BAB 5 IMPLEMENTASI

5.1 Spesifikasi Lingkungan Implementasi

Perangkat lunak untuk prediksi konsumsi energi primer Indonesia ini diimplementasikan dan dijalankan menggunakan komputer dengan spesifikasi perangkat keras sebagai berikut:

1. Prosesor Intel Core i5 3210M 2.5GHz
2. RAM 4GB
3. Harddisk 500GB

Sedangkan perangkat lunak yang digunakan dalam implementasi yakni:

1. Sistem operasi Windows 10 64bit
2. Java Development Kit 8
3. IntelliJ IDEA 2017.1.4

5.2 Batasan Implementasi

Agar implemetasi mempunyai ruang lingkup yang jelas dan mempermudah melakukan pemrograman dari hasil perancangan algoritma maupun manualisasi, maka diperlukan beberapa batasan implementasi yakni:

1. Perangkat lunak dibuat dalam bentuk aplikasi desktop menggunakan bahasa pemrograman Java dan JavaFX (untuk GUI)
2. Data penelitian disimpan dalam database SQLite meliputi data tahun, konsumsi energi primer, GNI, GDP, populasi, impor, dan ekspor Indonesia antara tahun 1967 sampai 2016
3. Parameter yang bisa diubah oleh pengguna meliputi nilai *c1, c2, w, k velocity clamping*, q, ξ, max iterasi, min konvergen, *popSize*, m semut baru, ruang pencarian intersep dan koefisien kemiringan regresi
4. Metode yang digunakan yakni Hybrid *Particle Swarm Optimization* dan *Continuous Ant Colony Optimization* ($PSOACO_R$) dengan fungsi *fitness* berupa selisih absolut regresi linear berganda antara data aktual dan prediksi
5. Keluaran perangkat lunak berupa nilai MAPE pemodelan regresi linear dan hasil prediksi pada data uji

5.3 Implementasi Algoritma

Perangkat lunak yang telah dibuat memiliki dua bagian utama yakni inisialisasi dan iterasi. Proses inisialisasi mencakup lima tahapan penting yakni:

1. Inisialisasi kecepatan partikel
2. Inisialisasi tabel partikel
3. Inisialisasi pbest
4. Inisialisasi gbest
5. Hitung ω dan peluang p

sedangkan pada tahap iterasi terdapat tujuh tahapan mulai dari:

1. *Update* kecepatan partikel
2. *Update* posisi partikel
3. *Update* pbest tahap PSO
4. *Update* gbest tahap PSO
5. Buat semut baru
6. *Replace* partikel terburuk dan *update* pbest tahap ACO_R
7. *Update* gbest tahap ACO_R

Terakhir setelah proses pembentukan pemodelan regresi linear telah selesai sesuai dengan kriteria berhenti, maka hitung nilai prediksi konsumsi energi primer serta MAPE pada data uji.

5.3.1 Implementasi Inisialisasi Kecepatan Partikel

Inisialisasi kecepatan merupakan proses penentuan kecepatan awal partikel yang dalam implementasinya dilakukan dengan cara mengisi semua data array dua dimensi kecepatan dengan angka 0. Sesuai rancangan dari flowchart pada Gambar 4.2, maka implementasi inisialisasi kecepatan ditunjukkan pada Kode Program 5.1.

```
1  public void inisialisasiKecepatan() {
2      velocityClamping();
3      v = new double[popSize][dimensi];
4      for (int i = 0; i < popSize; i++) {
5          for (int d = 0; d < dimensi; d++) {
6              v[i][d] = 0;
7          }
8      }
9  }
```

Kode Program 5.1 Inisialisasi kecepatan partikel

Pada Kode Program 5.1 baris ke-4 sampai 8 adalah perulangan untuk mengisi array v (kecepatan) dua dimensi dengan angka 0. Untuk Kode Program 5.1 baris ke-2 merupakan kode untuk memanggil method *velocity clamping* yang akan mengacu pada Kode Program 5.2.

```
1  public void velocityClamping() {
2      vMin = new double[dimensi];
3      vMax = new double[dimensi];
4      for (int d = 0; d < dimensi; d++) {
5          vMax[d] = k * ((xMax[d] - xMin[d]) / 2);
6          vMin[d] = -vMax[d];
7      }
8  }
```

Kode Program 5.2 *Velocity clamping*

Velocity clamping diperlukan supaya menjaga keseimbangan antara *global exploration* dan *local exploitation* (Marini & Walczak, 2015). Proses *velocity clamping* sebenarnya dilakukan sebelum *update* kecepatan partikel pada tahap

iterasi, supaya proses tidak dilakukan berulang-ulang dan karena tujuan utamanya untuk menentukan $-v_{max}$ dan v_{max} yang nilainya tidak berubah sepanjang proses iterasi maka proses ini dapat ditaruh pada tahap inisialisasi.

Batas atas *velocity clamping* disimpan dalam array v_{max} dan batas bawah disimpan dalam array $-v_{max}$ (Kode Program 5.2 baris 2 sampai 3). Kode Program 5.2 baris ke-5 merupakan implementasi sesuai dengan Persamaan 2.4.

5.3.2 Implementasi Inisialisasi Partikel

Tabel Partikel merupakan tabel untuk menyimpan set solusi yang dalam PSO disebut tabel partikel dan dalam ACO disebut tabel *pheromone* (dalam ACO_R tetap disebut solusi). Berdasarkan perancangan algoritma dari flowchart Gambar 4.3, implementasi proses inisialisasi tabel partikel ditunjukkan pada Kode Program 5.3.

```
1  public void inisialisasiPartikel(){
2      x = new double[popSize][dimensi];
3      fitnessPartikel = new double[popSize];
4      for (int i = 0; i < popSize; i++) {
5          for (int d = 0; d < dimensi; d++) {
6              x[i][d] = xMin[d]+(rand.nextDouble()*(xMax[d]-xMin[d]));
7          }
8          fitnessPartikel[i] = rumus.hitungFitness(x[i], dataLatih);
9      }
10 }
```

Kode Program 5.3 Inisialisasi partikel

Tabel partikel diinisialisasi secara *uniformly distributed*, sehingga random yang dipakai pada pemrograman menggunakan *random.nextDouble* (Kode Program 5.3 baris ke-6). Pada Kode Program 5.3 baris ke-8 setiap selesai men-*generate* satu partikel maka dilakukan proses perhitungan *fitness* yakni selisih absolut regresi linear berganda antara nilai aktual dengan nilai prediksi. Fungsi *fitness* diimplementasikan pada Kode Program 5.4.

```
1  public double hitungFitness(double[] x, double[][] dataLatih) {
2      double fitness = 0;
3      for (int i = 0; i < dataLatih.length; i++) {
4          double nilaiAktual = dataLatih[i][1];
5          double nilaiPrediksi = x[0] + (x[1] * dataLatih[i][2]) +
6                  (x[2] * dataLatih[i][3]) + (x[3] * dataLatih[i][4]) +
7                  (x[4] * dataLatih[i][5]) + (x[5] * dataLatih[i][6]);
8          fitness += Math.pow((nilaiAktual - nilaiPrediksi), 2);
9      }
10     return fitness;
11 }
```

Kode Program 5.4 Hitung *fitness*

Kode Program 5.4 baris 3 sampai 8 adalah perhitungan nilai *fitness* seperti pada Persamaan 2.21 yakni nilai total selisih kuadrat antara nilai prediksi yang dihasilkan partikel ke-i dengan nilai aktual pada data latih.

5.3.3 Implementasi Inisialisasi Pbest

Pada inisialisasi pbest, nilai setiap elemen didalamnya identik dengan nilai partikel (tabel partikel) itu sendiri beserta *fitness*-nya, sehingga untuk inisialisasi pbest hanya perlu menyalin keseluruhan elemen partikel pada tabel partikel. Implementasi inisialisasi pbest ditunjukkan pada Kode Program 5.5.

```
1  public void inisialisasiPbest() {
2      pbest = new double[popSize][dimensi];
3      fitnessPbest = new double[popSize];
4      for (int i = 0; i < popSize; i++) {
5          pbest[i] = Arrays.copyOf(x[i], dimensi);
6          fitnessPbest[i] = fitnessPartikel[i];
7      }
8  }
```

Kode Program 5.5 Inisialisasi Pbest

Pada baris ke-5 pada Kode Program 5.5, penggunaan *Array.copyOf* diperlukan untuk menyalin isi array dua dimensi selain menggunakan perulangan *nested for* agar variabel yang berbeda tidak mereferensikan pada lokasi memori yang sama. Elemen yang disalin berupa posisi partikel (baris ke-5 Kode Program 5.5) dan *fitness* partikel (baris ke-6 Kode Program 5.5).

5.3.4 Implementasi Inisialisasi Gbest

```
1  public void inisilisasiGbest() {
2      gbest = pbest[0];
3      fitnessGbest = fitnessPbest[0];
4      for (int i = 1; i < popSize; i++) {
5          double[] tempGbest = pbest[i];
6          double tempFitnessGbest = fitnessPbest[i];
7          if (tempFitnessGbest < fitnessGbest) {
8              gbest = tempGbest;
9              fitnessGbest = tempFitnessGbest;
10         }
11     }
12 }
```

Kode Program 5.6 Inisialisasi Gbest

Inisialisasi *global best* atau gbest didapatkan dari pbest yang memiliki nilai *fitness* terbaik, dimana dalam permasalahan penelitian ini merupakan pbest dengan *fitness* terkecil. Dalam Kode Program 5.6, untuk mencari gbest maka dilakukan perbandingan satu persatu nilai *fitness* dimulai dari pbest pertama

dengan seluruh pbest lain, sampai pbest ke-i terakhir dengan cara melakukan perulangan *for* pada baris 4-11 Kode Program 5.6.

5.3.5 Implementasi Hitung ω dan Peluang p

Tujuan utama proses perhitungan ω yakni untuk menghitung nilai peluang p sebab ACO_R merupakan metode optimasi probabilistik (Socha & Dorigo, 2008). Besar tidaknya suatu solusi (partikel) terpilih sebagai acuan untuk membuat semut baru ditentukan oleh nilai ω ini, bukan berdasarkan nilai *fitness* masing-masing partikel. Nilai nilai ω bersifat tetap atau tidak berubah sepanjang proses iterasi. Sesuai rancangan flowchart pada Gambar 4.6 maka implementasi hitung ω ditunjukkan pada Kode Program 5.7.

```
1  public void hitungOmega(){
2      omega = new double[popSize];
3      for (int i = 0; i < popSize; i++) {
4          omega[i] = 1 / (q * popSize * Math.sqrt(2 * Math.PI)) *
5                  Math.exp(-(Math.pow(i, 2) / 2 * Math.pow(q, 2) *
6                          Math.pow(popSize, 2)));
7      }
8  }
```

Kode Program 5.7 Hitung ω

Untuk menampung nilai ω diperlukan array ω satu dimensi dengan besar sesuai ukuran popSize (Kode Program 5.7 baris ke-2). Pada Kode Program 5.7 baris ke-4 menunjukkan tidak perlu operasi pengurangan -1 karena perhitungan sudah dimulai dari index ke-0, sedangkan pada Persamaan 2.11 dimulai dari nilai 1 sehingga memerlukan pengurangan -1 agar nilainya 0.

```
1  public void hitungProb () {
2      double sumOmega = 0;
3      for (int i = 0; i < omega.length; i++){
4          sumOmega += omega[i];
5      }
6
7      prob = new double[popSize];
8      probCum = new double[popSize];
9      double tempSum = 0;
10     for (int i = 0; i < popSize; i++){
11         prob[i] = omega[i]/sumOmega;
12         tempSum += prob[i];
13         probCum[i] = tempSum;
14     }
15 }
```

Kode Program 5.8 Hitung peluang p

Langkah selanjutnya setelah menghitung yakni menghitung nilai p yang berfungsi untuk memilih partikel sebagai s_g dengan menggunakan *roulette*

wheel. Berdasarkan flowchart pada Gambar 4.6 implementasi hitung peluang p ditunjukkan dengan Kode Program 5.8.

Method hitung peluang p dimulai dengan menghitung jumlah total nilai ω yakni pada baris ke-3 sampai 5. Baris 10-14 merupakan implementasi dari Persamaan 2.12 yakni persamaan untuk menghitung peluang p.

5.3.6 Implementasi *Update* Kecepatan Partikel

Saat memasuki awal iterasi $PSOACO_R$ yang dihitung pertama kali adalah *update* kecepatan partikel. Sesuai dengan flowchart pada Gambar 4.7, maka *update* kecepatan partikel diimplementasikan pada Kode Program 5.9. Perubahan kecepatan ini akan digunakan partikel untuk berpindah posisi dalam ruang pencarian yang dipengaruhi oleh tiga parameter utama yakni pertama nilai ω atau inersia untuk mencegah partikel berpindah terlalu jauh dari posisi sebelumnya, kedua adalah *cognitive coefficient* atau c1 agar partikel cenderung berganti posisi disekitar pbest pada iterasi sebelumnya, dan ketiga berupa c2 atau *social coefficient* yang mempengaruhi partikel agar bergerak disekitar gbest iterasi sebelumnya (Marini & Walczak, 2015).

```
1  public void updateKecepatan() {
2      for (int i = 0; i < popSize; i++) {
3          for (int d = 0; d < dimensi; d++) {
4              double r1 = rand.nextDouble();
5              double r2 = rand.nextDouble();
6              v[i][d] = w*v[i][d] + c1c2[0]*r1*(pbest[i][d]-x[i][d])
7                      + c1c2[1]*r2*(gbest[d]-x[i][d]);
8              if (v[i][d] > vMax[d]){
9                  v[i][d] = vMax[d];
10             } else if (v[i][d] < vMin[d]){
11                 v[i][d] = vMin[d];
12             }
13         }
14     }
15 }
```

Kode Program 5.9 *Update* kecepatan partikel

Pada Kode Program 5.9 baris ke-5 dan 6, random r1 dan r2 di-*generate* secara *uniformly distributed* sehingga menggunakan *random*.*nextDouble*. Baris ke-6 merupakan implementasi persamaan *update* kecepatan partikel yakni Persamaan 2.3, sedangkan baris 8 sampai 12 merupakan implementasi dari *velocity clamping* seperti pada Persamaan 2.5.

5.3.7 Implementasi *Update* Posisi Partikel

Update posisi partikel merupakan penjumlahan posisi partikel iterasi sebelumnya dengan kecepatan baru yang telah di-*update*. Kode Program 5.10 merupakan implementasi *update* posisi berdasarkan flowchart pada Gambar 4.8.

Untuk mengontrol agar partikel baru tidak keluar dari ruang pencarian maka selain dilakukan *velocity clamping* dilakukan juga limit partikel menggunakan kode pada baris 5-9. Terakhir *fitness* partikel baru dihitung pada baris ke-11 setelah posisi semua dimensi pada satu partikel didalamnya sudah berganti posisi.

```
1  public void updatePosisi() {
2      for (int i = 0; i < popSize; i++) {
3          for (int d = 0; d < dimensi; d++) {
4              x[i][d] = x[i][d] + v[i][d];
5              if (s[i][d] > xMax[d]){
6                  x[i][d] = xMax[d];
7              } else if (s[i][d] < xMin[d]){
8                  x[i][d] = xMin[d];
9              }
10         }
11         fitnessPartikel[i] = rumus.hitungFitness(x[i], dataLatih);
12     }
13 }
```

Kode Program 5.10 *Update* posisi partikel

5.3.8 Implementasi *Update* Pbest Tahap PSO

Update pbest tahap PSO dilakukan dengan membandingkan semua nilai *fitness* pbest iterasi sebelumnya dengan *fitness* partikel baru dan dipilih yang terbaik. Berdasarkan Persamaan 2.8 maka *update* pbest tahap PSO diimplementasikan pada Kode Program 5.11.

```
1  public void updatePbest() {
2      for (int i = 0; i < popSize; i++) {
3          double tempPbestfitness = fitnessPartikel[i];
4          if (tempPbestfitness < fitnessPbest[i]) {
5              fitnessPbest[i] = tempPbestfitness;
6              pbest[i] = Arrays.copyOf(s[i], dimensi);
7          }
8      }
9  }
```

Kode Program 5.11 *Update* pbest tahap PSO

Pada kasus penelitian ini pencarian pbest tahap PSO dilakukan dengan membandingkan antara pbest lama dengan partikel baru yang sejajar dan dipilih yang mempunyai *fitness* terkecil, proses yang ditunjukkan pada Kode Program 5.11 baris ke-4 sampai 7.

5.3.9 Implementasi *Update* Gbest Tahap PSO

Terakhir siklus PSO setiap iterasi dengan meng-*update* gbest dengan melakukan perbandingan antara *fitness* gbest iterasi sebelumnya dengan *fitness* pbest terbaik (Kode Program 5.12 baris 4-7).

```
1  public void updateGbest() {
2      for (int i = 0; i < popSize; i++) {
3          double tempGbestfitness = fitnessPbest[i];
4          if (tempGbestfitness < fitnessGbest) {
5              fitnessGbest = tempGbestfitness;
6              gbest = Arrays.copyOf(pbest[i], dimensi);
7          }
8      }
9  }
```

Kode Program 5.12 *Update* gbest tahap PSO

5.3.10 Implementasi Buat Semut Baru

```
1  public void urutPartikel() {
2      xTemp = new double[popSize][dimensi];
3      fitnessPartikelTemp = new double[popSize];
4      for (int i = 0; i < popSize; i++) {
5          xTemp[i] = Arrays.copyOf(x[i], dimensi);
6          fitnessPartikelTemp[i] = fitnessPartikel[i];
7      }
8
9      for (int i = 0; i < xTemp.length; i++) {
10         for(int j = i + 1; j < xTemp.length; j++) {
11             if (fitnessPartikelTemp[j] < fitnessPartikelTemp[i]) {
12                 double tempFitness = fitnessPartikelTemp[i];
13                 fitnessPartikelTemp[i] = fitnessPartikelTemp[j];
14                 fitnessPartikelTemp[j] = tempFitness;
15
16                 double[] tempSolution = Arrays.copyOf(xTemp[i], dimensi);
17                 xTemp[i] = Arrays.copyOf(xTemp[j], dimensi);
18                 xTemp[j] = Arrays.copyOf(tempSolution, dimensi);
19             }
20         }
21     }
22 }
```

Kode Program 5.13 Urutkan partikel secara sementara

Sebelum membuat semut baru, tabel partikel harus di-*sorting* (diurutkan) terlebih dahulu. Setelah tabel partikel diurutkan secara sementara maka untuk setiap banyak m semut dilakukan seleksi dengan menggunakan metode *roulette wheel.* Jika sudah memilih partikel maka lakukan perhitungan sigma perdimensi

semut baru, hitung semut baru, dan terakhir hitung *fitness* semut baru tersebut. Proses pengurutan partikel diurutkan dari partikel terbaik yakni yang mempunyai *fitness* terkecil, proses pengurutan tersebut diimplementasikan pada Kode Program 5.13.

Proses pengurutan partikel secara sementara dilakukan dengan cara menyalin posisi partikel beserta *fitness*-nya saat itu kedalam array baru yakni array *xTemp* untuk menyimpan posisi partikel dan array *fitnessPartikelTemp* untuk menyimpan *fitness*-nya. Proses penyalinan nilai tersebut menggunakan Kode Program 5.13 baris ke-4 sampai 7. Saat semua partikel sudah disalin lakukan proses *sorting* menggunakan Kode Program 5.13 baris ke-9 sampai 21 dari *fitness* terkecil ke terbesar. Metode yang dipakai untuk proses pengurutan menggunakan *selection sort*. Metode selection sort bekerja dengan cara mencari yang terkecil dahulu sebelum dilakukan proses *swap* (tukar posisi).

```
1  public double[] rouletteWheel(){
2      double random = rand.nextDouble();
3      int rouletteIndex = 0;
4      for (int i = 0; i < popSize; i++){
5          if (random <= probCum[i]){
6              rouletteIndex = i;
7              break;
8          }
9      }
10     double[] xTerpilih = Arrays.copyOf(xTemp[rouletteIndex],dimensi);
11     return xTerpilih;
12 }
```

Kode Program 5.14 Roulette wheel untuk memilih partikel

```
1  public double[] hitungSigma(double[] seleksi){
2      double[] selisih = new double[dimensi];
3      double[] sigma = new double[dimensi];
4      for (int d = 0; d < dimensi; d++) {
5          for (int i = 0; i < popSize; i++){
6              selisih[d] += (Math.abs(x[i][d]-seleksi[d]))/(popSize-1);
7          }
8          sigma[d] = xi * selisih[d];
9      }
10     return sigma;
11 }
```

Kode Program 5.15 Hitung sigma

Setelah tabel partikel sudah diurutkan secara sementara, maka selanjutnya dilakukan proses pemilihan partikel yang menjadi acuan bagi semut baru yakni mencari s_g dengan menggunakan *roulette wheel*. Proses pemilihan ini tahap pertamanya dengan men-*generate* random *uniformly distributed* dengan nilai antara 0-1, kemudian menbandingkan dengan nilai peluang kumulatif. Jika angka

random lebih kecil atau sama dengan peluang kumulatif partikel ke-i, maka partikel ke-i terpilih sebagai s_g. Sesuai dengan rancangan flowchart Gambar 4.12, maka implementasi roulette wheel ditunjukkan pada Kode Program 5.14.

Hitung nilai σ_g^d masing-masing semut baru sesuai dengan Persamaan 2.15. Implementasi hitung σ_g^d ditunjukkan pada Kode Program 5.15.

```
1  public void buatSemutBaru() {
2     urutPartikel();
3     semutBaru = new double[m][dimensi];
4     fitnessSemutBaru = new double[m];
5     for(int i = 0; i < m; i++){
6        double[] xTerpilih = rouletteWheel();
7        double[] sigma = hitungSigma(xTerpilih);
8        for(int d = 0; d < dimensi; d++){
9           semutBaru[i][d]=xTerpilih[d]+rand.nextGaussian() * sigma[d];
10          if (semutBaru[i][d] > xMax[d]){
11              semutBaru[i][d] = xMax[d];
12          } else if (semutBaru[i][d] < xMin[d]){
13              semutBaru[i][d] = xMin[d];
14          }
15       }
16    fitnessSemutBaru[i]=fitness.fitnessLinear(semutBaru[i],dataLatih);
17    }
18 }
```

Kode Program 5.16 Buat semut baru

Terakhir, untuk membuat semut baru, *generate random* Z yang bersifat *normally distributed* dengan menggunakan *random.nextGaussian* yang ditunjukkan baris ke-9 pada Kode Program 5.16 dan lakukan operasi perhitungan seperti pada Persamaaan (2.14. Agar dimensi semut baru yang telah dibuat tidak keluar dari ruang pencarian, maka diperlukan limit semut baru yakni pada Kode Program 5.16 baris 10-14. Hitung *fitness* semut baru jika nilai semua dimensi semut baru telah dibuat (Kode Program 5.16 baris 16).

5.3.11 Implementasi Replace Partikel dan *Update* Pbest Tahap ACO_R

Jika *fitness* semut baru yang dihasilkan lebih baik daripada *fitness* partikel terburuk maka lakukan *replace* partikel terburuk tersebut dengan semut baru dan lakukan juga *update* pbest tahap ACO_R hanya untuk pbest yang sejajar dengan partikel terburuk tersebut. Implementasi replace partikel dan *update* pbest tahap ACO_R diimplementasikan dalam Kode Program 5.17 sesuai dengan Persamaan 2.17 untuk replace partikel dan Persamaan 2.18 untuk *update* pbest tahap ACO_R.

Sebelum proses replace partikel terburuk dan *update* pbest tahap ACO_R, dilakukan terlebih dahulu proses pencarian partikel terburuk. Partikel terburuk yang dimaksud adalah partikel yang mempunyai *fitness* terbesar diantara *fitness*

partikel lainnya. Untuk mencari partikel terburuk maka diimplementasikan Kode Program 5.18 yakni dengan cara membandingkan nilai *fitness* masing-masing partikel dan dicari yang paling besar nilai *fitness*-nya dimana dalam kasus global minimum seperti pada penelitian ini partikel dengan *fitness* terbesar adalah partikel terburuk. Hasil kembalian dari method cariPartikelTerburuk seperti yang ditunjukkan pada Kode Program 5.18 baris ke-10 berupa *index* atau posisi dimana partikel terburuk tersebut berada dalam array.

```
1  public void replacePartikelPbestACOR(){
2      for(int i = 0; i < m; i++) {
3          int xTerburuk = cariPartikelTerburuk();
4          if (fitnessSemutBaru[i] < fitnessPartikel[xTerburuk]) {
5              x[xTerburuk] = Arrays.copyOf(semutBaru[i], dimensi);
6              fitnessPartikel[xTerburuk] = fitnessSemutBaru[i];
7
8              double tempPbestfitness = fitnessSemutBaru[i];
9              if (tempPbestfitness < fitnessPbest[xTerburuk]) {
10                 fitnessPbest[xTerburuk] = tempPbestfitness;
11                 pbest[xTerburuk]=Arrays.copyOf(x[xTerburuk],dimensi);
12             }
13         }
14     }
15 }
```

Kode Program 5.17 Replace partikel terburuk dan *update* pbest tahap ACO_R

```
1  public int cariPartikelTerburuk() {
2      int indexTerburuk = 0;
3      double tempFitnessTerburuk = fitnessPartikel[0];
4      for (int i = 1; i < popSize; i++) {
5          if (tempFitnessTerburuk < fitnessPartikel[i]){
6              tempFitnessTerburuk = fitnessPartikel[i];
7              indexTerburuk = i;
8          }
9      }
10     return indexTerburuk;
11 }
```

Kode Program 5.18 Cari partikel terburuk

5.3.12 Implementasi *Update* Gbest Tahap ACO_R

Proses *update* gbest tahap ACO_R dilakukan jika semut baru mempunyai *fitness* lebih baik daripada gbest yang ditemukan oleh *update* gbest tahap PSO. Kode Program 5.19 diimplementasikan berdasarkan Persamaan 2.19.

```
1  public void updateGbestACOR(int iterasi){
2      for(int i = 0; i < m; i++) {
3          if (fitnessSemutBaru[i] < fitnessGbest) {
4              fitnessGbest = fitnessSemutBaru[i];
5              gbest = Arrays.copyOf(semutBaru[i], dimensi);
6          }
7      }
8      fitnessIterasi[iterasi] = fitnessGbest;
9  }
```

Kode Program 5.19 *Update* gbest tahap ACO_R

5.3.13 Implementasi Hitung Prediksi dan MAPE pada Data Uji

Implementasi hitung prediksi dan hitung MAPE (Kode Program 5.20) dilakukan setelah proses pencarian pemodelan regresi linear terbaik menggunakan PSOACOR telah selesai. Proses hitung prediksi dan hitung MAPE keduanya dihitung pada data uji yang tidak pernah dimasukkan pada data latih.

```
1  public double[][] hitungPrediksi(double[] x, double[][] dataUji) {
2      double[][] data = new double[dataUji.length][4];
3      for (int i = 0; i < dataUji.length; i++) {
4          data[i][0] = dataUji[i][0];
5          data[i][1] = dataUji[i][1];
6          data[i][2] = x[0] + (x[1] * dataUji[i][2]) + (x[2] *
7                  dataUji[i][3]) + (x[3] * dataUji[i][4]) +
8                  (x[4] * dataUji[i][5]) + (x[5] * dataUji[i][6]);
9          data[i][3] = Math.abs(data[i][1] - data[i][2]);
10     }
11     return data;
12 }
```

Kode Program 5.20 Hitung nilai prediksi pada data uji

```
1  public double hitungMAPE(double[] x, double[][] dataUji) {
2      double selisih = 0;
3      for (int i = 0; i < dataUji.length; i++) {
4          double nilaiAktual = dataUji[i][1];
5          double nilaiPrediksi = x[0] + (x[1]*dataUji[i][2]) + (x[2]*
6                  dataUji[i][3]) + (x[3] * dataUji[i][4]) +
7                  (x[4] * dataUji[i][5]) + (x[5] * dataUji[i][6]);
8          selisih += Math.abs(nilaiAktual- nilaiPrediksi)/nilaiAktual;
9      }
10     double mape = selisih / dataUji.length * 100;
11     return mape;
12 }
```

Kode Program 5.21 Hitung MAPE pada data uji

Pada Kode Program 5.20 parameter input yang diperlukan berupa nilai gbest pada iterasi terakhir serta data uji. Method memiliki nilai kembalian berupa array dua dimensi dengan tujuan agar mudah mengolahnya untuk ditampilkan dalam mode GUI. Baris ke-4 merupakan data tahun, baris ke-5 berisi data nilai aktual konsumsi energi primer pada data uji, baris ke-6 sampai merupakan proses pencarian nilai prediksi menggunakan pemodelan regresi linear, dan baris ke-8 berisi data selisih absolut antara nilai aktual dengan nilai hasil prediksi.

Implementasi hitung prediksi dan hitung MAPE dapat disatukan dalam satu method saja, akan tetapi timbul permasalahan proses perhitungan MAPE memerlukan proses lagi nantinya karena Java tidak bisa mengembalikan data yang berbeda (MAPE hanya berupa satu nilai sedangkan Kode Program 5.20 mengembalikan dalam bentuk array). Karena hal itu, maka diperlukan method tersendiri untuk menghitung MAPE sesuai Persamaan 2.22. Berikut implementasi hitung MAPE yang ditunjukkan pada Kode Program 5.21.

5.4 Implementasi Antarmuka Pengguna

Antarmuka pengguna terdiri dari antarmuka prediksi, ubah parameter, detail fitness, detail hasil, dan lihat data.

5.4.1 Implementasi Antarmuka Prediksi

Antarmuka prediksi konsumsi energi primer dengan $PSOACO_R$ merupakan antarmuka yang pertama kali ditampilkan perangkat lunak ke pengguna yang ditunjukkan seperti Gambar 5.2. Antarmuka tersebut menampilkan nilai parameter $PSOACO_R$ secara default yang dapat diubah melalui button Ubah Parameter. Jika pengguna menginginkan metode lain seperti PSO sendiri atau ACO_R sendiri dapat memilih melalui menubar Gambar 5.1.

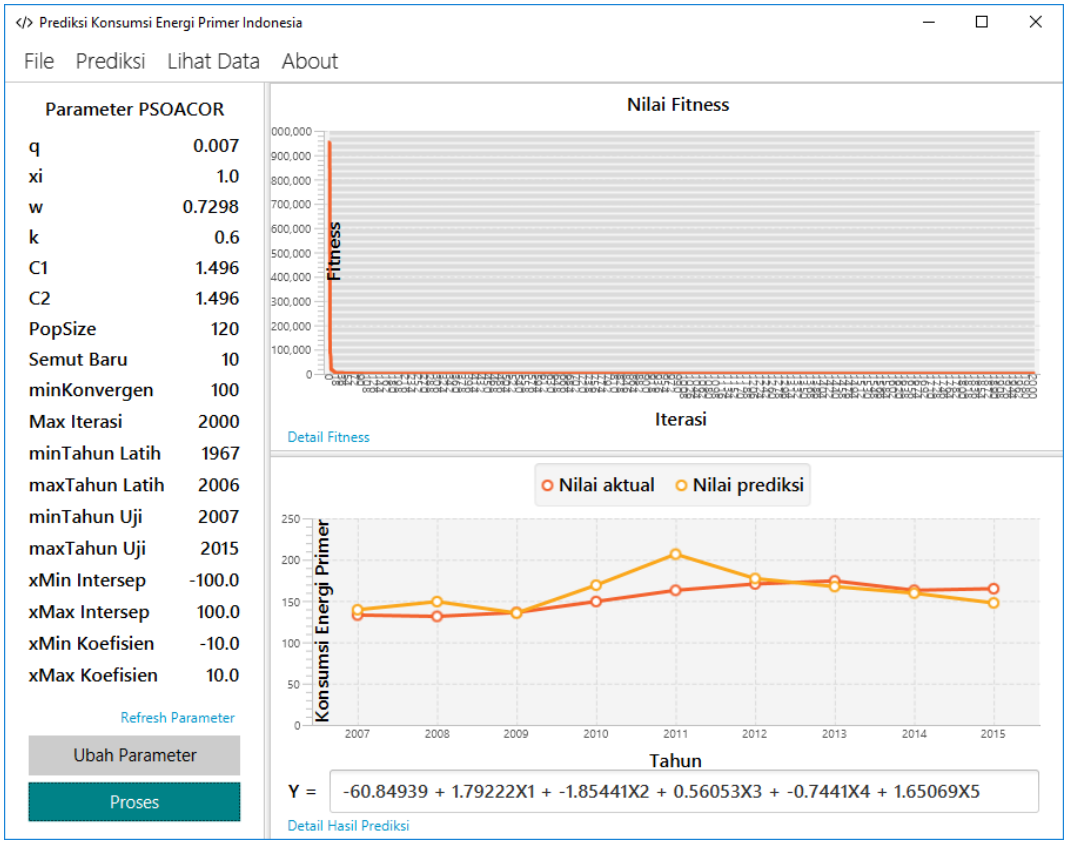
Gambar 5.1 Menubar memilih metode prediksi

Tampilan utama prediksi (Gambar 5.2) memiliki 3 kolom utama yakni kolom yang menampilkan parameter $PSOACO_R$, kolom menampilkan nilai *fitness* dalam grafik garis, dan kolom hasil prediksi yang dibandingkan dengan nilai aktual. Detail kolom utama tersebut yakni:

1. Parameter $PSOACO_R$ menampilkan parameter-parameter yang ada dalam metode $PSOACO_R$ sendiri. Dibawahnya terdapat dua button utama yakni Ubah Parameter dan Proses. Ubah Parameter berfungsi untuk mengubah parameter, setelah parameter diubah pengguna bisa me-*refresh* nilai

parameter melalui hyperlink Refresh Parameter. Tombol proses berguna untuk melakukan prediksi yang didalamnya berupa inisialisasi PSOACO$_R$ dan iterasinya. Setelah proses selesai yang durasinya bergantung pada parameter yang dimasukkan maka secara otomatis grafik garis muncul pada kolom nilai *fitness* dan hasil prediksi.

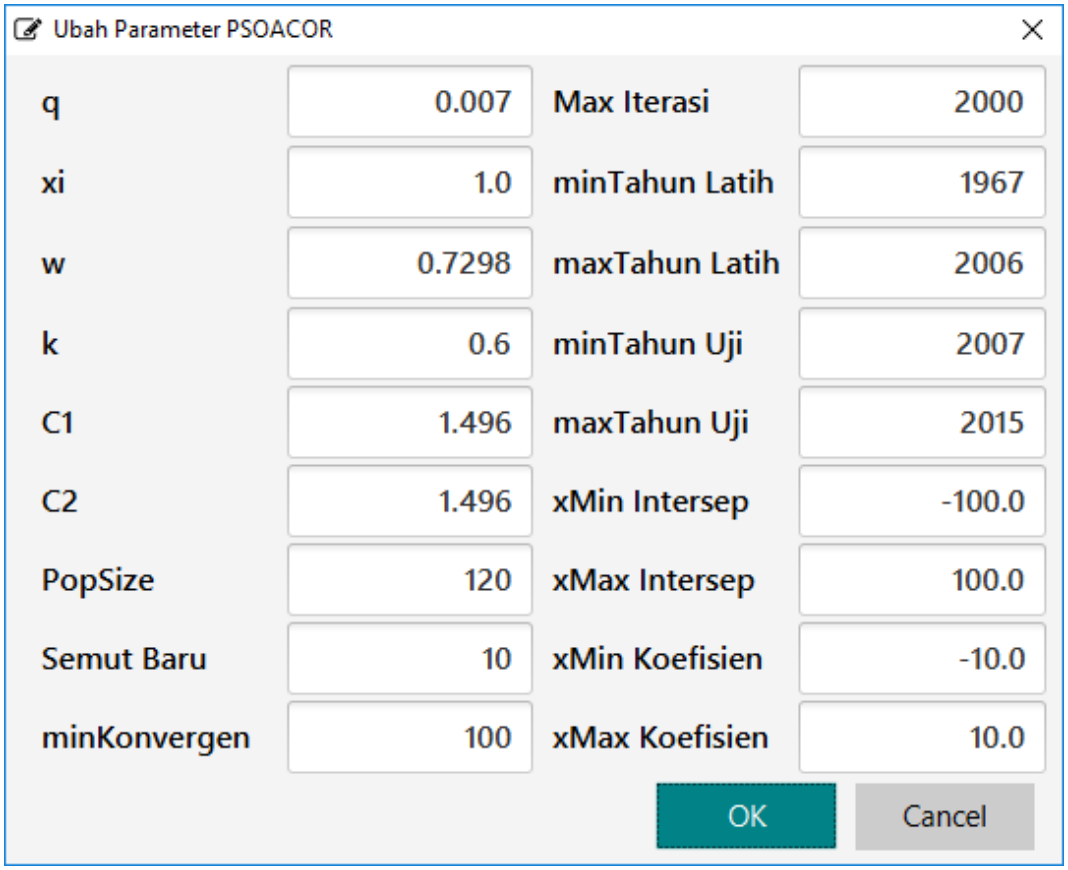
2. Kolom nilai *fitness* berisi grafik garis yang menampilkan nilai *fitness* setiap iterasinya. Jika ingin melihat angka persis nilai *fitness* bisa dilakukan dengan memilih hyperlink detail *fitness*.
3. Kolom hasil prediksi berisi grafik garis yang menampilkan nilai prediksi pertahunnya pada data uji. Garis warna orange menunjukkan nilai aktual konsumsi energi primer sedangkan warna kuning menunjukkan nilai prediksinya. Pada bagian terbawah terdapat nilai pemodelan regresi linear yang diperoleh dari nilai gbest iterasi terakhir.



Gambar 5.2 Antarmuka prediksi dengan PSOACO$_R$

5.4.2 Implementasi Antarmuka Ubah Parameter

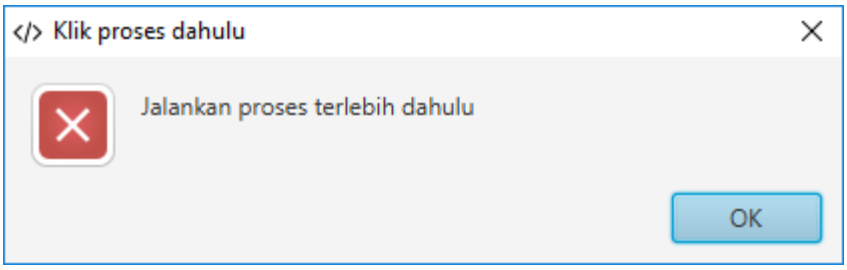
Untuk mengubah nilai parameter, pertama pilih button Ubah Parameter. Saat antarmuka Ubah Parameter muncul seperti pada Gambar 5.3, terdapat *textfield* untuk mengubah nilai masing-masing parameter. Jika nilai parameter yang diinginkan sudah diubah, pilih *hyperlink* refresh parameter untuk meng-*update* nilai pada antarmuka utama. Nilai parameter-parameter secara default di-set dengan angka optimal hasil pengujian pada bab 6



Gambar 5.3 Antarmuka ubah parameter

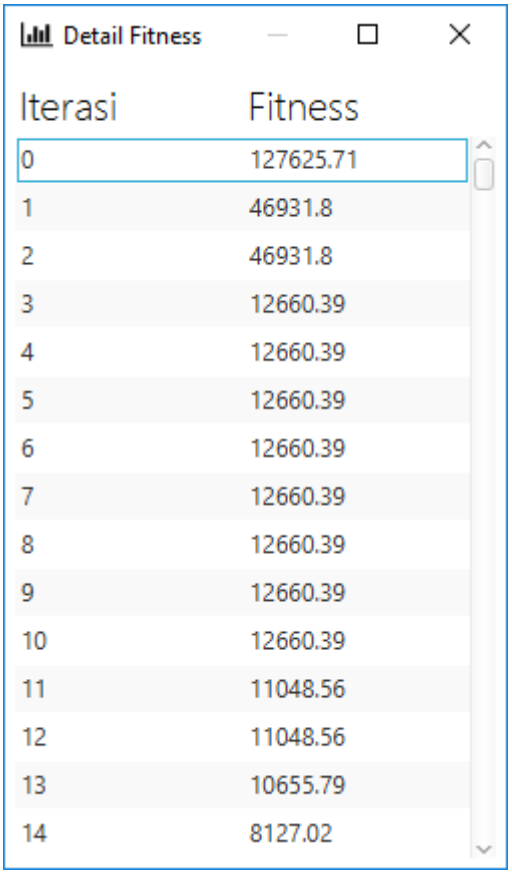
5.4.3 Antarmuka Detail Fitness

Untuk melihat lebih detail nilai *fitness* setiap iterasinya, klik *hyperlink* detail *fitness* setelah proses prediksi dijalankan. Jika proses prediksi belum dijalankan maka akan muncul antarmuka dialog yang menampilkan pesan kesalahan seperti pada Gambar 5.4.



Gambar 5.4 Pesan kesalahan detail fitness

Nilai *fitness* yang ditampilkan merupakan nilai selisih absolut antara nilai aktual dengan nilai prediksi. Iterasi ke-0 menunjukkan hasil tahap inisialisasi sedangkan iterasi ke-1 dan seterusnya merupakan hasil tahap iterasi. Nilai *fitness* ditampilkan dalam format dua angka dibelakang koma agar mudah dibaca. Antarmuka detail *fitness* ditunjukkan oleh Gambar 5.5.



Gambar 5.5 Antarmuka detail fitness

5.4.4 Implementasi Antarmuka Detail Prediksi

Antarmuka detail hasil berguna untuk melihat nilai aktual dan prediksi pertahunnya secara jelas. Data nilai prediksi dan nilai aktual ditampilkan dalam format dua angka dibelakang koma. Pada bagian bawah terdapat nilai MAPE yang dihasilkan prediksi pada data uji dalam satuan persen (Gambar 5.6).

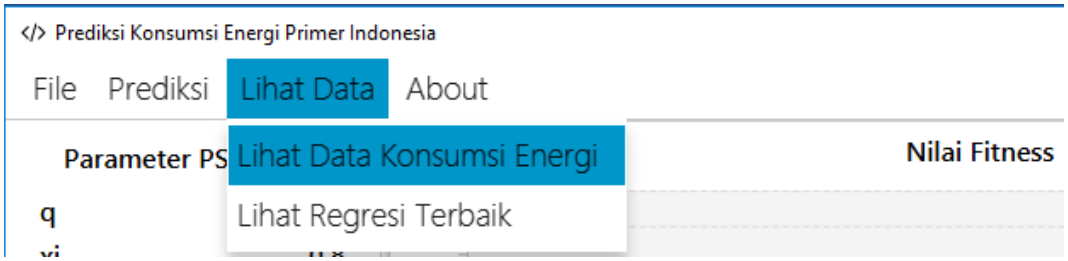
5.4.5 Implementasi Antarmuka Lihat Data

Dalam penelitian ini data disimpan dalam database local SQlite, untuk melihat semua data yang digunakan dalam penelitian serta hasil pemodelan regresi yang terbentuk. Untuk mengaksesnya pertama pilih menu bar lihat data (Gambar 5.7) kemudian pilih Lihat Data Konsumsi Energi. Data konsumsi energi akan ditampilkan pertahunnya berupa nilai konsumsi energi primer (nilai aktual), GNI, GDP, Populasi, Impor, dan Ekspor Indonesia seperti yang ditunjukkan pada Gambar 5.8.

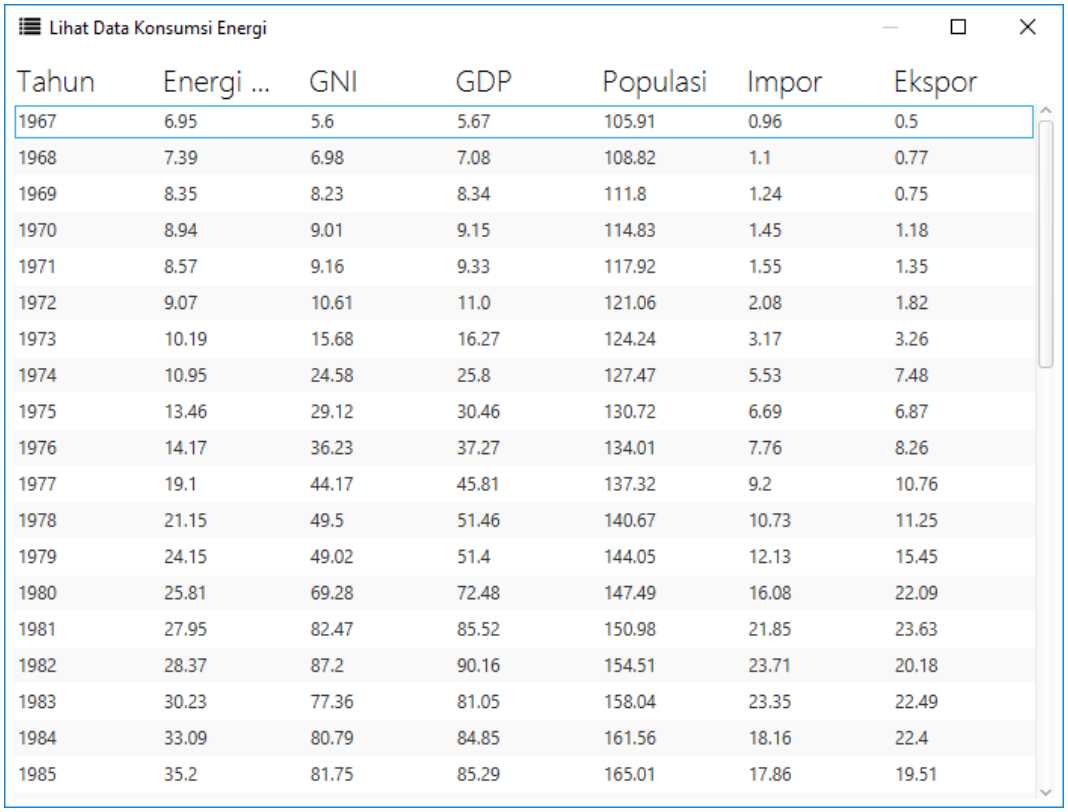
Agar pemodelan regresi yang telah terbentuk yang menghasilkan nilai MAPE terbaik tidak hilang jika proses pencarian dilakukan lagi, maka perlu dilakukannya penyimpanan nilai intersep beserta koefisien kemiringan regresi dalam sebuah tabel baru dalam database SQLite. Pada Gambar 5.9 menampilkan isi dari database pemodelan regresi yang terbentuk yang diurutkan mulai dari pemodelan dengan MAPE terbaik (terkecil). Untuk kolom metode nilai 1 memiliki

arti bahwa pemodelan regresi linear dibentuk menggunakan metode PSOACO$_R$, 2 untuk PSO, dan 3 untuk ACO$_R$.

Gambar 5.6 Antarmuka detail prediksi

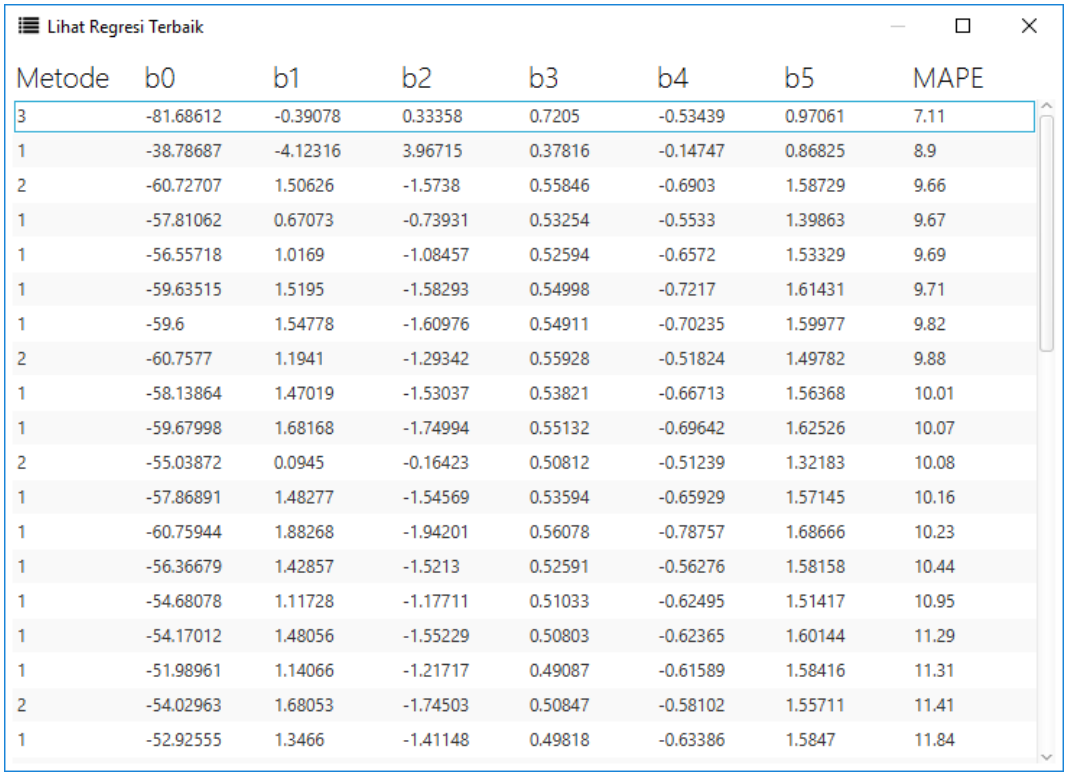


Gambar 5.7 Menu bar lihat data



Tahun	Energi ...	GNI	GDP	Populasi	Impor	Ekspor
1967	6.95	5.6	5.67	105.91	0.96	0.5
1968	7.39	6.98	7.08	108.82	1.1	0.77
1969	8.35	8.23	8.34	111.8	1.24	0.75
1970	8.94	9.01	9.15	114.83	1.45	1.18
1971	8.57	9.16	9.33	117.92	1.55	1.35
1972	9.07	10.61	11.0	121.06	2.08	1.82
1973	10.19	15.68	16.27	124.24	3.17	3.26
1974	10.95	24.58	25.8	127.47	5.53	7.48
1975	13.46	29.12	30.46	130.72	6.69	6.87
1976	14.17	36.23	37.27	134.01	7.76	8.26
1977	19.1	44.17	45.81	137.32	9.2	10.76
1978	21.15	49.5	51.46	140.67	10.73	11.25
1979	24.15	49.02	51.4	144.05	12.13	15.45
1980	25.81	69.28	72.48	147.49	16.08	22.09
1981	27.95	82.47	85.52	150.98	21.85	23.63
1982	28.37	87.2	90.16	154.51	23.71	20.18
1983	30.23	77.36	81.05	158.04	23.35	22.49
1984	33.09	80.79	84.85	161.56	18.16	22.4
1985	35.2	81.75	85.29	165.01	17.86	19.51

Gambar 5.8 Antarmuka lihat data konsumsi energi



Metode	b0	b1	b2	b3	b4	b5	MAPE
3	-81.68612	-0.39078	0.33358	0.7205	-0.53439	0.97061	7.11
1	-38.78687	-4.12316	3.96715	0.37816	-0.14747	0.86825	8.9
2	-60.72707	1.50626	-1.5738	0.55846	-0.6903	1.58729	9.66
1	-57.81062	0.67073	-0.73931	0.53254	-0.5533	1.39863	9.67
1	-56.55718	1.0169	-1.08457	0.52594	-0.6572	1.53329	9.69
1	-59.63515	1.5195	-1.58293	0.54998	-0.7217	1.61431	9.71
1	-59.6	1.54778	-1.60976	0.54911	-0.70235	1.59977	9.82
2	-60.7577	1.1941	-1.29342	0.55928	-0.51824	1.49782	9.88
1	-58.13864	1.47019	-1.53037	0.53821	-0.66713	1.56368	10.01
1	-59.67998	1.68168	-1.74994	0.55132	-0.69642	1.62526	10.07
2	-55.03872	0.0945	-0.16423	0.50812	-0.51239	1.32183	10.08
1	-57.86891	1.48277	-1.54569	0.53594	-0.65929	1.57145	10.16
1	-60.75944	1.88268	-1.94201	0.56078	-0.78757	1.68666	10.23
1	-56.36679	1.42857	-1.5213	0.52591	-0.56276	1.58158	10.44
1	-54.68078	1.11728	-1.17711	0.51033	-0.62495	1.51417	10.95
1	-54.17012	1.48056	-1.55229	0.50803	-0.62365	1.60144	11.29
1	-51.98961	1.14066	-1.21717	0.49087	-0.61589	1.58416	11.31
2	-54.02963	1.68053	-1.74503	0.50847	-0.58102	1.55711	11.41
1	-52.92555	1.3466	-1.41148	0.49818	-0.63386	1.5847	11.84

Gambar 5.9 Antarmuka lihat regresi terbaik