BAB 5 IMPLEMENTASI

Pada bab ini akan dijelaskan mengenai implementasi sistem dan antarmuka berdasarkan perancangan sistem yang telah dibuat untuk melakukan implementasi algoritma *Extreme Machine Learning* untuk prediksi beban pemanasan dan pendinginan bangunan.

5.1 Implementasi Sistem

Berdasarkan perancangan sistem yang telah dilakukan pada Bab 4, selanjutnya akan dijelaskan mengenai implementasi sistem sesuai dengan perancangan yang telah dibuat. Imlementasi sistem pada penelitian ini dilakukan menggunakan Bahasa pemrograman C# dengan Microsoft Visual Studio 2013.

5.1.1 Implementasi Normalisasi Data

Implementasi normalisasi data merupakan proses yang bertujuan agar mendapatkan data dengan ukuran yang lebih kecil antara 0.1-0.99 yang mewakili data asli tanpa kehilangan karakteristik data itu sendiri. Metode yang digunakan dalam normalisasi data pada penelitian ini adalah normalisasi Min-Max. Inisialisasi normalisasi ditunjukkan dengan kode program pada Tabel 5.1

Tabel 5.1 Tabel Kode Program Normalisasi Data

No	Kode Program
1	`public double[,] normalisasi(double [,] data_normalisasi)`
2	`{`
3	`data_normalisasi = readDataGrid();`
4	`double BA = 0.99;`
5	`double BB = 0.1;`
6	`int RC = 0; int SA = 1; int WA = 2; int HL = 3; int CL = 4;`
7	`for (int z = 0; z < data_normalisasi.GetLength(0); z++)`
8	`{`
9	`data_normalisasi[z, RC]=(BA-BB)*(data_normalisasi[z, RC]/1)+BB;`
10	`}`
11	`for (int z = 0; z < data_normalisasi.GetLength(0); z++)`
12	`{`
13	`data_normalisasi[z, SA]=(BA-BB)*(data_normalisasi[z, SA]/1000)+BB;`
14	`}`
15	`for (int z = 0; z < data_normalisasi.GetLength(0); z++)`
16	`{`
17	`data_normalisasi[z, WA]=(BA-BB)*(data_normalisasi[z, WA]/1000)+BB;`
18	`}`
19	`for (int z = 0; z < data_normalisasi.GetLength(0); z++)`
20	`{`
21	`data_normalisasi[z, HL]=(BA-BB)*(data_normalisasi[z, HL]/100)+BB;`
22	`}`
23	`for (int z = 0; z < data_normalisasi.GetLength(0); z++)`
24	`{`
25	`data_normalisasi[z, CL]=(BA-BB)*(data_normalisasi[z, CL]/100)+BB;`
26	`}`
27	`return data_normalisasi;`
28	`}`

Penjelasan kode program untuk implementasi normalisasi data menggunakan Min-Max adalah sebagai berikut:

Baris 3 s.d 6	: Inisialisasi data_normalisasi, batas atas, batas bawah, dan indeks kolom fitur RC, SA, WA, serta HL dan CL.
Baris 7 s.d 10	: Proses perulangan untuk normalisasi fitur RC dengan nilai minimum 0 dan maksimum 1.
Baris 11 s.d 14	: Proses perulangan untuk normalisasi fitur SA dengan nilai minimum 0 dan maksimum 1000.
Baris 15 s.d 18	: Proses perulangan untuk normalisasi fitur WA dengan nilai minimum 0 dan maksimum 1000.
Baris 19 s.d 22	: Proses perulangan untuk normalisasi target prediksi HL dengan nilai minimum 0 dan maksimum 100.
Baris 23 s.d 26	: Proses perulangan untuk normalisasi target prediksi CL dengan nilai minimum 0 dan maksimum 100.
Baris 27	: Mengembalikan nilai data_normalisasi yang telah dinormalisasi.

5.1.2 Implementasi Inisialisasi Bobot *Input*

Implementasi inisialisasi bobot *input* merupakan proses yang bertujuan untuk membentuk matriks bobot input yang mempunyai *ordo hidden neuron* x *input layer*. Nilai yang ada dalam matriks bobot *input* diperoleh dari nilai acak dengan *range* -1 sampai 1. Implementasi inisialisasi bobot *input* ditunjukkan dengan kode program yang terdapat pada Tabel 5.2 berikut.

Tabel 5.2 Tabel Kode Program Inisialisasi Bobot *Input*

No	Kode Program
1	`public double[,] generate_bobot(int hidden_neuron, int input_layer)`
2	`{`
3	`  double[,] bobot_input = new double[hidden_neuron, input_layer];`
4	`  Random random = new Random();`
5	`  double bobot = 0;`
6	`  for (int i = 0; i < bobot_input.GetLength(0); i++)`
7	`  {`
8	`   for (int j = 0; j < bobot_input.GetLength(1); j++)`
9	`   {`
10	`    bobot = random.NextDouble() * (0.99 - -0.99) + -0.99;`
11	`    bobot_input[i, j] = System.Math.Round(bobot, 2);`
12	`   }`
13	`  }`
14	`  return bobot_input;`
15	`}`

Penjelasan kode program untuk implementasi inisialisasi bobot *input* adalah sebagai berikut:

Baris 1	: Fungsi generate_bobot menerima parameter masukan berupa hidden_neuron dan input_layer.

Baris 3 : Proses pembentukan matriks bobot_input dengan *ordo hidden neuron* x *input layer*.

Baris 4 s.d 5 : Inisialisasi objek *random* dan nilai bobot.

Baris 6 s.d 13 : Proses perulangan untuk mengisi matriks bobot_input dengan nilai yang telah diacak dari -1 sampai 1 serta membulatkan nilainya menjadi 2 angka di belakang koma.

Baris 14 : Mengembalikan nilai matriks bobot_input.

5.1.3 Implementasi Inisialisasi Bias

Implementasi Inisisalisasi bias merupakan proses yang bertujuan untuk membentuk matriks bias yang mempunyai *ordo* 1 x *hidden neuron*. Nilai yang ada dalam matriks bias diperoleh dari nilai acak dengan range 0 sampai 1. Implementasi inisialisasi bias ditunjukkan dengan kode program pada Tabel 5.3 berikut.

Tabel 5.3 Tabel Kode Program Inisialisasi Bias

No	Kode Program
1	`public double[,] generate_bias(int hidden_neuron)`
2	`{`
3	`  double[,] bias = new double[1, hidden_neuron];`
4	`  Random rand = new Random();`
5	`  double masukan;`
6	`  for (int i = 0; i < hidden_neuron; i++)`
7	`  {`
8	`    masukan = rand.NextDouble() * (0.99 – 0.01) + 0.01;`
9	`    bias[0, i] = Math.Abs(Math.Round(masukan, 2));`
10	`  }`
11	`  return bias;`
12	`}`

Penjelasan kode program untuk implementasi inisialisasi bobot *input* adalah sebagai berikut:

Baris 1 : Fungsi generate_bias menerima parameter masukan hidden_neuron.

Baris 3 : Proses pembentukan matriks bias dengan *ordo* 1 x *hidden neuron*.

Baris 4 s.d 5 : Inisialisasi objek *random* dan nilai masukan.

Baris 6 s.d 10 : Proses perulangan untuk mengisi matriks bias dengan nilai yang telah diacak dari 0 sampai 1 serta membulatkan nilainya menjadi 2 angka di belakang koma.

Baris 11 : Mengembalikan nilai matriks bias.

5.1.4 Implementasi Transpose Matriks

Implementasi transpose matriks merupakan proses yang bertujuan untuk membentuk matriks baru yang mempunyai ordo kolom matriks awal x baris matriks awal. Jadi pada intinya proses ini mengubah baris dan kolom matriks awal menjadi kolom dan baris pada matriks baru. Implementasi transpose matriks ditunjukkan dengan kode program pada Tabel 5.4

Tabel 5.4 Tabel Kode Program Transpose Matriks

No	Kode Program
1	`public double[,] matriks_transpose(double[,] matriks)`
2	`{`
3	`  double[,]matriks_transpose=new double[matriks.GetLength(1),`
4	`matriks.GetLength(0)];`
5	
6	`  for (int i = 0; i < matriks_transpose.GetLength(0); i++)`
7	`  {`
8	`   for (int j = 0; j < matriks_transpose.GetLength(1); j++)`
9	`   {`
10	`     matriks_transpose[i, j] = matriks[j, i];`
11	`   }`
12	`  }`
13	`  return matriks_transpose;`
14	`}`

Penjelasan kode program untuk implementasi transpose matriks adalah sebagai berikut:

Baris 1 :Fungsi matriks_transpose menerima parameter matriks masukan sebagai matriks.

Baris 3 :Proses pembentukan matriks baru dengan nama matriks_transpose dengan *ordo* kolom matriks masukan x baris matriks masukan.

Baris 6 s.d 12 : Proses perulangan untuk melakukan transpose matriks dengan cara mengubah baris dan kolom matriks masukan menjadi kolom dan baris baru yang disimpan pada matriks baru matriks_transpose.

Baris 13 : Mengembalikan nilai matriks_transpose.

5.1.5 Implementasi Perkalian Matriks

Implementasi perkalian matriks merupakan proses yang bertujuan untuk melakukan perhitungan perkalian dua buah matriks. Implementasi perkalian matriks ditunjukkan dengan kode program pada Tabel 5.5 berikut.

Tabel 5.5 Tabel Kode Program Perkalian Matriks

No	Kode Program
1	`public double[,]matriks_perkalian(double[,]MatriksA,double[,]MatriksB)`
2	`{`
3	`  double[,] matriks_hasil = new double[MatriksA.GetLength(0),`
4	`MatriksB.GetLength(1)];`
5	`  for (int i = 0; i < MatriksA.GetLength(0); i++)`
6	`  {`

```
7     for (int j = 0; j < MatriksB.GetLength(1); j++)
8     {
9      matriks_hasil[i, j] = 0;
10     for (int k = 0; k < MatriksA.GetLength(1); k++)
11     {
12      matriks_hasil[i,j]=matriks_hasil[i,j]+MatriksA[i,k]*MatriksB[k,j];
13      matriks_hasil[i, j] = Math.Round(matriks_hasil[i, j], 4);
14     }
15    }
16   }
17   return matriks_hasil;
18  }
```

Penjelasan kode program untuk implementasi perkalian matriks adalah sebagai berikut:

Baris 1 : Fungsi matriks_perkalian menerima parameter sebagai matriksA dan matriksB.

Baris 3 :Proses pembentukan matriks baru dengan nama matriks_hasil dengan *ordo* baris matriksA x kolom matriksB.

Baris 5 s.d 16 : Proses perulangan untuk melakukan perkalian matriks dengan cara mengalikan baris matriksA dengan kolom matriksB dan menyimpan hasil perkalian matriks pada matriks_hasil.

Baris 17 : Mengembalikan nilai matriks_hasil.

5.1.6 Implementasi Penjumlahan Matriks dengan Bias

Implementasi penjumlahan dengan bias merupakan proses untuk melakukan perhitungan penjumlahan matriks masukan dengan matriks bias. Implementasi ini dilakukan ketika akan melakukan perhitungan H_{init}. Implementasi penjumlahan dengan bias ditunjukkan dengan kode program pada Tabel 5.6 berikut.

Tabel 5.6 Tabel Kode Program Penjumlahan matriks dengan bias

No	Kode Program
1	`public double[,] ones_bias(double[,] H_init, double[,] bias)`
2	`{`
3	`  for (int i = 0; i < H_init.GetLength(0); i++ )`
4	`  {`
5	`   for (int j = 0; j < H_init.GetLength(1); j++)`
6	`   {`
7	`     H_init[i, j] = H_init[i, j] + bias[0, j];`
8	`   }`
9	`  }`
10	`  return H_init;`
11	`}`

Penjelasan kode program untuk implementasi penjumlahan matriks dengan bias adalah sebagai berikut:

Baris 1 : Fungsi ones_bias menerima parameter sebagai matriks H_init dan matriks bias.

Baris 3 s.d 9 : Proses perulangan untuk melakukan penjumlahan matriks H_init dengan matriks bias sehingga diperoleh matriks H_init.

Baris 10 : Mengembalikan nilai matriks H_init.

5.1.7 Implementasi *Training*

Implementasi *training* merupakan implementasi yang menunjukkan proses-proses yang ada dalam tahap *training* sebagaimana telah dijabarkan pada perancangan bab 4.

5.1.7.1 Implementasi Perhitungan H_{init}

Implementasi perhitungan H_{init} merupakan proses yang bertujuan untuk menghitung matriks H_{init} dengan cara melakukan perkalian matriks $X_{training}$ dengan matriks bobot masukan W^T dan menambahkan dengan matriks bias. Implementasi perhitungan H_{init} ditunjukkan dengan kode program pada Tabel 5.7 berikut.

Tabel 5.7 Tabel Kode Program Perhitungan H_{init}

No	Kode Program
1	`//Menghitung H_init`
2	`bobot_input_transpose = matriks_transpose(bobot_input);`
3	`H_init = matriks_perkalian(X_training, bobot_input_transpose);`
4	`H_init = ones_bias(H_init, bias);`

Penjelasan kode program untuk implementasi perhitungan H_{init} adalah sebagai berikut:

Baris 2 : Melakukan proses transpose matriks bobot dengan cara melakukan pemanggilan fungsi matriks_transpose dan memberikan matriks bobot_input sebagai matriks masukan untuk fungsi tersebut serta menerima kembalian fungsi transpose sebagai matriks bobot_input_transpose. Fungsi matriks_transpose telah dijabarkan pada Tabel 5.4.

Baris 3 : Melakukan proses perkalian matriks dengan cara melakukan pemanggilan fungsi matriks_perkalian dan memberikan matriks X_training dan bobot_input_transpose sebagai matriks masukan untuk fungsi tersebut serta menerima kembalian fungsi perkalian sebagai matriks H_init. Fungsi matriks_perkalian telah dijabarkan pada Tabel 5.5.

Baris 4 : Melakukan proses penjumlahan matriks dengan cara melakukan pemanggilan fungsi ones_bias dan memberikan matriks masukan H_init dan bias serta menerima kembalian fungsi sebagai matriks H_init. Fungsi ones_bias telah dijabarkan pada Tabel 5.6.

5.1.7.2 Implementasi Perhitungan *H*

Implementasi perhitungan *H* merupakan proses yang bertujuan untuk menghitung matriks keluaran *hidden layer* yakni matriks *H* menggunakan fungsi aktivasi *sigmoid biner*. Implementasi perhitungan H ditunjukkan dengan kode program pada Tabel 5.8 berikut.

Tabel 5.8 Tabel Kode Program Perhitungan *H*

No	Kode Program
1	`//Menghitung H`
2	`H = sigmoid_biner(H_init, H);`
3	
4	`public double[,] sigmoid_biner(double[,] H_init, double[,] H)`
5	`{`
6	`  for (int i = 0; i < H_init.GetLength(0); i++ )`
7	`  {`
8	`   for (int j = 0; j < H_init.GetLength(1); j++)`
9	`   {`
10	`     H[i, j] = 1 / (1 + Math.Exp(-H_init[i, j]));`
11	`   }`
12	`  }`
13	`  return H;`
14	`}`

Penjelasan kode program untuk implementasi perhitungan *H* adalah sebagai berikut:

Baris 2 : Melakukan proses perhitungan H dengan cara memanggil fungsi sigmoid_biner. Fungsi sigmoid_biner membutuhkan parameter masukan yaitu matriks H_init yang telah dihitung sebelumnya. Fungsi sigmoid_biner akan memberikan nilai kembalian yang akan disimpan pada matriks H_uji.

Baris 4 : Fungsi sigmoid_biner menerima parameter sebagai matriks H_init dan matriks H.

Baris 6 s.d 12 : Proses perulangan untuk melakukan perhitungan H menggunakan fungsi aktivasi *sigmoid biner*.

Baris 13 : Mengembalikan nilai matriks H.

5.1.7.3 Implementasi Perhitungan Invers Matriks

Implementasi perhitungan invers matriks merupakan proses yang bertujuan melakukan invers matriks menggunakan Operasi Baris Elementer (OBE) dengan cara mengubah matriks masukan menjadi matriks identitas sedangkan matriks identitas menjadi matriks baru sebagai hasil dari proses invers. Implementasi perhitungan invers matriks ditunjukkan dengan kode program pada Tabel 5.9 berikut.

Tabel 5.9 Tabel Kode Program Perhitungan *Invers* Matriks

No	Kode Program
1	`public double[,] inverse_matriks(double[,] matriksA)`
2	`{`
3	`  double[,]matriks=new double[matriksA.GetLength(0),`
4	`matriksA.GetLength(1)];`
5	`  double[,] matriks_invers = new double[matriks.GetLength(0),`
6	`matriks.GetLength(1)];`
7	
8	`  for (int i = 0; i < matriks.GetLength(0); i++)`
9	`  {`
10	`   for (int j = 0; j < matriks.GetLength(1); j++)`
11	`   {`
12	`    matriks[i, j] = matriksA[i, j];`
13	`   }`
14	`  }`
15	
16	`  for (int i = 0; i < matriks.GetLength(0); i++)`
17	`  {`
18	`   for (int j = 0; j < matriks.GetLength(1); j++)`
19	`   {`
20	`    if (i == j)`
21	`    {`
22	`     matriks_invers[i, j] = 1;`
23	`    }`
24	`    else`
25	`    {`
26	`     matriks_invers[i, j] = 0;`
27	`    }`
28	`   }`
29	`  }`
30	
31	`  for (int i = 0; i < matriks.GetLength(0); i++)`
32	`  {`
33	`   double t = matriks[i, i];`
34	`   for (int k = 0; k < matriks.GetLength(1); k++)`
35	`   {`
36	`    matriks_invers[i, k] = matriks_invers[i, k] / t;`
37	`    matriks[i, k] = matriks[i, k] / t;`
38	`   }`
39	
40	`   for (int j = 0; j < matriks.GetLength(1); j++)`
41	`   {`
42	`    double c = matriks[j, i];`
43	`    for (int l = 0; l < matriks.GetLength(1); l++)`
44	`    {`
45	`     if (i != j)`
46	`     {`
47	`      matriks_invers[j,l]=matriks_invers[j,l] - c*matriks_invers[i, l];`
48	`      matriks[j, l] = matriks[j, l] - c * matriks[i, l];`
49	`     }`
50	`    }`
51	`   }`
52	`  }`
53	`  return matriks_invers;`
54	`}`

Penjelasan kode program untuk implementasi perhitungan *invers* matriks adalah sebagai berikut:

Baris 1 : Fungsi inverse_matriks menerima parameter sebagai matriksA yang akan dilakukan proses invers.

Baris 3 s.d 5 :Inisialisasi matriks baru yaitu *matriks* dan matriks_invers. *Matriks* adalah matriks masukan yang akan diinvers sedangkan matriks_invers adalah matriks yang nantinya bernilai hasil proses invers.

Baris 8 s.d 14 : Proses perulangan untuk menyalin nilai matriks masukan matriksA ke matriks *matriks* agar proses invers tidak menghilangkan nilai matriks awal.

Baris 16 s.d 29 : Proses perulangan untuk membuat matriks_invers menjadi matriks identitas.

Baris 31 s.d 52 : Proses perulangan yang merupakan inti dari proses invers dalam fungsi ini yaitu menyelesaikan proses invers menggunakan metode OBE. Metode OBE menyandingkan matriks awal dengan matriks identitas selanjutnya melakukan serangkaian tahap perhitungan untuk menjadikan matriks awal menjadi matriks identitas sedangkan matriks identitas awal menjadi hasil proses invers.

Baris 53 : Proses mengembalikan nilai matriks_invers.

5.1.7.4 Implementasi Perhitungan Matriks H^+

Implementasi perhitungan matriks H^+ atau matriks *Moore Penrose Pseudo Invers* merupakan proses untuk melakukan perhitungan untuk mencari nilai matriks H^+ yang nantinya digunakan dalam perhitungan nilai $\hat{\beta}$. Implementasi perhitungan matriks H+ ditunjukkan dengan kode program pada Tabel 5.10 berikut.

Tabel 5.10 Tabel Kode Program Perhitungan H^+

No	Kode Program
1	//Menghitung H+
2	H_transpose = matriks_transpose(H);
3	_HT_H = matriks_perkalian(H_transpose, H);
4	HT_H = inverse_matriks(_HT_H);
5	H_plus = matriks_perkalian(HT_H, H_transpose);

Penjelasan kode program untuk implementasi perhitungan H^+ adalah sebagai berikut:

Baris 2 : Melakukan proses transpose matriks H dengan cara memanggil fungsi matriks_transpose dan memberikan nilai matriks H sebagai parameter masukan. Fungsi tersebut akan memberikan nilai kembalian dan

disimpan pada matriks H_transpose. Fungsi matriks_transpose telah dijabarkan pada Tabel 5.4.

Baris 3 : Melakukan proses perkalian matriks antara matriks H_transpose dengan matriks H menggunakan fungsi matriks_perkalian. Fungsi tersebut akan memberikan nilai kembalian dan disimpan pada matriks _HT_H. Fungsi matriks_perkalian telah ditunjukkan pada Tabel 5.5.

Baris 4 : Melakukan proses invers matriks _HT_H dengan cara memanggil fungsi inverse_matriks. Fungsi tersebut akan memberikan nilai kembalian dan disimpan pada matriks HT_H. Fungsi inverse_matriks telah ditunjukkan pada Tabel 5.9.

Baris 5 : Melakukan proses perkalian matriks antara matriks HT_H dengan matriks H_transpose menggunakan fungsi matriks_perkalian. Fungsi tersebut akan memberikan nilai kembalian dan disimpan pada matriks H_plus. Fungsi matriks_perkalian telah ditunjukkan pada Tabel 5.5.

5.1.7.5 Implementasi Perhitungan $\hat{\beta}$

Implementasi perhitungan $\hat{\beta}$ merupakan proses yang bertujuan untuk melakukan proses perhitungan $\hat{\beta}$ dengan cara mengalikan matriks H^+ dengan matriks $Y_{training}$. Hasil perkalian kedua matriks tersebut menghasilkan matriks bobot keluaran $\hat{\beta}$ yang akan digunakan pada tahap testing. Implementasi perhitungan $\hat{\beta}$ ditunjukkan dengan kode program pada Tabel 5.11 berikut.

Tabel 5.11 Tabel Kode Program Perhitungan $\hat{\beta}$

No	Kode Program
1	//Menghitung beta
2	beta = matriks_perkalian(H_plus, Y_training);

Penjelasan kode program untuk implementasi perhitungan $\hat{\beta}$ adalah sebagai berikut:

Baris 2 : Melakukan proses perkalian matriks antara matriks H_plus dengan matriks Y_training menggunakan fungsi matriks_perkalian. Fungsi tersebut akan memberikan nilai kembalian dan disimpan pada matriks beta. Matriks beta akan digunakan untuk tahap testing selanjutnya. Fungsi matriks_perkalian telah ditunjukkan pada Tabel 5.5.

5.1.8 Implementasi Testing

Imlementasi *testing* merupakan implementasi yang menunjukkan proses-proses yang ada dalam tahap *testing* sebagaimana telah dijabarkan pada perancangan bab 4.

5.1.8.1 Implementasi Perhitungan H_{init}

Implementasi perhitungan H_{init} merupakan proses yang bertujuan untuk menghitung matriks H_{init} dengan cara melakukan perkalian matriks $X_{testing}$ dengan matriks bobot masukan W^T dan menambahkan dengan matriks bias. Implementasi perhitungan H_{init} ditunjukkan dengan kode program pada Tabel 5.12 berikut.

Tabel 5.12 Tabel Kode Program Perhitungan H_{init}

No	Kode Program
1	`//Menghitung H_init`
2	`bobot_input_transpose = matriks_transpose(bobot_input);`
3	`H_init_uji = matriks_perkalian(X_uji, bobot_input_transpose);`
4	`H_init_uji = ones_bias(H_init_uji, bias);`

Penjelasan kode program untuk implementasi perhitungan H_{init} adalah sebagai berikut:

Baris 2 : Melakukan proses transpose matriks bobot dengan cara melakukan pemanggilan fungsi matriks_transpose dan memberikan matriks bobot_input sebagai matriks masukan untuk fungsi tersebut serta menerima kembalian fungsi transpose sebagai matriks bobot_input_transpose. Fungsi matriks_transpose telah dijabarkan pada Tabel 5.4.

Baris 3 : Melakukan proses perkalian matriks dengan cara melakukan pemanggilan fungsi matriks_perkalian dan memberikan matriks X_uji dan bobot_input_transpose sebagai matriks masukan untuk fungsi tersebut serta menerima kembalian fungsi perkalian sebagai matriks H_init_uji. Fungsi matriks_perkalian telah dijabarkan pada Tabel 5.5.

Baris 4 : Melakukan proses penjumlahan matriks dengan cara melakukan pemanggilan fungsi ones_bias dan memberikan matriks masukan H_init_uji dan bias serta menerima kembalian fungsi sebagai matriks H_init_uji. Fungsi ones_bias telah dijabarkan pada Tabel 5.6

5.1.8.2 Implementasi Perhitungan H

Implementasi perhitungan *H* merupakan proses yang bertujuan untuk menghitung matriks keluaran *hidden layer* yakni matriks *H* menggunakan fungsi aktivasi *sigmoid biner*. Implementasi perhitungan H ditunjukkan dengan kode program pada Tabel 5.13 berikut.

Tabel 5.13 Tabel Kode Program Perhitungan *H*

No	Kode Program
1	//Menghitung H
2	H_uji = sigmoid_biner(H_init_uji, H_uji);
3	
4	public double[,] sigmoid_biner(double[,] H_init, double[,] H)
5	{
6	for (int i = 0; i < H_init.GetLength(0); i++)
7	{
8	for (int j = 0; j < H_init.GetLength(1); j++)
9	{
10	H[i, j] = 1 / (1 + Math.Exp(-H_init[i, j]));
11	}
12	}
13	return H;
14	}

Penjelasan kode program untuk implementasi perhitungan *H* adalah sebagai berikut:

Baris 2 : Melakukan proses perhitungan H_uji dengan cara memanggil fungsi sigmoid_biner. Fungsi sigmoid_biner membutuhkan parameter masukan yaitu matriks H_init_uji yang telah dihitung sebelumnya. Fungsi sigmoid_biner akan memberikan nilai kembalian yang akan disimpan pada matriks H_uji.

Baris 4 : Fungsi sigmoid_biner menerima parameter sebagai matriks H_init dan matriks H.

Baris 6 s.d 12 : Proses perulangan untuk melakukan perhitungan H menggunakan fungsi aktivasi *sigmoid biner*.

Baris 13 : Mengembalikan nilai matriks H.

5.1.8.3 Implementasi Perhitungan $\hat{Y}$

Implementasi perhitungan $\hat{Y}$ merupakan proses yang bertujuan untuk melakukan perhitungan matriks *Y* prediksi $\hat{Y}$ dengan cara mengalikan matriks *H* dari tahap *testing* dengan matriks $\hat{\beta}$ dari tahap *training*. Implementasi perhitungan $\hat{Y}$ ditunjukkan dengan kode program pada Tabel 5.14 berikut.

Tabel 5.14 Tabel Kode Program Perhitungan $\hat{Y}$

No	Kode Program
1	//Menghitung Y_prediksi
2	Y_prediksi = matriks_perkalian(H_uji, beta);

Penjelasan kode program untuk implementasi perhitungan $\hat{Y}$ adalah sebagai berikut:

Baris 2 : Melakukan proses perkalian matriks antara matriks H_uji dengan matriks beta menggunakan fungsi matriks_perkalian. Fungsi tersebut akan memberikan nilai kembalian dan disimpan pada matriks Y_prediksi.

5.1.9 Implementasi Perhitungan MAPE

Implementasi perhitungan MAPE merupakan proses yang bertujuan untuk melakukan perhitungan nilai evaluasi MAPE dengan cara membandingkan hasil prediksi dengan data testing. Pada proses ini akan dihasilkan matriks MAPE yang terdiri dari MAPE untuk HL, MAPE untuk CL dan rata-rata MAPE. Implementasi perhitungan MAPE ditunjukkan dengan kode program pada Tabel 5.15 berikut.

Tabel 5.15 Tabel Kode Program Perhitungan MAPE

No	Kode Program
1	`//Menghitung MAPE`
2	`MAPE = hitung_MAPE(Y_prediksi, Y_uji, MAPE);`
3	
4	`public double[,] hitung_MAPE(double[,] Y_prediksi, double[,] Y_uji,`
5	`double[,] MAPE)`
6	`{`
7	`  double tempHL = 0;`
8	`  double tempCL = 0;`
9	`  double[,] temp = new double[Y_prediksi.GetLength(0),`
10	`Y_prediksi.GetLength(1)];`
11	`  for (int i = 0; i < Y_prediksi.GetLength(0); i++ )`
12	`  {`
13	`   for (int j = 0; j < Y_prediksi.GetLength(1); j++)`
14	`   {`
15	`    temp[i,j]=Math.Abs(((Y_prediksi[i,j]-Y_uji[i,j])/Y_uji[i, j]*100));`
16	`   }`
17	`   tempHL= tempHL + temp[i, 0];`
18	`   tempCL= tempCL + temp[i, 1];`
19	`  }`
20	`  MAPE[0, 0] = tempHL / Y_prediksi.GetLength(0);//nilai MAPE HL`
21	`  MAPE[0, 1] = tempCL / Y_prediksi.GetLength(0);//nilai MAPE CL`
22	`  MAPE[0, 2] = (MAPE[0, 0] + MAPE[0, 1]) / 2;   //nilai rata_rata MAPE`
23	`  return MAPE;`
24	`}`

Penjelasan kode program untuk implementasi perhitungan MAPE adalah sebagai berikut:

Baris 2 : Melakukan proses perhitungan MAPE dengan cara memanggil fungsi hitung_MAPE. Fungsi hitung_MAPE membutuhkan parameter masukan yaitu matriks Y_prediksi yang telah dihitung sebelumnya, Y_uji dan matriks MAPE. Fungsi hitung_MAPE akan memberikan nilai kembalian yang akan disimpan pada matriks MAPE.

Baris 4 s.d 5 : Fungsi hitung_MAPE menerima parameter sebagai matriks Y_prediksi, matriks Y_uji dan matriks MAPE.

Baris 7 s.d 10 : Inisialisasi variabel tempHL, tempCL dan matriks temp.

Baris 11 s.d 19 : Proses perulangan untuk melakukan perhitungan MAPE menggunakan rumus MAPE seperti yang telah dijelaskan pada persamaan 2.1.

Baris 20 : Proses perhitungan untuk mencari nilai MAPE HL dan disimpan di matriks MAPE elemen ke [0,0].

Baris 21 : Proses perhitungan untuk mencari nilai MAPE CL dan disimpan di matriks MAPE elemen ke [0,1].

Baris 22 : Proses perhitungan untuk mencari nilai rata-rata MAPE dan disimpan di matriks MAPE elemen ke [0,2].

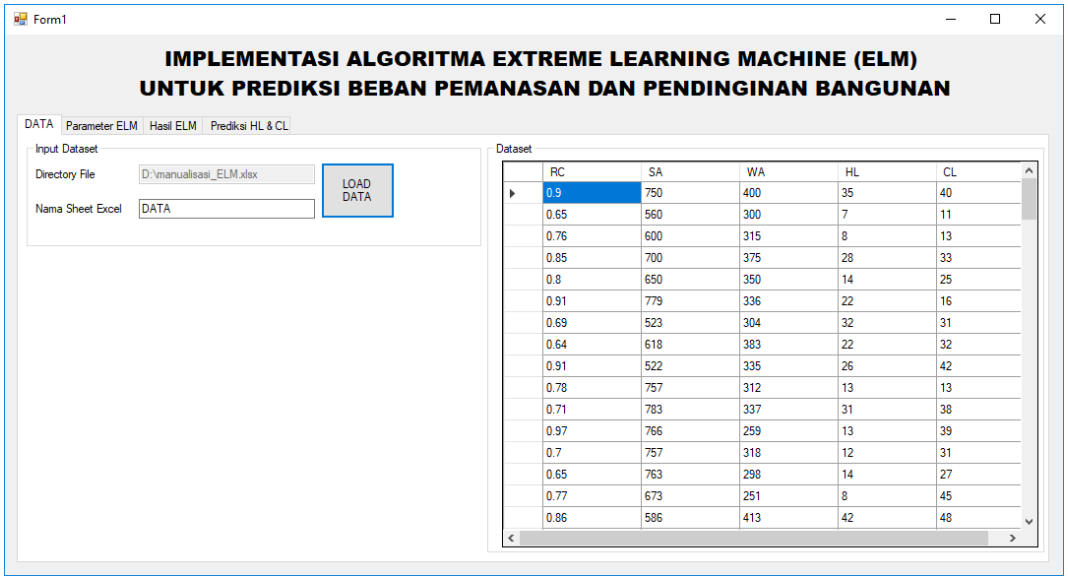
Baris 23 : Mengembalikan nilai matriks MAPE.

5.2 Implementasi Antarmuka

Implementasi antarmuka untuk implementasi algoritma *Extreme Learning Machine* untuk prediksi beban pemanasan dan pendinginan bangunan digunakan oleh pengguna sistem agar lebih mudah dalam berinteraksi secara langsung dengan sistem. Antarmuka sistem pada penelitian ini terdiri dari halaman data, parameter ELM, hasil ELM dan prediksi.

5.2.1 Antarmuka Halaman Data

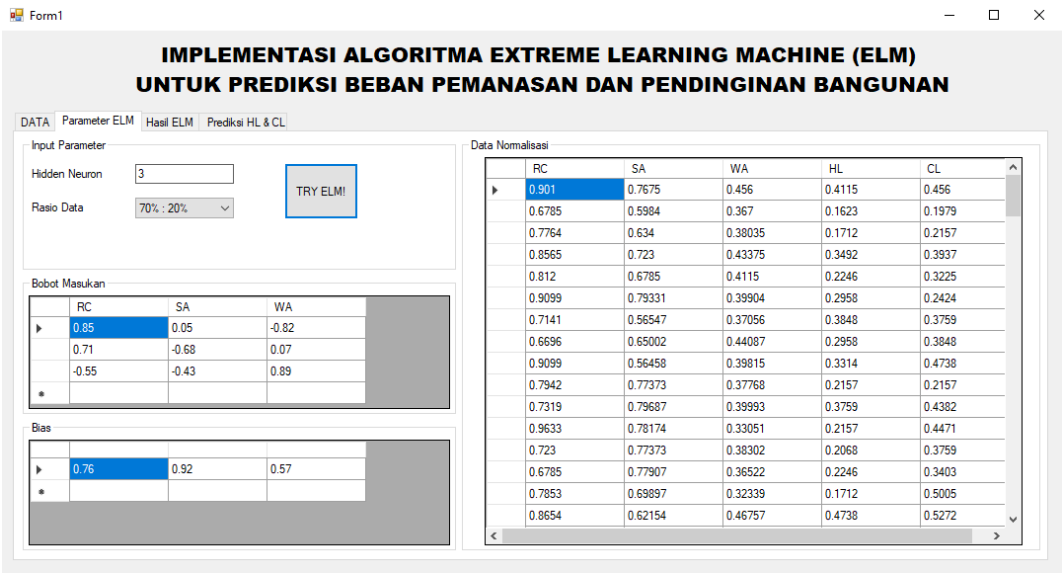
Antarmuka Halaman Data berisi halaman untuk memasukkan dataset yang berupa *file Microsoft Excel* (.xls / .xlsx). Selain itu terdapat *form* untuk mengisi nama *sheet* yang terdapat di *file Excel* tersebut. Pada halaman ini menampilkan tabel dataset yang telah dimuat untuk proses ELM. Antarmuka halaman data ditampilkan pada Gambar 5.1.



Gambar 5.1 Implementasi Halaman Data

5.2.2 Antarmuka Halaman Parameter ELM

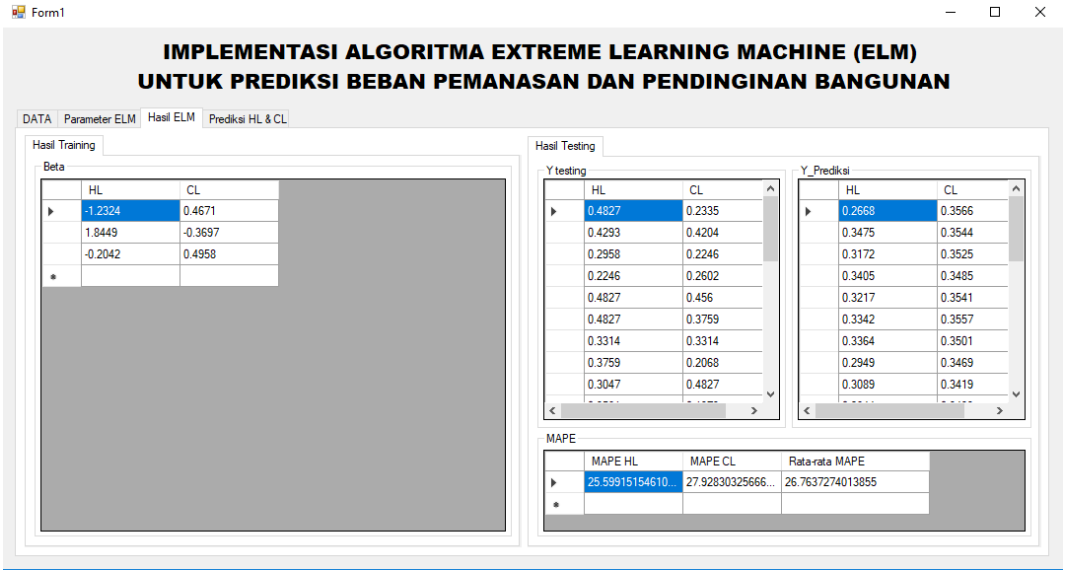
Antarmuka Halaman Parameter ELM berisi halaman untuk memasukkan parameter ELM yaitu jumlah *hidden neuron* dan rasio data latih dan uji. Pada halaman ini akan menampilkan data normalisasi, matriks bobot masukan *W* dan matriks bias setelah *button* Proses ELM ditekan. Implementasi halaman Parameter ELM dapat dilihat pada Gambar 5.2.



Gambar 5.2 Implementasi Halaman Parameter ELM

5.2.3 Antarmuka Halaman Hasil ELM

Antarmuka Halaman Hasil ELM berisi halaman yang menampilkan hasil fase *training*, hasil fase *testing* serta nilai MAPE. Pada hasil fase *training* akan ditampilkan matriks bobot keluaran $\hat{\beta}$. Sedangkan pada hasil fase *testing* akan ditampilkan hasil prediksi HL dan CL dari data *testing* serta matriks Y *testing*. Untuk hasil MAPE akan ditampilkan MAPE untuk HL, MAPE untuk CL dan rata-rata dari MAPE HL dan MAPE CL. Implementasi halaman Hasil ELM dapat dilihat pada Gambar 5.3.



Gambar 5.3 Implementasi Halaman Hasil ELM