.^2);
            deltaVij1 = alpha * X1(n, :)' * do_j1;
            deltaV0j1 = alpha * do_j1;

            % Update bobot, bias, dan gradien (delta) menggunakan
momentum
            Wjk1 = Wjk1 + deltaWjk1 + (mu * deltaWjk1_old);
            W0k1 = W0k1 + deltaW0k1;
            Vij1 = Vij1 + deltaVij1 + (mu * deltaVij1_old);
            V0j1 = V0j1 + deltaV0j1;
            deltaWjk1_old = deltaWjk1;
            deltaVij1_old = deltaVij1;
        end
        error(1, epoch) = sum(E)/train_size1;
        if mod(epoch, 50) == 0 % Print setiap 50 epoch
            fprintf('Epoch %d: \n', epoch);
            fprintf('Error rata-rata: %d\n', error(1, epoch));
        end
        if error(1, epoch) < stop_crit
            stop_flag = 1;
        end
        epoch = epoch + 1;
    end

    % Plot error terhadap epoch
    plot(1:size(error, 2), error)
    xlabel('Epoch')
```

```
    ylabel('Error rata-rata')

    % Menyimpan bobot
    save(strcat('weights_', file, '_DIC', '.mat'), 'V0j1', 'Vij1',
'W0k1', 'Wjk1', 'V0j2', 'Vij2', 'W0k2', 'Wjk2');
    disp('Proses training selesai!');
end
function test_DIC_nonlinear(file)
    % Import data
    % Struktur data: y(k) | y(k-1) | y(k-2) | u(k) | u(k-1) | u(k-2)
    test_data = load(strcat(file, '_preprocessed.mat')).test_data;

    % Pemisahan input dengan target
    % Struktur input: y(k) | y(k-1) | y(k-2) | u(k-1) | u(k-2)
    % Struktur output: u(k)
    test_input = test_data(:, [1 2 3 5 6]);
    test_target = test_data(:, 4);
    [test_size, col_test] = size(test_input);

    % Import bobot NN invers
    weights1 = load('weights_data1_inverse.mat');
    V0j1 = weights1.V0j;
    Vij1 = weights1.Vij;
    W0k1 = weights1.W0k;
    Wjk1 = weights1.Wjk;

    % Import bobot NN ID
    weights2 = load('weights_data1.mat');
    V0j2 = weights2.V0j;
    Vij2 = weights2.Vij;
    W0k2 = weights2.W0k;
    Wjk2 = weights2.Wjk;

    % NN inv
    X1 = zeros(1, col_test);
    y1 = zeros(1, 1);

    % NN ID menggunakan 5 input
    % Struktur input: y(k-1) | y(k-2) | u(k) | u(k-1) | u(k-2)
    X2 = zeros(1, 5);
    y2 = zeros(1, 1);
    yhat2 = zeros(1500, 1); % variabel untuk menyimpan output

    for n = 1:test_size
        % Feedforward NN inv
        X1(n, :) = test_input(n, :);
        y1(n, :) = test_target(n, :);
        Z_inj1 = V0j1 + X1(n, :) * Vij1;
        Zj1 = 2 ./ (1 + exp(-2*Z_inj1)) - 1; % fungsi aktivasi tanh
        Y_ink1 = W0k1 + Zj1 * Wjk1;
        Yk1 = 2 ./ (1 + exp(-2*Y_ink1)) - 1; % fungsi aktivasi tanh
```

```
        % FEEDBACK untuk NN inv
        % dengan Yk1 = u(k)
        test_input(n+1, 5) = test_input(n, 4); % u(k-1) di-pass ke
u(k-2)
        test_input(n+1, 4) = Yk1; % u(k) di-pass ke u(k-1)

        % Feedforward NN ID
        X2(n, 3) = Yk1; % passing output dari NN inv ke neuron u(k) NN
ID
        y2(n, :) = test_input(n, 1); % target berasal dari input y(k)
NN inv
        Z_inj2 = V0j2 + X2(n, :) * Vij2;
        Zj2 = 2 ./ (1 + exp(-2*Z_inj2)) - 1; % fungsi aktivasi tanh
        Y_ink2 = W0k2 + Zj2 * Wjk2;
        Yk2 = 2 ./ (1 + exp(-2*Y_ink2)) - 1; % fungsi aktivasi tanh

        % Meneruskan input yang ter-delay
        X2(n+1, 5) = X2(n, 4); % u(k-1) di-pass ke u(k-2)
        X2(n+1, 4) = X2(n, 3); % u(k) di-pass ke u(k-1)

        % FEEDBACK untuk NN ID
        % dengan Yk2 = y(k)
        X2(n+1, 2) = X2(n, 1); % y(k-1) di-pass ke y(k-2)
        X2(n+1, 1) = Yk2; % y(k) di-pass ke y(k-1)

        yhat2(n) = Yk2; % menyimpan untuk diplot
        E(1, n) = 0.5 * sum((X1(n, 1) - Yk2).^2); % MSE loss function
    end
    avg_error = sum(E)/test_size;
    disp("Pengujian terhadap test set");
    fprintf("\tMSE: %.5f\n", avg_error);

    % Plot hasil
    k = 1:800;
    plot(k, yhat2(k, :))
    hold on
    plot(k, X1(k, 1))
    hold off
    title('Plot hasil uji pada test set')
    legend('Keluaran NN DIC', 'Sinyal referensi')
end
function test_DIC_linear(file)
    % Import data
    % Struktur data: y(k) | y(k-1) | y(k-2) | y(k-3) | u(k) | u(k-1) |
    % u(k-2) | u(k-3)
    test_data = load(strcat(file, '_preprocessed.mat')).test_data;

    % Pemisahan input dengan target
    % Struktur input: y(k) | y(k-1) | y(k-2) | y(k-3) | u(k-1) | u(k-
2) |
    % u(k-3)
    % Struktur output: u(k)
```

```
    test_input = test_data(:, [1 2 3 4 6 7 8]);
    test_target = test_data(:, 5);
    [test_size, col_test] = size(test_input);

    % Import bobot NN invers
    weights1 = load('weights_data2_more_inverse.mat');
    V0j1 = weights1.V0j;
    Vij1 = weights1.Vij;
    W0k1 = weights1.W0k;
    Wjk1 = weights1.Wjk;

    % Import bobot NN ID
    weights2 = load('weights_data2_more.mat');
    V0j2 = weights2.V0j;
    Vij2 = weights2.Vij;
    W0k2 = weights2.W0k;
    Wjk2 = weights2.Wjk;

    % NN inv
    X1 = zeros(1, col_test);
    y1 = zeros(1, 1);

    % NN ID menggunakan 7 input
    % Struktur input: y(k-1) | y(k-2) | y(k-3) | u(k) | u(k-1) | u(k-
2) |
    % u(k-3)
    X2 = zeros(1, 7);
    y2 = zeros(1, 1);
    yhat2 = zeros(1500, 1); % variabel untuk menyimpan output

    for n = 1:test_size
        % Feedforward NN inv
        X1(n, :) = test_input(n, :);
        y1(n, :) = test_target(n, :);
        Z_inj1 = V0j1 + X1(n, :) * Vij1;
        Zj1 = 2 ./ (1 + exp(-2*Z_inj1)) - 1; % fungsi aktivasi tanh
        Y_ink1 = W0k1 + Zj1 * Wjk1;
        Yk1 = 2 ./ (1 + exp(-2*Y_ink1)) - 1; % fungsi aktivasi tanh

        % FEEDBACK untuk NN inv
        % dengan Yk1 = u(k)
        test_input(n+1, 7) = test_input(n, 6); % u(k-2) di-pass ke
u(k-3)
        test_input(n+1, 6) = test_input(n, 5); % u(k-1) di-pass ke
u(k-2)
        test_input(n+1, 5) = Yk1; % u(k) di-pass ke u(k-1)

        % Feedforward NN ID
        X2(n, 4) = Yk1; % passing output dari NN inv ke neuron u(k) NN
ID
        y2(n, :) = test_input(n, 1); % target berasal dari input y(k)
NN inv
```

```
        Z_inj2 = V0j2 + X2(n, :) * Vij2;
        Zj2 = 2 ./ (1 + exp(-2*Z_inj2)) - 1; % fungsi aktivasi tanh
        Y_ink2 = W0k2 + Zj2 * Wjk2;
        Yk2 = 2 ./ (1 + exp(-2*Y_ink2)) - 1; % fungsi aktivasi tanh

        % Meneruskan input yang ter-delay
        X2(n+1, 7) = X2(n, 6); % u(k-2) di-pass ke u(k-3)
        X2(n+1, 6) = X2(n, 5); % u(k-1) di-pass ke u(k-2)
        X2(n+1, 5) = X2(n, 4); % u(k) di-pass ke u(k-1)

        % FEEDBACK untuk NN ID
        % dengan Yk2 = y(k)
        X2(n+1, 2) = X2(n, 1); % y(k-1) di-pass ke y(k-2)
        X2(n+1, 1) = Yk2; % y(k) di-pass ke y(k-1)

        yhat2(n) = Yk2; % menyimpan untuk diplot
        E(1, n) = 0.5 * sum((X1(n, 1) - Yk2).^2); % MSE loss function
    end
    avg_error = sum(E)/test_size;
    disp("Pengujian terhadap test set");
    fprintf("\tMSE: %.5f\n", avg_error);

    % Plot hasil
    k = 1:1200;
    plot(k, yhat2(k, :))
    hold on
    plot(k, X1(k, 1))
    hold off
    title('Plot hasil uji pada test set')
    legend('Keluaran NN DIC', 'Sinyal referensi')
end
function test_DIC_retrain_nonlinear(file)
    % Import data
    % Struktur data: y(k) | y(k-1) | y(k-2) | u(k) | u(k-1) | u(k-2)
    test_data = load(strcat(file, '_preprocessed.mat')).test_data;

    % Pemisahan input dengan target
    % Struktur input: y(k) | y(k-1) | y(k-2) | u(k-1) | u(k-2)
    % Struktur output: u(k)
    test_input1 = test_data(:, [1 2 3 5 6]);
    test_target1 = test_data(:, 4);
    test_input2 = test_data(:, 2:end);
    test_target2 = test_data(:, 1);
    [test_size, col_test] = size(test_input1);

    % Import bobot NN invers
%     weights = load('weights_data1_DIC.mat');
    weights = load(strcat('weights_', file, '_DIC.mat'));
    V0j1 = weights.V0j1;
    Vij1 = weights.Vij1;
    W0k1 = weights.W0k1;
    Wjk1 = weights.Wjk1;
```

```
    % Import bobot NN ID
    V0j2 = weights.V0j2;
    Vij2 = weights.Vij2;
    W0k2 = weights.W0k2;
    Wjk2 = weights.Wjk2;

    % NN inv
    X1 = zeros(1, col_test);
    y1 = zeros(1, 1);

    % NN ID menggunakan 5 input
    % Struktur input: y(k-1) | y(k-2) | u(k) | u(k-1) | u(k-2)
    X2 = zeros(1, 5);
    y2 = zeros(1, 1);
    yhat2 = zeros(1500, 1); % variabel untuk menyimpan output

    for n = 1:test_size
        % Feedforward NN inv
        X1(n, :) = test_input1(n, :);
        y1(n, :) = test_target1(n, :);
        Z_inj1 = V0j1 + X1(n, :) * Vij1;
        Zj1 = 2 ./ (1 + exp(-2*Z_inj1)) - 1; % fungsi aktivasi tanh
        Y_ink1 = W0k1 + Zj1 * Wjk1;
        Yk1 = 2 ./ (1 + exp(-2*Y_ink1)) - 1; % fungsi aktivasi tanh

        % FEEDBACK untuk NN inv
        % dengan Yk1 = u(k)
        test_input1(n+1, 5) = test_input1(n, 4); % u(k-1) di-pass ke
u(k-2)
        test_input1(n+1, 4) = Yk1; % u(k) di-pass ke u(k-1)

        % Feedforward NN ID
        X2(n, :) = test_input2(n, :);
        X2(n, 3) = Yk1; % passing output dari NN inv ke neuron u(k) NN
ID
        y2(n, :) = test_target2(n, :); % target berasal dari input
y(k) NN inv
        Z_inj2 = V0j2 + X2(n, :) * Vij2;
        Zj2 = 2 ./ (1 + exp(-2*Z_inj2)) - 1; % fungsi aktivasi tanh
        Y_ink2 = W0k2 + Zj2 * Wjk2;
        Yk2 = 2 ./ (1 + exp(-2*Y_ink2)) - 1; % fungsi aktivasi tanh

        % Meneruskan input yang ter-delay
        X2(n+1, 5) = X2(n, 4); % u(k-1) di-pass ke u(k-2)
        X2(n+1, 4) = X2(n, 3); % u(k) di-pass ke u(k-1)

        % FEEDBACK untuk NN ID
        % dengan Yk2 = y(k)
        X2(n+1, 2) = X2(n, 1); % y(k-1) di-pass ke y(k-2)
        X2(n+1, 1) = Yk2; % y(k) di-pass ke y(k-1)
```

```
        yhat2(n) = Yk2; % menyimpan untuk diplot
        E(1, n) = 0.5 * sum((X1(n, 1) - Yk2).^2); % MSE loss function
    end
    avg_error = sum(E)/test_size;
    disp("Pengujian terhadap test set");
    fprintf("\tMSE: %.5f\n", avg_error);

    % Plot hasil
    k = 1:800;
    plot(k, yhat2(k, :))
    hold on
    plot(k, X1(k, 1))
    hold off
    title('Plot hasil uji pada test set')
    legend('Keluaran NN ID', 'Sinyal referensi')
end
function test_DIC_retrain_linear(file)
    % Import data
    % Struktur data: y(k) | y(k-1) | y(k-2) | y(k-3) | u(k) | u(k-1) |
    % u(k-2) | u(k-3)
    test_data = load(strcat(file, '_preprocessed.mat')).test_data;

    % Pemisahan input dengan target
    % Struktur input: y(k) | y(k-1) | y(k-2) | u(k-1) | u(k-2)
    % Struktur output: u(k)
    test_input1 = test_data(:, [1 2 3 4 6 7 8]);
    test_target1 = test_data(:, 5);
    test_input2 = test_data(:, 2:end);
    test_target2 = test_data(:, 1);
    [test_size, col_test] = size(test_input1);

    % Import bobot NN invers
    weights = load('weights_data2_more_DIC.mat');
    V0j1 = weights.V0j1;
    Vij1 = weights.Vij1;
    W0k1 = weights.W0k1;
    Wjk1 = weights.Wjk1;

    % Import bobot NN ID
    V0j2 = weights.V0j2;
    Vij2 = weights.Vij2;
    W0k2 = weights.W0k2;
    Wjk2 = weights.Wjk2;

    % NN inv
    X1 = zeros(1, col_test);
    y1 = zeros(1, 1);

    % NN ID menggunakan 5 input
    % Struktur input: y(k-1) | y(k-2) | y(k-3) | u(k) | u(k-1) | u(k-
2) |
    % u(k-3)
```

```
    X2 = zeros(1, 7);
    y2 = zeros(1, 1);
    yhat2 = zeros(1500, 1); % variabel untuk menyimpan output

    for n = 1:test_size
        % Feedforward NN inv
        X1(n, :) = test_input1(n, :);
        y1(n, :) = test_target1(n, :);
        Z_inj1 = V0j1 + X1(n, :) * Vij1;
        Zj1 = 2 ./ (1 + exp(-2*Z_inj1)) - 1; % fungsi aktivasi tanh
        Y_ink1 = W0k1 + Zj1 * Wjk1;
        Yk1 = 2 ./ (1 + exp(-2*Y_ink1)) - 1; % fungsi aktivasi tanh

        % FEEDBACK untuk NN inv
        % dengan Yk1 = u(k)
        test_input1(n+1, 7) = test_input1(n, 6); % u(k-2) di-pass ke
u(k-3)
        test_input1(n+1, 6) = test_input1(n, 5); % u(k-1) di-pass ke
u(k-2)
        test_input1(n+1, 5) = Yk1; % u(k) di-pass ke u(k-1)

        % Feedforward NN ID
        X2(n, :) = test_input2(n, :);
        X2(n, 4) = Yk1; % passing output dari NN inv ke neuron u(k) NN
ID
        y2(n, :) = test_target2(n, :); % target berasal dari input
y(k) NN inv
        Z_inj2 = V0j2 + X2(n, :) * Vij2;
        Zj2 = 2 ./ (1 + exp(-2*Z_inj2)) - 1; % fungsi aktivasi tanh
        Y_ink2 = W0k2 + Zj2 * Wjk2;
        Yk2 = 2 ./ (1 + exp(-2*Y_ink2)) - 1; % fungsi aktivasi tanh

        % Meneruskan input yang ter-delay
        X2(n+1, 7) = X2(n, 6); % u(k-2) di-pass ke u(k-3)
        X2(n+1, 6) = X2(n, 5); % u(k-1) di-pass ke u(k-2)
        X2(n+1, 5) = X2(n, 4); % u(k) di-pass ke u(k-1)

        % FEEDBACK untuk NN ID
        % dengan Yk2 = y(k)
        X2(n+1, 2) = X2(n, 1); % y(k-1) di-pass ke y(k-2)
        X2(n+1, 1) = Yk2; % y(k) di-pass ke y(k-1)

        yhat2(n) = Yk2; % menyimpan untuk diplot
        E(1, n) = 0.5 * sum((X1(n, 1) - Yk2).^2); % MSE loss function
    end
    avg_error = sum(E)/test_size;
    disp("Pengujian terhadap test set");
    fprintf("\tMSE: %.5f\n", avg_error);

    % Plot hasil
    k = 1:800;
    plot(k, yhat2(k, :))
```

```
    hold on
    plot(k, X1(k, 1))
    hold off
    title('Plot hasil uji pada test set')
    legend('Keluaran NN ID', 'Sinyal referensi')
end
```