BAB V IMPLEMENTASI

1.1 Implementasi Program

Implementasi program dimulai dengan pembentukan *class-class* yang berisi kode-kode program sehingga menghasilkan sebuah sistem. Pada implementasi ini *class-class* yang digunakan adalah class M*ain* dan FTS. *Class Main* berisi kode program untuk menampilkan data pengangguran dan pemanggilan method-method yang ada di *class-class* lain. *S*edangkan *class FTS* berisi proses *Fuzzy Time Series* dan Algoritme Genetika.

1.1.1 Implementasi Proses Pengambilan Data Pengangguran

Proses pengambilan data pengangguran dilakukan dengan pengambilan data di excel yang telah dibuat. Data yang diambil berupa data tahunan jumlah pengangguran di Jawa Timur. Pada implementasi ini juga menampilkan nilai minimum dan maksimum dari data yang digunakan. Potongan kode program tersebut ditunjukkan pada Gambar 5.1.

```
1  public void inputData() throws IOException,
2  BiffException {
3      File f = new File("D:\\radif\\proposal\\Data
4  Pengangguran.xls");
5      Workbook w = Workbook.getWorkbook(f);
6      Sheet sheet = w.getSheet(0);
7      sheetUmum = sheet;
8      baris = sheet.getRows();
9      tahun = new int[sheet.getRows() - 1];
10     dataPengangguran = new double[sheet.getRows() - 1];
11     selisih = new double[dataPengangguran.length];
12     bykData = sheet.getRows() - 1;
13     Cell cel = sheet.getCell(1, 1);
14     Cell cel2 = sheet.getCell(1, 2);
15     min = Double.valueOf(cel.getContents().replace(",",
16 ".")) - Double.valueOf(cel2.getContents().replace(",",
17 "."));
18     max = Double.valueOf(cel.getContents().replace(",",
19 ".")) - Double.valueOf(cel2.getContents().replace(",",
20 "."));
21     for (int j = 1; j < sheet.getRows(); j++) {
22      Cell data = sheet.getCell(1, j);
23      Cell data2 = sheet.getCell(1, j - 1);
24      Cell th = sheet.getCell(0, j);
25      dataPengangguran[j - 1] =
26 Double.valueOf(data.getContents().replace(",", "."));
27      tahun[j - 1] =
28 Integer.valueOf(th.getContents().replace(",", "."));
29      if (j == 1 ||
30 Double.valueOf(data2.getContents().replace(",", ".")) ==
```

```
0 || Double.valueOf(data.getContents().replace(",",
".")) == 0) {
     selisih[j - 1] = 0;
     } else {
      selisih[j - 1] =
Double.valueOf(data2.getContents().replace(",", ".")) -
Double.valueOf(data.getContents().replace(",", "."));
     if (min == 0) {
      min = selisih[j - 1];
     } else {
      if (min > selisih[j - 1] && selisih[j - 1] != 0) {
       min = selisih[j - 1];
      }
      if (max < selisih[j - 1] && selisih[j - 1] != 0) {
       max = selisih[j - 1];
      }
     }
    }
    System.out.println(tahun[j - 1] + "\t" +
formatter.format(dataPengangguran[j - 1]) + "\t" +
formatter.format(selisih[j - 1]));
        }
    }
```

Gambar 5.1 Implementasi Pengambilan Data

Keterangan :

1. Baris 3-7 merupakan proses memilih file pada direktori, kemudian membaca workboookknya dan memilih sheet yang digunakan.
2. Baris 14-21 merupakan proses set nilai minimum dan maksimum data.
3. Baris 30-51 merupakan proses menghitung nilai minimun dam maksimum menggunakan kondisi if-else.

1.1.2 Implementasi Inisialisasi Populasi Awal

Proses inisialisasi awal dilakukan dengan memberikan nilai tiap kromosom yang diambil. Pengambilan kromosom dilakukan secara random sebanyak data yang digunakan. Inisialisasi populasi awal diambil sebanyak nilai *popsize* yang telah dimasukkan. Potongan kode program ditunjukkan pada Gambar 5.2.

```
public void kromosomAwal() {
  for (int i = 0; i < popSize; i++) {
    for (int j = 0; j < kromosom; j++) {
       individu[i][j] = (Math.random() * ((max - min)) +
      min);
    }
  }
  for (int a = 0; a < popSize; a++) {
    for (int x = 0; x < kromosom; x++) {
      for (int i = 0; i < kromosom - 1; i++) {
        if (individu[a][i] > individu[a][i + 1]) {
```

```
12              double temp = individu[a][i];
13              individu[a][i] = individu[a][i + 1];
14              individu[a][i + 1] = temp;
15            }
16              populasi[0][a][i] = individu[a][i];
17          }
18              populasi[0][a][kromosom - 1] =
19          individu[a][kromosom - 1];
20          }
21      }
22  }
23  public void tampilIndividu() {
24      for (int a = 0; a < popSize; a++) {
25        System.out.print("Individu ke - " + a + " : ");
26      for (int b = 0; b < kromosom; b++) {
27        System.out.print(populasi[0][a][b] + "\t");
28      }
29        System.out.println("");
30      }
31  }
```

Gambar 5.2 Implementasi Inisialisasi Populasi Awal

Keterangan :

1. Baris 1-7 merupakan proses pengambilan nilai random sebanyak jumlah kromosom.
2. Baris 8-22 merupakan proses pengisian nilai kromosom dengan syarat nilai kanan tidak boleh lebih kecil dari nilai kiri. Pada proses ini dilakukan *sorting* menggunakan bubble short.

1.1.3 Implementasi Proses Crossover

Proses *crossover* merupakan proses pemilihan dua induk dari populasi awal yang kemudian dilakukan proses *crossover* antara dua induk yang terpilih. Proses *crossover* yang digunakan adalah *one-cut point*. Jumlah *offspring* yang dihasilkan dari proses ini adalah berdasarkan hasil *popsize* * *cr*. Potongan kode program tersebut ditunjukkan pada Gambar 5.3.

```
1   public void Crossover() {
2     int P1, P2, pointer;
3     int offspring = (int) Math.ceil(cr * popSize);
4     System.out.println("Offspring Crossover = " +
5   offspring);
6      int sisa = offspring % 2;
7      if (sisa == 1) {
8        offspring = offspring + 1;
9      }
10     int proses = offspring / 2;
11     double[][] child = new double[proses * 2][kromosom];
12     for (int i = 0, j = 0; i < proses; i++, j += 2) {
13       System.out.println("");
```

```
14        System.out.println("Crossover ke " + (i + 1));
15        P1 = random.nextInt((popSize - 1) + 1);
16        P2 = random.nextInt((popSize - 1) + 1);
17        pointer = random.nextInt((kromosom - 1) + 1) + 1;
18        System.out.println("p1 :" + P1);
19        System.out.println("p2 :" + P2);
20        System.out.println("pointer :" + pointer);
21        for (int k = 0; k < kromosom; k++) {
22           if (k >= pointer) {
23              child[j][k] = individu[P1][k];
24              child[j + 1][k] = individu[P2][k];
25           } else {
26              child[j][k] = individu[P2][k];
27              child[j + 1][k] = individu[P1][k];
28           }
29        }
30        System.out.println("Offspring dari crossover :");
31        for (int i = 0; i < proses*2; i++) {
32           for (int j = 0; j < kromosom; j++) {
33              System.out.print(child[i][j] + " ");
34           }
35              System.out.println();
36        }
37  }
```

Gambar 5.3 Implementasi *Crossover*

Keterangan :

1. Baris 3-4 merupakan proses penentuan jumlah *offspring* dengan mengalikan jumlah *popsize* dengan nilai *cr.*
2. Baris 6-10 merupakan proses pengecekan untuk jumlah *offspring,* jika jumlah *offspring* bernilai ganjil atau sisa 1 maka jumlah *offspring* ditambah 1.
3. Baris 15-17 merupakan proses pengambilan nilai untuk *parent1, parent2,* dan menentukan *pointer* keberapa yang akan dilakukan pertukaran atau *cutpoint* secara acak.
4. Baris 18-20 merupakan proses pemberian nilai pada kromosom tiap *parent.*
5. Baris 21-29 merupakan proses pertukaran antara dua induk berdasarkan posisi *pointer* yang ditentukan.

5.1.4 Implementasi Proses Mutasi

Proses mutasi yang digunakan pada program ini adalah menggunakan metode *random mutation* dengan memilih satu kromosom secara acak kemudian ditukar dengan nilai kromosom baru menggunakan rumus *random mutation*. Potongan program tersebut ditunjukkan pada Gambar 5.4.

```
1   public void Mutasi(){
2     int P1, P2, pointer1;
3     double r = -0.89;
4     r = random.nextDouble();
5     System.out.println("");
6     System.out.println("Proses Mutasi");
7     int offspring = (int) Math.ceil(mr * popSize);
8     for (int i = 0; i < offspring; i++) {
9        P1 = random.nextInt((popSize - 1) + 1);
10       pointer1 = random.nextInt((kromosom - 1) + 1) + 1;
11       System.out.println("p1 :" + P1);
12       System.out.println("pointer :" + pointer1);
13         for (int j = 0; j < kromosom; j++) {
14            min = individu[P1][0];
15            max = individu[P1][0];
16            if(max < individu[P1][j]){
17               max = individu[P1][j];
18            }
19            if (min > individu[P1][j]){
20            min = individu[P1][j];
21            }
22         }
23     System.out.println("max :" + max + "\nmin :" + min);
24     System.out.print("P1\t:");
25     for (int k = 0; k < kromosom; k++) {
26       System.out.print(individu[P1][k]+"\t");
27     }
28     int x = pointer1 - 1;
29       individu[P1][x] = (individu[P1][x] + (r * (max -
30  min)));
31       System.out.println("");
32       System.out.println("Hasil Mutasi :" +
33  (individu[P1][x]));
34     for (int k = 0; k < kromosom; k++) {
35       System.out.print(individu[P1][k] + "\t");
36       populasi[0][popSize + maxChildCrossover][k] =
37  individu[P1][k];
38             }
```

Gambar 5.4 Implementasi Mutasi

Keterangan :

1. Baris 1-4 adalah inisialisasi P1, pointer dan nilai r, dimana nilai r adalah nilai antara -0,1 sampai 0,1.
2. Baris 7 adalah menentukan banyaknya jumlah *offspring* berdasarkan jumlah *popsize* dikali nilai *mr.*
3. Baris 8-12 adalah proses mengisian nilai P1 dan pointer keberapa yang akan digunakan.
4. Baris 15-22 adalah proses menetukan nilai minimum dan maksimum dari individu yang dipilih, nilai minimum dan maksimum ini digunakan untuk mencari nilai mutasinya.

5. Baris 25-27 adalah proses pengambilan nilai kromosom berdasarkan individu yang dipilih.
6. Baris 29-38 adalah proses menghitung nilai mutasi dengan cara pointer pada individu yang dipilih ditambah nilai r kemudian dikalikan nilai maksimum dikurangi minimum.

5.1.5 Implementasi Menghitung Nilai *Fitness*

Perhitungan untuk memperoleh nilai *fitness* dilakukan dengan mencari nilai eror tiap individu yang diambil. Perhitungan nilai *fitness* dengan cara 1 dibagi nilai eror. Kode program tersebut ditunjukkan pada Gambar 5.5

```
1  public void Fitness() {
2     System.out.println("Fitness");
3     for (int i = 0; i < rmse.length; i++) {
4        System.out.print("ind " + i + ": ");
5        fitness[i] = 1 / rmse[i];
6        System.out.println(fitness[i]);
7     }
8        System.out.println();
9  }
```

Gambar 5.5 Implementasi Nilai *Fitness*

Keterangan :

1. Baris 3-9 merupakan baris untuk menghitung nilai *fitness* dengan cara 1 dibagi nilai RMSE. Proses menghitung nilai *fitness* dilakukan sebanyak jumlah RMSE yang dihasilkan.

5.1.6 Implementasi Proses Seleksi

Proses seleksi yang digunakan adalah menggunakan *elitism selection* yaitu mencari nilai *fitness* tertinggi.

```
1  public void Seleksi(int generasi) {
2    int individuBaru[][] = new int[popSize][kromosom];
3    for (int i = 0; i < populasi[0].length; i++) {
4    for (int j = 1; j < populasi[0].length; j++) {
5        if (fitness[j] > fitness[j - 1]) {
6        double temp = fitness[j];
7        fitness[j] = fitness[j - 1];
8        fitness[j - 1] = temp;
9        double temp3[] = populasi[generasi][j - 1];
10       populasi[generasi][j - 1] = populasi[generasi][j];
11       populasi[generasi][j] = temp3;
```

Gambar 5.6 Implementasi Seleksi

Keterangan :

1. Baris 1-4 adalah proses perulangan untuk tiap populasi
2. Baris 5-8 adalah proses *sorting* nilai *fitness*, nilai *fitness* di *sorting* dari nilai *fitness* terbesar.
3. Baris 9-11 adalah proses *sorting* untuk kromosom tiap individu.

5.1.7 Implementasi *Fuzzifikasi*

Proses *fuzzifikasi* adalah menentukan nilai rule dari tiap data yang digunakan, pada proses ini menggunakan nilai selisih dari data asli.

```
1  public void fuzzifikasi(double selisih[]) {
2     int nextState[] = new int[selisih.length];
3     int FLR[][][] = new int[popSize][selisih.length][2];
4     int index = 0; System.out.println(rule[0].length +
5  "*");
6     System.out.println("fuzzifikasi");
7     for (int k = 0; k < popSize + maxChildCrossover +
8  maxChildMutasi; k++) {
9           System.out.println("ind " + k);
10    for (int i = 0; i < selisih.length; i++) {
11        for (int j = 0; j < kromosomIn; j++) {
12          if (selisih[i] >= batasBawahIn[k][j] &&
13       selisih[i] <= batasAtasIn[k][j]) {
14             rule[k][i] = j;
15           }
16         }
17     }
```

Gambar 5.7 Implementasi Fuzzifikasi

Keterangan :

1. Baris 7-16 merupakan proses menetukan rule dari tiap data, dilakukan pengecekan sesuai interval tiap individu yang telah dibuat sebelumnya.

5.1.8 Implementasi FLR dan FLRG

Proses penetuan FLR dan FLRG dilakukan untuk menghitung nilai *defuzzifikasi*. Proses FLR adalah menentukan *current state* dan *next state.* Sedangkan FLRG adalah mengelompokkan nilai FLR yang sama. Potongan kode ditunjukkan pada Gambar 5.8.

```
1  for (int i = 0; i < selisih.length; i++) {
2     if (i != selisih.length - 1) {
3        nextState[i] = rule[k][i + 1];
4     }
5     for (int j = 0; j < selisih.length; j++) {
6        if (j > 1) {
7           FLR[k][j][0] = rule[k][j - 1];
8           FLR[k][j][1] = nextState[j - 1];
```

```
9         } else {
10           FLR[k][j][0] = 9;
11           FLR[k][j][1] = 9;
12        }
13      }
14      if (i != 0) {
15         System.out.println(formatter.format(selisih[i]) +
16  "\t" + rule[k][i] + "\t" + FLR[k][i][0] + "-->" +
17  FLR[k][i][1]);
18      } else {
19          System.out.println(formatter.format(selisih[i])
20  + "\t***\t" + FLR[k][i][0] + "-->" + FLR[k][i][1]);
21      }
22    }
          //FLRG
23    index = 0;
24    for (int o = 0; o < selisih.length; o++) {
25       if (cek(FLR[k][o][0], FLR[k][o][1], o, FLR[k])) {
26          FLRG[k][index][0] = FLR[k][o][0];
27          FLRG[k][index][1] = FLR[k][o][1];
28          index++;
29       }
30     }
31     indexrule[k] = index;
32     System.out.println("FLRG");
33     for (int i = 0; i < index; i++) {
34        System.out.println(FLRG[k][i][0] + "-->" +
35        FLRG[k][i][1]);
36     }
37     System.out.println("++++\n");
38   }
39   private boolean cek(int FLR1, int FLR2, int i, int
40  FLR[][]) {
41     boolean status = true;
42     for (int a = 0; a < i; a++) {
43        if ((FLR1 == (FLR[a][0])) && (FLR2 ==
44        (FLR[a][1]))) {
45            status = false;
46            break;
47         }
49      }
50      return status;
51  }
```

Gambar 5.8 Implementasi FLR dan FLRG

Keterangan :

1. Baris 1-4 adalah proses pengecekan jika data tidak sama dengan data sebelumnya maka next state adalah rule + 1.
2. Baris 5-21 adalah proses menentukan nilai FLR dari hasil fuzzifikasi. Dilakukan proses pengecekan terhadap datanya, jika nilai data lebih dari 1 maka current statenya adalah nilai fuzzifikasi sebelumnya dan nilai next statenya adalah nilai fuzzifikasi saat itu.

3. Baris 23-30 adalah prosses FLRG dengan melakukan pengecekan untuk FLRG pertama sama dengan FLR pertama, dilakukan sebanyak data yang digunakan.
4. Baris 39-51 merupakan proses pengecekan jika ada FLR yang sama maka tidak perlu dimasukkan lagi k FLRG.

5.1.9 Implementasi *Defuzzifikasi*

Proses *defuzzifikasi* dilakukan untuk memperoleh nilai selisih peramalan, nilai yang digunakan untuk *defuzzifikasi* adalah nilai FLRG dan nilai median tiap interval yang dibuat. Potongan program tersebut ditunjukkan pada Gambar 5.9.

```
1  public void defuzzifikasi() {
2     System.out.println("Defuzzifikasi");
3     for (int m = 0; m < popSize + maxChildCrossover +
4  maxChildMutasi; k++; m++) {
5     System.out.print("ind " + m + " :");
6     for (int i = 0; i < indexrule[m]; i++) {
7         if (FLRG[m][i][0] == rule[m][bykData - 1]) {
8            a.add(FLRG[m][i][1]);
9         }
10    }
11    double def = 0;
12    double maksder[] = new double[5];
13    double maks = 0;
14    ArrayList<Integer> tempatmaks = new ArrayList();
15    if (a.size() == 1) {
16       defuzzifikasi[m] = (batasAtasIn[m][a.get(0)] +
17       batasBawahIn[m][a.get(0)]) / 2;
18       System.out.println("Defuzzufikasi: " +
19       defuzzifikasi[m]);
20    } else if (a.size() > 1) {
21       for (int i = 0; i < maksder.length; i++) {
22          maks = derajatKeanggotaan[a.get(0)][i];
23       for (int j = 0; j < a.size(); j++) {
24          if (derajatKeanggotaan[a.get(j)][i] > maks) {
25             maks = derajatKeanggotaan[a.get(j)][i];
26          }
27       }
28          maksder[i] = maks;
29       }
30       maks = maksder[0];
31       for (int i = 0; i < maksder.length; i++) {
32          if (maksder[i] > maks) {
33             maks = maksder[i];
34          }
35       }
36       for (int i = 0; i < maksder.length; i++) {
37          if (maksder[i] == maks) {
38             tempatmaks.add(i);
39          }
40       }
```

```
41          double median[] = new double[tempatmaks.size()];
42          for (int i = 0; i < tempatmaks.size() - 1; i++) {
43             median[i] = (batasAtasIn[m][tempatmaks.get(i)]
44          + batasBawahIn[m][tempatmaks.get(i)]) / 2;
45          }
46          for (int i = 0; i < median.length; i++) {
47             def += median[i];
48          }
49          defuzzifikasi[m] = def / median.length;
50          System.out.println(def);
51              } else if (a.size() == 0) {
52                 defuzzifikasi[m] = 0;
53              }
54              a.clear();
55           }
56           System.out.println();
57       }
```

Gambar 5.9 Implementasi Defuzzifikasi

Keterangan :

1. Baris 1-10 proses pengecekan jika melakukan peramalan maka FLRG yang digunakan berdasarkan rule dari data terakhir.
2. Baris 14-28 adalah proses pengecekan jika terdapat 1 relasi grup maka nilai *defuzzifikasi*nya adalah nilai tengah yang memiliki nilai keanggotaan maksimum. Dan jika terdapat lebih dari 1 relasi grup maka dilakukan pengecekan lagi untuk mencari nilai maksimal derajat keanggotaannya.
3. Baris 29-34 proses cek maksimal di derajat keanggotaan.
4. Baris 35-44 adalah proses pengecekan jika ada nilai maksimal yang berurutan di derajat keanggotaan.
5. Baris 45-57 adalah proses untuk mencari nilai median dari tiap derajat keanggotaan yang memiliki nilai maksimal. Kemudian mencari nilai rata-rata dari median tersebut.

5.1.10 Implementasi Hasil Peramalan

Proses peramalan dilakukan untuk mencari nilai eror. Perhitungan nilai peramalan didapat dari nilai defuzzifikasi ditambah nilai selisih data yang akan diramal. Penjelasan kode program ditunjukkan pada Gambar 5.10.

```
1   public void peramalan(double[] selisih) {
2     System.out.println("Peramalan");
3     for (int i = 0; i < defuzzifikasi.length; i++) {
4        System.out.print("ind " + i + ": ");
5        peramalan[i] = defuzzifikasi[i] -
6   selisih[selisih.length - 1];
7        System.out.println(peramalan[i]);
8     }
9     System.out.println();
10  }
```

Gambar 5.10 Implementasi Hasil Peramalan

Keterangan :

1. Baris 1-8 merupakan proses menghitung hasil peramalan dengan cara hasil defuzzifikasi ditambah data yang akan diramal.

5.1.11 Implementasi Menghitung RMSE

Proses RMSE dilakukan untuk mencari nilai eror dari proses peramalan yang dilakukan. Proses mencari nilai eror menggunakan rumus akar dari nilai peramalan dikurangi data uji dipangkatkan kemudian dibagi sejumlah data uji. Potongan program ditunjukkan pada Gambar 5.11.

```
1  public void rmse() {
2     System.out.println("RMSE");
3     for (int i = 0; i < peramalan.length; i++) {
4        System.out.print("ind " + i + ": ");
5        rmse[i] = (Math.sqrt(Math.pow((peramalan[i] -
6        datauji[0]), 2)) / datauji.length);
7        System.out.println(rmse[i]);
8      }
9         System.out.println();
10 }
```

Gambar 5.11 Implementasi RMSE

Keterangan :

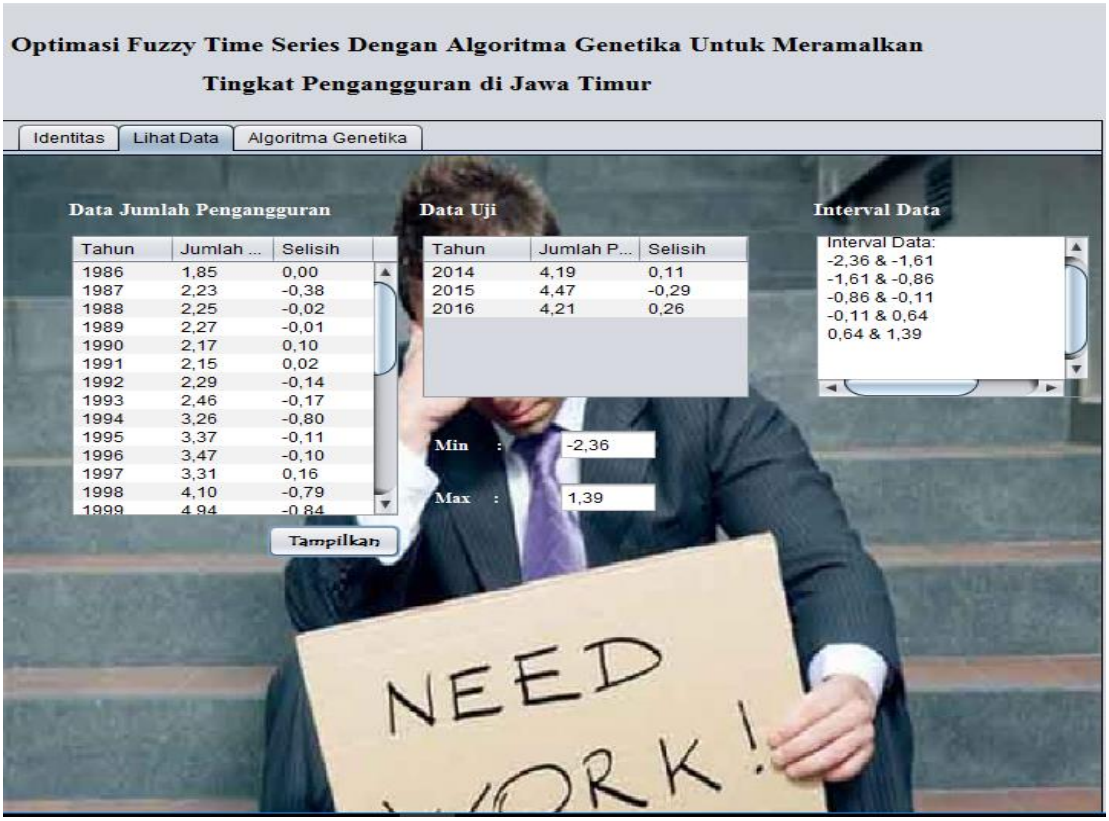
1. Baris 1-8 adalah proses mencari nilai RMSE dengan menggunakan rumus yang telah dijelaskan sebelumnya. Nilai RMSE dihitung sesuai jumlah peramalan yang dihasilkan.

5.2 Implementasi *User Interface*

Pada program yang telah dibuat ada 3 *user interface* yang digunakan, yaitu halaman utama, halaman lihat data dan halaman algoritme genetika.

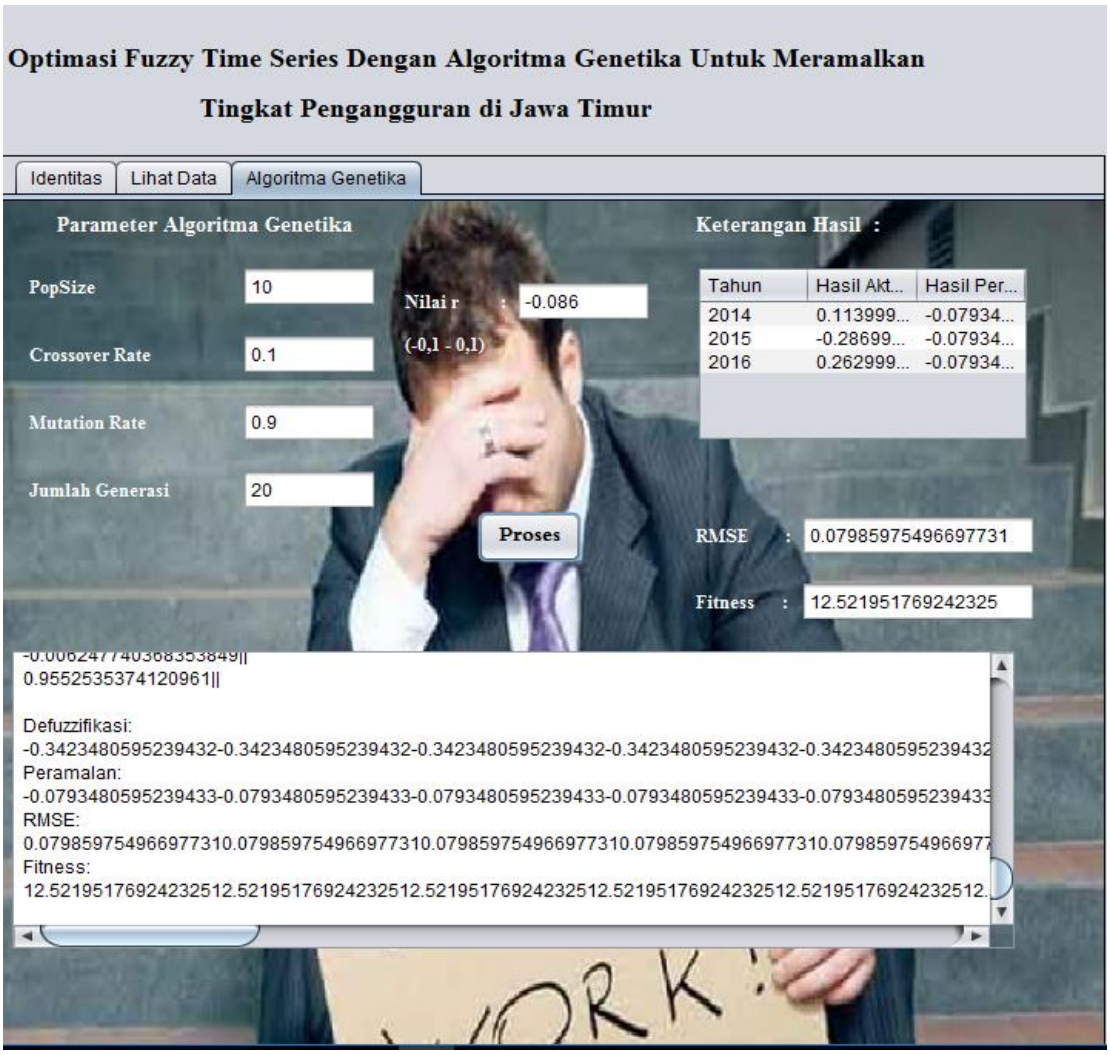
Gambar 5.12 Halaman Utama

Gambar 5.1 merupakan halaman utama yang hanya menampilkan identitas penulis.



Gambar 5.13 Halaman Lihat Data

Gambar 5.2 merupakan halaman lihat data yang menampilkan data pengangguran, data uji yang digunakan, nilai minimum dan maksimum dari data jumlah pengangguran dan nilai interval dari data pengangguran.



Gambar 5.14 Halaman Algoritme Genetika

Gambar 5.3 merupakan halaman algoritme genetika yang menampilkan hasil dari proses *Fuzzy Time Series* menggunakan Algoritme Genetika, pada halaman ini *user* diminta untuk memasukkan nilai *popsize, crossover rate, mutation rate*, jumlah generasi dan nilai r.