

CHAPTER V

IMPLEMENTATION AND TESTING

5.1 Implementation

This Section will explain how to use the program and how the program works. This will explain the code that used for solving the polynomial function.

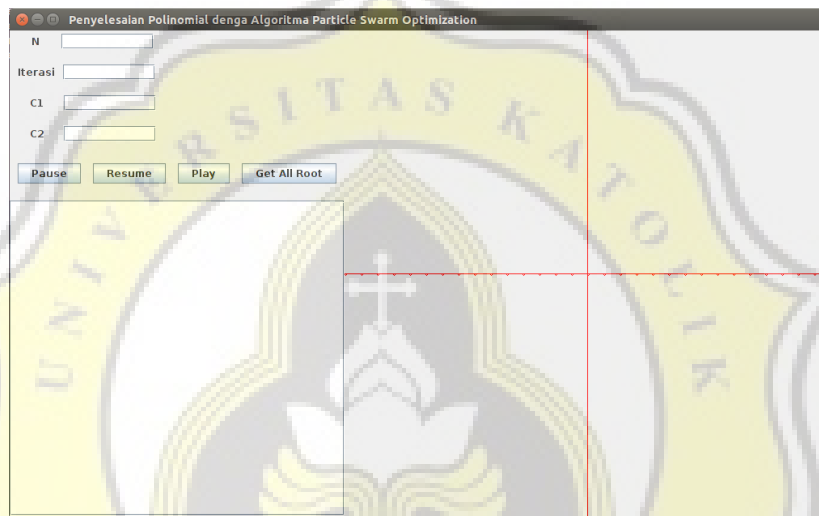


Figure 3. Graphical User Interface View

This is the Graphical User Interface view for the program. To use the program, first, input the polynomial function inside "myFile.txt".

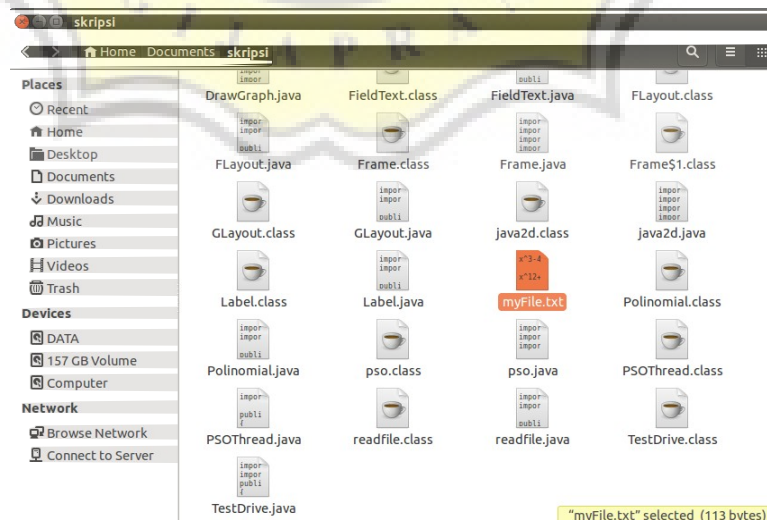
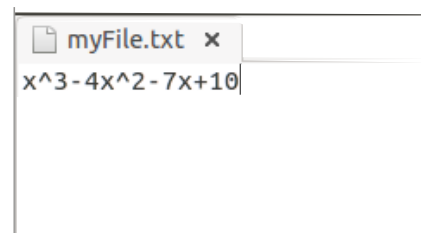


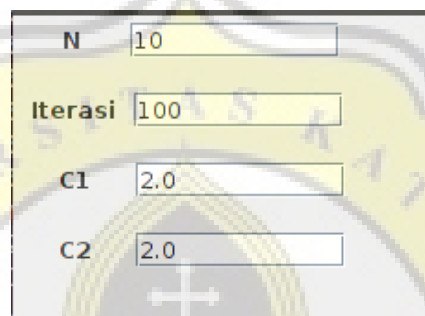
Figure 4. "myFile.txt" Location



```
myFile.txt x  
x^3-4x^2-7x+10
```

Figure 5. Inside "myFile.txt"

After that, input the parameters that used for Particle Swarm Optimization Algorithm.



N	<input type="text" value="10"/>
Iterasi	<input type="text" value="100"/>
C1	<input type="text" value="2.0"/>
C2	<input type="text" value="2.0"/>

Figure 6 Input the Parameters



Figure 7. The Button List

Then, there are 4 button below the parameter. Click the “Play” Button to play the algorithm for searching 1 root. This program will do the algorithm with a slower speed (0.5 second) and simultaneously draw the layout of each particle in the chart. It will show the calculation result in the text area too.

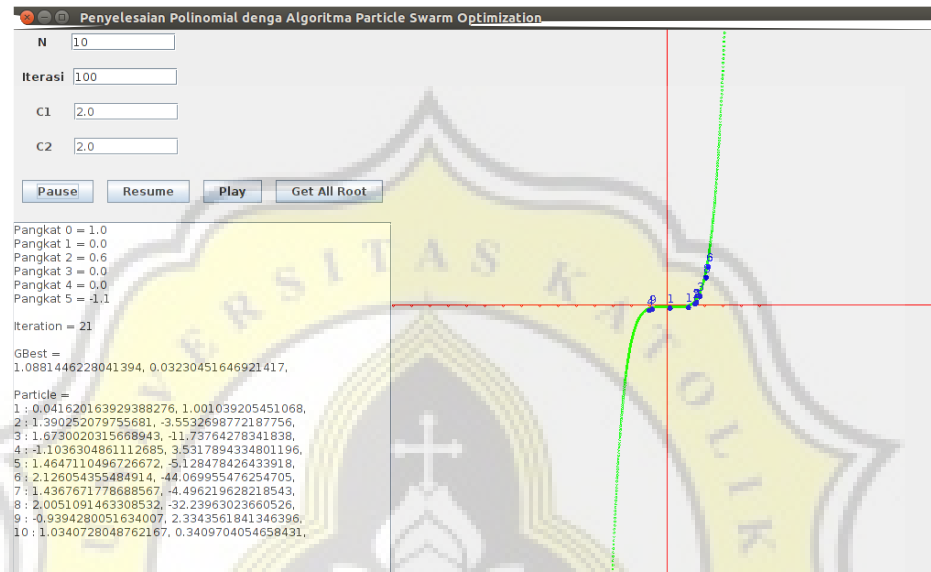


Figure 8. Search 1 Root

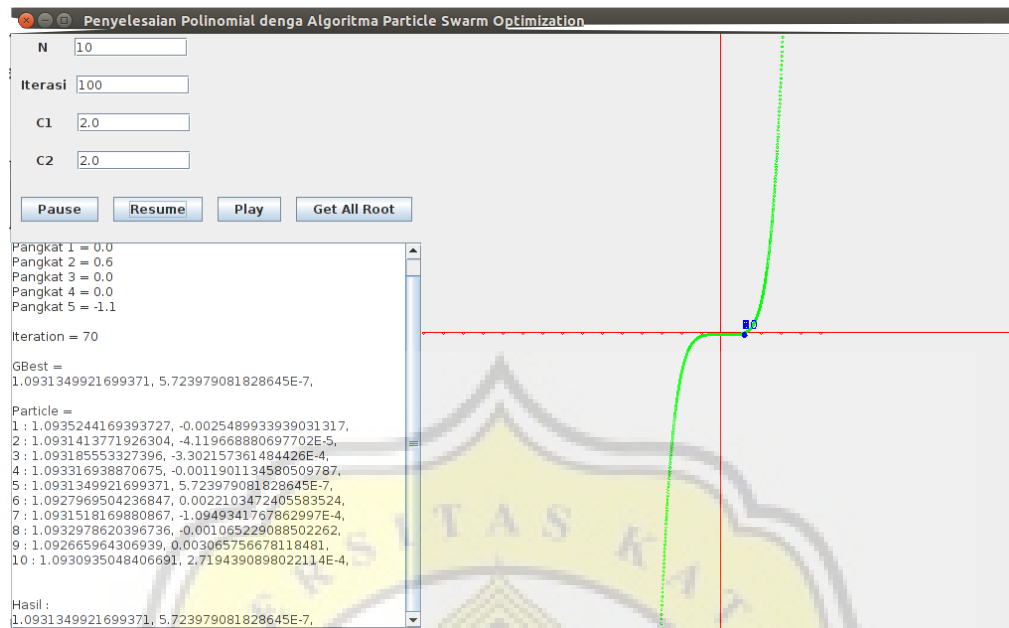


Figure 9. The Result when Searching 1 Root

“Pause” and “Resume” button used if you want to pause and resume the algorithm process so the user can look and inspect the particle. Click the “Get All Root” button to search all root from the polynomial function with a fast speed. It just show the result in the text area, but not draw the particle in the chart.

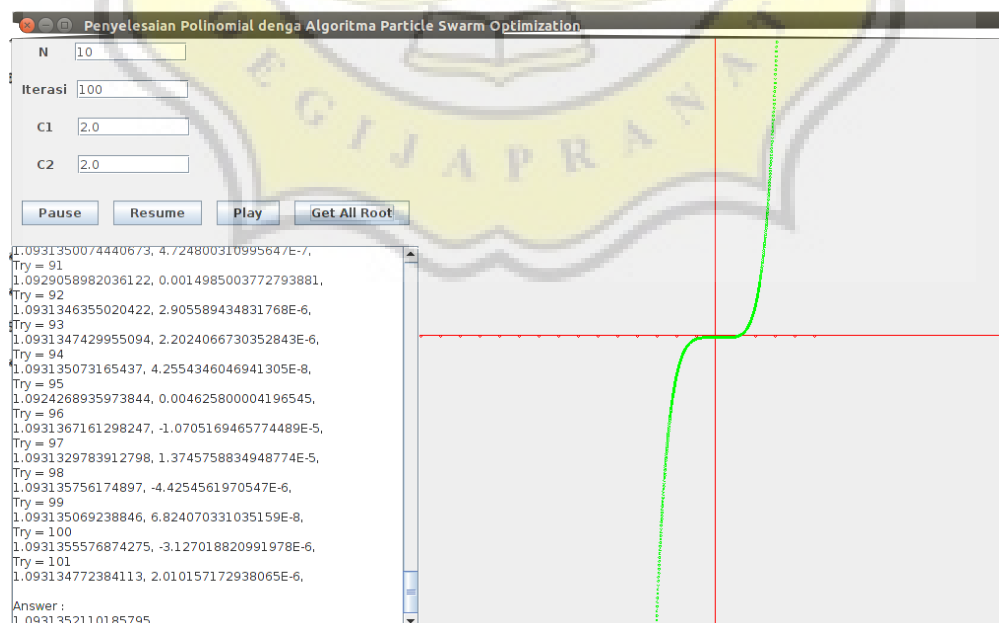


Figure 10. The Result when Searching All Root

This is the flow of the program. First, the program will read the text from file (myFile.txt).

```
public void read() throws Exception
{
    fr = new FileReader(this.namaFile);
    br = new BufferedReader(fr);
    this.text = br.readLine();
    fr.close();
}
```

Figure 11. Method for Read File

After that, the text will be processed to get the coefficients, exponents, and the highest exponent from the function. First, the program will get the term of the polynomial from the string of polynomial. There are many condition that need to solve some problem such as when user input the 0 and 1 value exponent, negative number, and 1 coefficient value.

```
private void PolinomialToSuku()
{
    String s = this.polinomial.replace("-", "+-");
    this.suku = s.split("\\+"); // mendapatkan nilai negatif
    pattern = Pattern.compile("[0-9]x");
    for( int i = 0; i < suku.length; i++)
    {
        // untuk konstanta
        if(!suku[i].contains("x"))
        {
            suku[i] = suku[i] + "x^0";
        }
        // untuk koefisien pangkat 1 (x)
        if(!suku[i].contains("^"))
        {
            suku[i] = suku[i] + "^1";
        }
        match = pattern.matcher(suku[i]);
        // untuk suku dengan koefisien 1 (1x^5 = x^5)
        if(!match.find())
        {
            if(suku[i].contains("-") && suku[i].contains("x"))
            {
                suku[i] = suku[i].replace("-", "");
                suku[i] = "-1" + suku[i] ;
            }
            else if(!suku[i].contains("-") && suku[i].contains("x"))
            {
                suku[i] = "1" + suku[i] ;
            }
        }
    }
}
```

Figure 12. Method to Get the Terms of Polynomial Function

After get the terms, the program will get the Coefficients and exponents from every term. The coefficients and exponents will be save into the 2 dimension array.

```
private void SukuToKoefisienNPangkat()
{
    this.koefisienNpangkat = new Double[this.suku.length][2];
    for( int i = 0; i<suku.length;i++)
    {
        String[] temp = suku[i].split("x\\^"); //split dengan "x^"
        koefisienNpangkat[i][0] = Double.parseDouble(temp[0]); // isi koefisien
        koefisienNpangkat[i][1] = Double.parseDouble(temp[1]); // isi pangkat
    }
}
```

Figure 13. Method to get the Coefficients and the Exponents

And then, the program will search the highest exponent. Highest exponent is used for completing the term that has 0 coefficient. So, the program will get all of the coefficient from all exponents, from 0 until the highest exponent.

```
private void calcPangkatTertinggi()
{
    // ambil pangkat tertinggi
    this.pangkattertinggi = 0;
    for(int a = 0; a < koefisienNpangkat.length; a++)
    {
        if(pangkattertinggi < koefisienNpangkat[a][1])
        {
            Double t = new Double(koefisienNpangkat[a][1]);
            pangkattertinggi = t.intValue();
        }
    }
}
```

Figure 14. Method to Get the Highest Exponent

```
private void FillZeroKoefisien()
{
    // untuk koefisien 0 (0x^4 = 0 = tidak ditulis)
    // temp punya index sampai pangkat tertinggi dan diisi nol semua
    Double[] temp = new Double[this.pangkattertinggi+1];
    for(int i=0; i<=pangkattertinggi; i++)
    {
        temp[i] = 0.00;
    }
    // index temp sebagai pangkat, isi temp sebagai koefisien
    // temp[a] = b -> koefisien pangkat a = b
    for(int a = 0; a <= pangkattertinggi; a++)
    {
        for(int b = 0; b < koefisienNpangkat.length; b++)
        {
            if(a == koefisienNpangkat[b][1])
            {
                temp[a] = koefisienNpangkat[b][0];
            }
        }
    }
    // masukkan kembali ke dalam array koefisienNpangkat
    this.koefisienNpangkat = new Double[this.pangkattertinggi+1][2];
    for(int a = 0; a <= pangkattertinggi; a++)
    {
        koefisienNpangkat[a][1] = new Double(a);
        koefisienNpangkat[a][0] = temp[a];
    }
}
```

Figure 15. Method to Fill the Terms

This coefficients and exponents will be used for the Particle Swarm Optimization algorithm to calculate the fitness every particle. This is the steps to run the algorithm.

```

public void playPSO(Polinomial a) throws Exception
{
    // step-step PSO
    setFirstParticle();
    calcFitnessParticle(a);
    evaluateFirstPBest();
    CalcGbest();
    // TimeUnit.SECONDS.sleep(1);
    if(gbest[jumlahx] != fsearch)
    {
        for(int i = 1; i<=imax;i++) //looping sampai imax
        {
            updateParticle(i);
            calcFitnessParticle(a);
            evaluatePBest();
            CalcGbest();
            Thread.sleep(500);
            System.out.println("i = " + i);
            DisplayGbest();
            // DisplayxParticle();
            // DisplayvParticle();
            //bila sudah dapat langsung selesai
            if(Math.abs(gbest[jumlahx] - fsearch) < 0.000001)
            {
                break;
            }
        }
        DisplayGbest();
    }
}

```

Figure 16. Method to Run the Algorithm

First, the program will prepare all of the parameter that used for PSO. There are amount of particles, maximum iteration, fitness value that need to be searched, the learning rates for particle and global, the dimension of particle sloution, the limit of particle location, and the 2 dimension array.

```

pso(int x, int nt, int i, Double f, Double c1, Double c2, Double xm) throws Exception
{
    ntotal = nt; //jumlah partikel
    imax = i; // iterasi maksimum
    fsearch = f; // nilai fitness yang akan dicari
    pc = c1; // particle confidence
    gc = c2; // global confidence
    jumlahx = x; // dimensi solusi
    xmax = Math.abs(xm); // batas solusi persebaran partikel
    n = new Double[ntotal][jumlahx+1];
    v = new Double[ntotal][jumlahx];
    pbest = new Double[ntotal][jumlahx+1];
    gbest = new Double[jumlahx+1];
    fi = new Double[jumlahx];
    rand = new Random();
}

```

Figure 17. Method to prepare All of the Parameters and the Array

Then, the program generate the first swarm with random solution and velocity.

```

public void setFirstParticle()
{
    for(int r = 0; r<ntotal; r++)
    {
        for(int c = 0; c < jumlahx ; c++)
        {
            // isi v dan x tiap particle secara random
            if(xmax.intValue() != 0)
            {
                n[r][c] = new Double(rand.nextInt(xmax.intValue()*2) - xmax.intValue()) + rand.nextDouble();
                v[r][c] = new Double(rand.nextInt(xmax.intValue()*2) - xmax.intValue()) + rand.nextDouble();
            }
            // n[r][c] = rand.nextDouble();
            // v[r][c] = rand.nextDouble();
            else
            {
                n[r][c] = new Double(rand.nextInt());
                v[r][c] = new Double(rand.nextInt());
            }
        }
    }
}

```

Figure 18. Method to Generate the First Swarm

After that, the program calculate the fitness of every particle with polynomial function that already be prepared before and save it to the last index of particle array.

```

private Double[] GetValueForCalcFitness(int i)
{
    //get particle solution untuk hitung Fitness
    for(int c = 0; c < jumlahx ; c++)
    {
        fi[c] = n[i][c];
    }
    return fi;
}

private void setFitnessParticle(int i, Double a)
{
    //isi nilai fitness particle
    n[i][jumlahx] = a;
}

public void calcFitnessParticle(Polinomial a) throws Exception
{
    // hitung nilai fitness particle
    Double hasil = 0.0;
    for(int i = 0; i< ntotal; i++)
    {
        for(int c = 0; c < jumlahx; c++)
        {
            Double x = GetValueForCalcFitness(i)[c];
            hasil = a.calculateF(x);
        }
        setFitnessParticle(i, hasil);
    }
}

```

Figure 19. Methods to Get the Fitness Value and Save It to Array

When first iteration, the Particle best of every particle is empty. So, the program just input the solution and the fitness every particle into Particle best without comparing.

```
public void evaluateFirstPBest()
{
    // PBest langsung diisi pertama kali sesuai dengan first particle karena PBest belum ada isinya.
    //setiap Particle belum dapat achievement
    for(int r = 0; r<ntotal; r++)
    {
        for(int c = 0; c < jumlahx+1; c++)
        {
            pbest[r][c] = n[r][c];
        }
    }
}
```

Figure 20. Method to Save Particle Best for the First Time

After that, the program will get the Global best from the Particle best every particle. The program will compare it one by one.

```
public void CalcGbest()
{
    //isi gbest awal dengan salah 1 pbest particle
    for(int c = 0; c < jumlahx+1; c++)
    {
        gbest[c] = pbest[0][c];
    }
    //mengambil gbest dari semua pbest tiap particle dengan membandingkan 1-1
    for(int r = 0; r<ntotal; r++)
    {
        if(Math.abs(gbest[jumlahx]) > Math.abs(pbest[r][jumlahx]))
        {
            for(int c = 0; c < jumlahx+1; c++)
            {
                gbest[c] = pbest[r][c];
            }
        }
    }
}
```

Figure 21. Method to Get the Global Best

The program check the termination criteria. If iteration already into maximum iteration or if the Global best fitness close enough with the fitness that want to be searched, the program will stop and display the result. If not, the process will continue to next step, updating the velocity and solution every particle.

```

public void updateParticle(int i)
{
    //update velocity dan location (kecepatan dan lokasi (v dan x))
    w = 0.9 - (((0.9-0.4)/imax)*i);
    // w = 0.1;
    r1 = rand.nextDouble();
    r2 = rand.nextDouble();
    // r1 = 0.1;
    // r2 = 0.1;
    for(int r = 0; r<ntotal; r++)
    {
        for(int c = 0; c<jumlahx; c++)
        {
            //update v dan x tiap particle
            v[r][c] = w*v[r][c] + pc*r1*(pbest[r][c] - n[r][c]) + gc*r2*(gbest[c] - n[r][c]);
            // membatasi agar kecepatan tidak melebihi batas kecepatan
            // digunakan bila tidak menggunakan W (weight inertia)
            // if(v[r][c] > vmax)
            // {
            //     v[r][c] = vmax;
            // }
            // else if(v[r][c] < (-1)*vmax)
            // {
            //     v[r][c] = (-1)*vmax;
            // }
            n[r][c] = n[r][c] + v[r][c];
        }
    }
}

```

Figure 22. Method to Update Velocity and Location Every Particle

After that, the program calculate the fitness again and evaluate the Particle best every particle. But now it will comparing between the new fitness of particle and the fitness of Particle best every particle. If the particle more better than Particle best, that the solution of particle will be saved into the Particle best.

```

public void evaluatePBest()
{
    //membandingkan f particle baru dengan f pbest particle
    for(int r = 0; r<ntotal; r++)
    {
        if(Math.abs(pbest[r][jumlahx]) > Math.abs(n[r][jumlahx]))
        {
            for(int c = 0; c < jumlahx+1; c++)
            {
                pbest[r][c] = n[r][c];
            }
        }
    }
}

```

Figure 23. Method to Get Particle Best Every Particle

And then, get the Global best, check the termination criteria, and so on until maximum iteration or it get the answer.

This is the code for draw the line graph.

```
public void drawGarisGrafik(Graphics2D g2d)
{
    g2d.setColor(Color.red);
    g2d.drawLine(0, 300, 600, 300);
    g2d.drawLine(300, 0, 300, 600);
    for(int a = 0; a<= 30; a++) // Draw Titik x dan y
    {
        g2d.draw(new Ellipse2D.Double(a*20, 300, 2, 2)); //draw x dot
        // g2d.draw(new Ellipse2D.Double(300, a*20, 2, 2));
    }
}
```

Figure 24. Method to Draw the Line Graph

And this is the code to draw the Polynomial graph. It will draw a point for every 0.01 x value. The y of the point obtained from the function value of the x.

```
public void drawGrafikPolinomial(Graphics2D g2d)
{
    g2d.setColor(Color.green);
    if(pol != null)
    {
        for(int i = 0; i < pol.searchKoeffisien(0)*2*100; i++) //draw polinomial
        {
            Double b = new Double(i*0.01 - pol.searchKoeffisien(0)); //draw dot every 0.01
            Double c = pol.calculateF(b);
            b = (b*20)+300;
            c = c+300;
            g2d.draw(new Ellipse2D.Double(b, c, 2, 2));
        }
    }
}
```

Figure 25. Method to Draw the Polynomial Graph

This is the code to draw the particle from Particle Swarm Optimization algorithm. The solution and the fitness value of the algorithm are the x and y in the graph.

```
public void drawParticlePSO(Graphics2D g2d)
{
    g2d.setColor(Color.blue);
    if(pol != null)
    {
        for(int i = 0; i < ntot; i++) // Draw PSO Particle
        {
            double x = p[i][0];
            double y = p[i][1];
            Double z = new Double(pol.searchKoeffisien(0));
            x = (x*20)+300;
            y = y+300;
            Ellipse2D.Double circle = new Ellipse2D.Double(x, y, 5, 5);
            g2d.fill(circle);
            g2d.draw(circle);
            float a = (float)x;
            y = y-3.0;
            float b = (float)y;
            g2d.drawString(Integer.toString(i+1),a,b); //Give Number
        }
    }
}
```

Figure 26. Method to Draw the Particle in Graph

5.2 Testing

In this section, it will show the result of testing with a different value of parameters. The parameters are amount of particle, maximum iteration, weigh inertia, random number, and learning rates for particle and global. The testing will use 2 polynomial function to show the difference with a different problem. It will test the weigh inertia value with the formula too,

which is $w_{max} - \left(\frac{w_{max} - w_{min}}{i_{max}} \right) i$. w_{max} is 0,9 and w_{min} is 0,4 .

Every value of parameter will be tested 10 times. There are 2 result that can be obtained from this testing, the iterations and the percentage. The iterations shows the average iterations that needed for find the root. The percentage shows how often the success for finding the result. So, if the the iterations smaller, it mean that the result better. Except, if the iterations are 0, it mean that all of the 10 times testing does not get the result. If the percentage smaller, it mean that the testing sometimes do not get the result. So, the more percentage is the better. 100 percentage mean that the algorithm can get the result 10 times from 10 times testing with the parameter.

The first polynomial function is $x^3 - 4x^2 - 7x + 10$. First, it will be tested with 1000 amount of particles, 1000 maximum iterations, r1 and r2 using random number, weigh inertia with the formula, and the learning rates 2.0 for all of them (c1 and c2).

And this is the testing result when changing the n value (particle amount). As show below, the algorithm search and get the result faster when using more amount of particle because the test get the result with smaller iterations. All of the percentage are 100. So from 10 times testing, all of the value can get the result 10 times too.

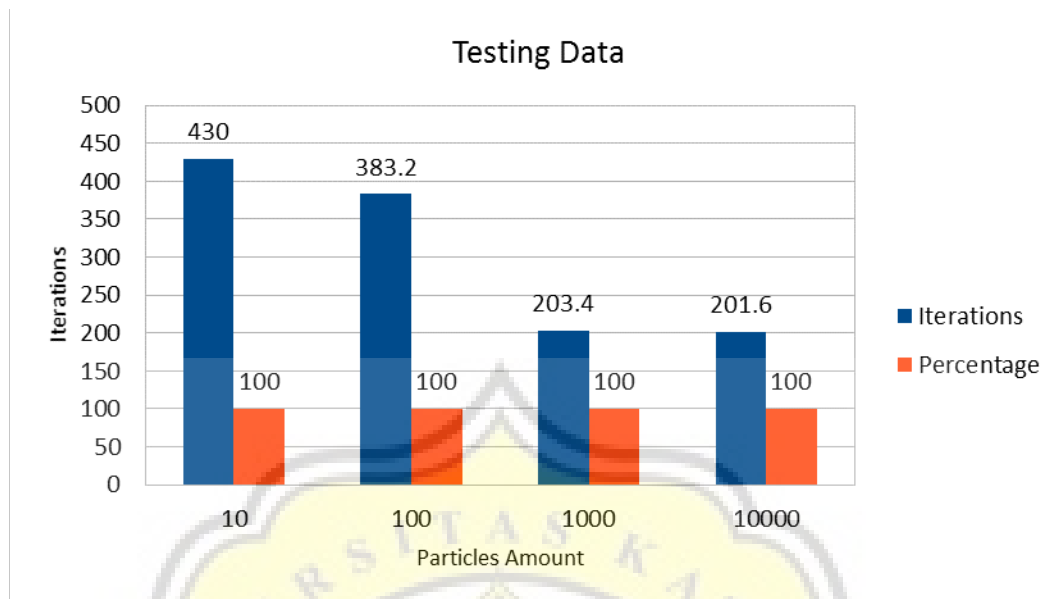


Figure 27. Testing Data when Change the Particle Amount

When changing the maximum iteration, it get the result more often with higher amount of maximum iteration because the percentage bigger. But it will be more slower too because the iterations bigger too. It get the result more faster with smaller iteration, but sometime it fail to get the result. The algorithm cannot get the result with 10 maximum iteration.

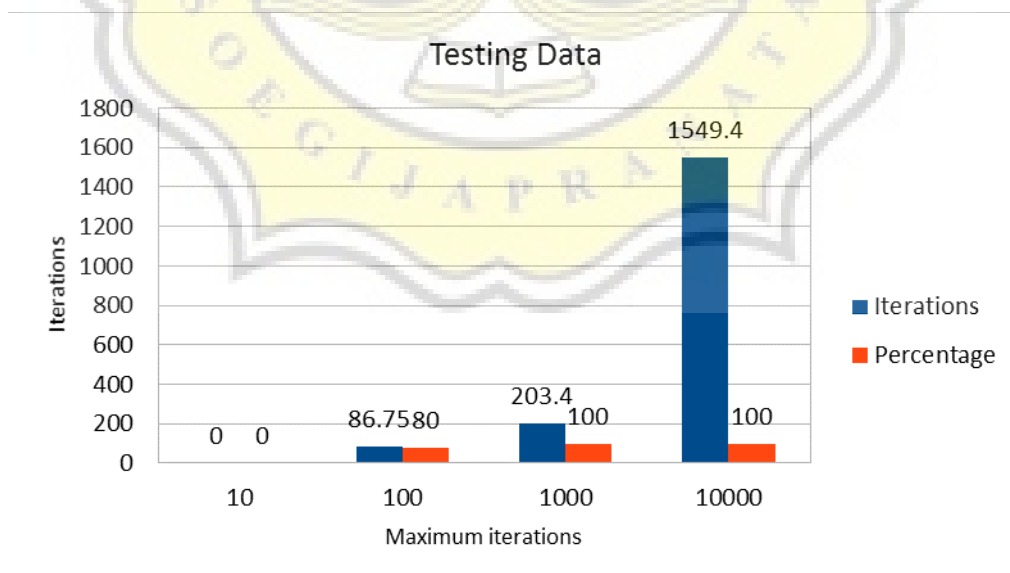


Figure 28. Testing Data when Change the Maximum Iterations

If changing the weigh inertia value, smaller value is better than bigger value because the iterations are smaller. It can get the result faster. But It slower when using the formula. Up to 0,9 value, it cannot get the result.

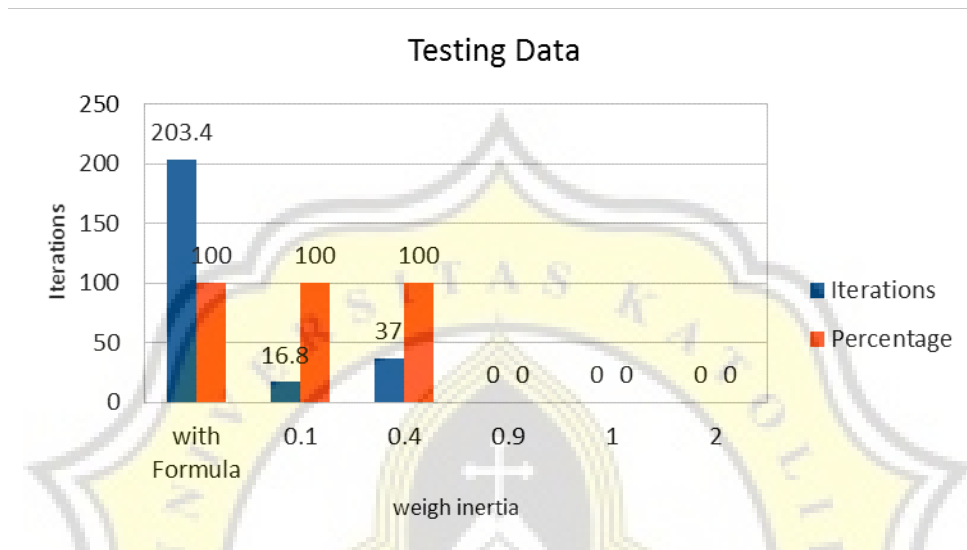


Figure 29. Testing Data when Change the Weigh Inertia

If changing the random value (r1 and r2), it will get the result when use the value below 1. The result get better with smaller value because the iterations are smaller, but it not change too much. It become slower when use the random value.

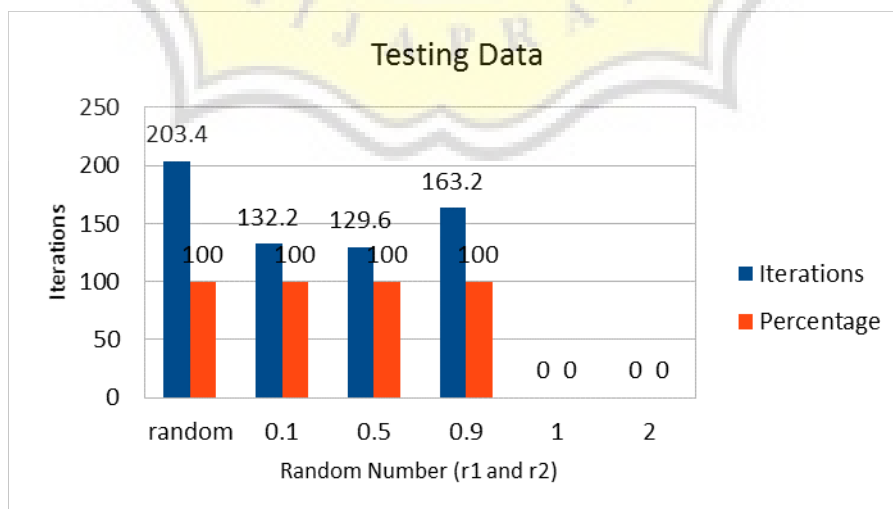


Figure 30. Testing Data when Change the Random Number

When change the learning rates, it get better result when use the smaller value because the iterations are smaller. When use a big value like 4, it will not get the result. If make the different value between c1 and c2, it will be better if give weigh more to the particle learning rates (c1) because the iterations are fewer when the c1 have more value than c2.

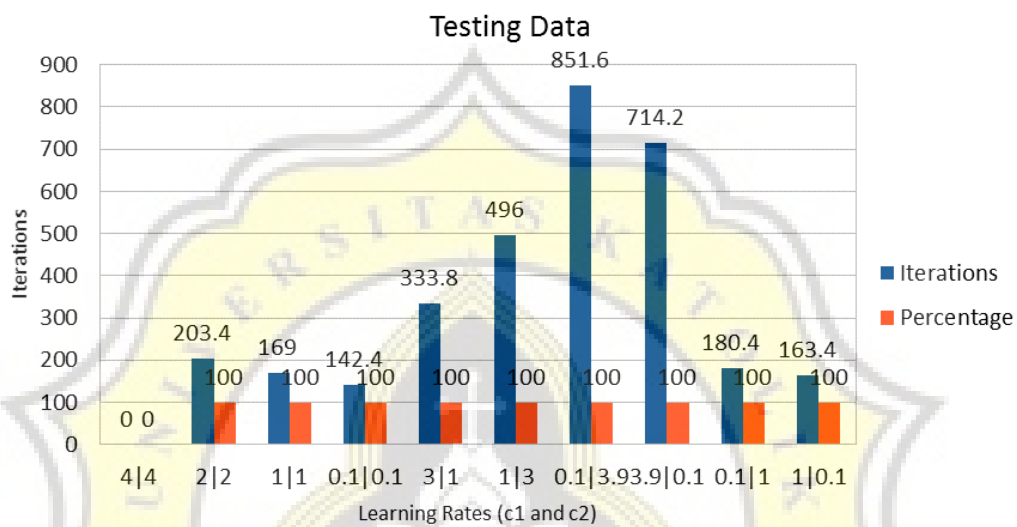
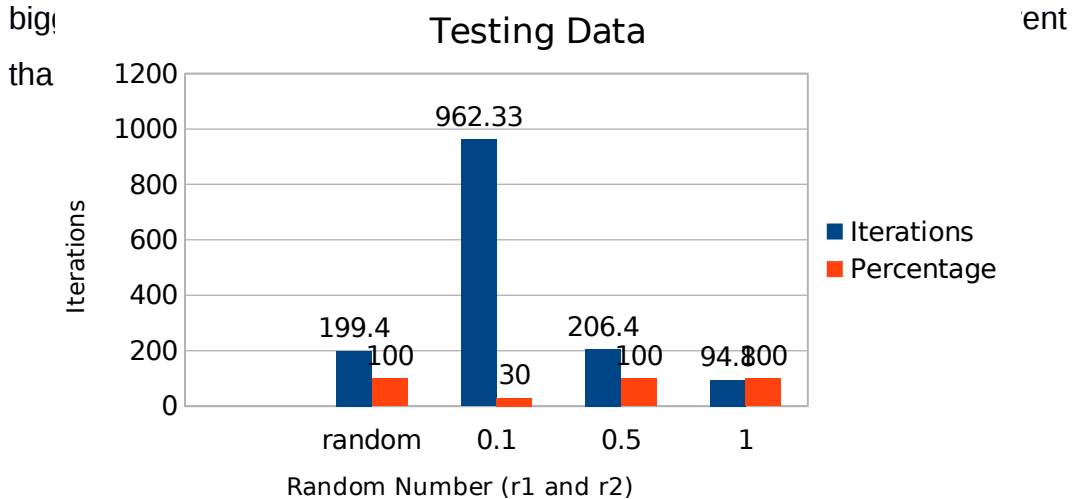


Figure 31. Testing Data when Change the Learning Rates

From the test result above, it can be assumed that the smaller value is better than the bigger value. But when all of the parameters are small(0.1), except the particle amount and the maximum iteration, it become more slower to get the result and the percentage to get the result become worse.

And this is the testing result if all of the parameters are small value (0.1). If change the weigh inertia value, the bigger value more better than smaller value because the iterations are smaller. It can search faster with big



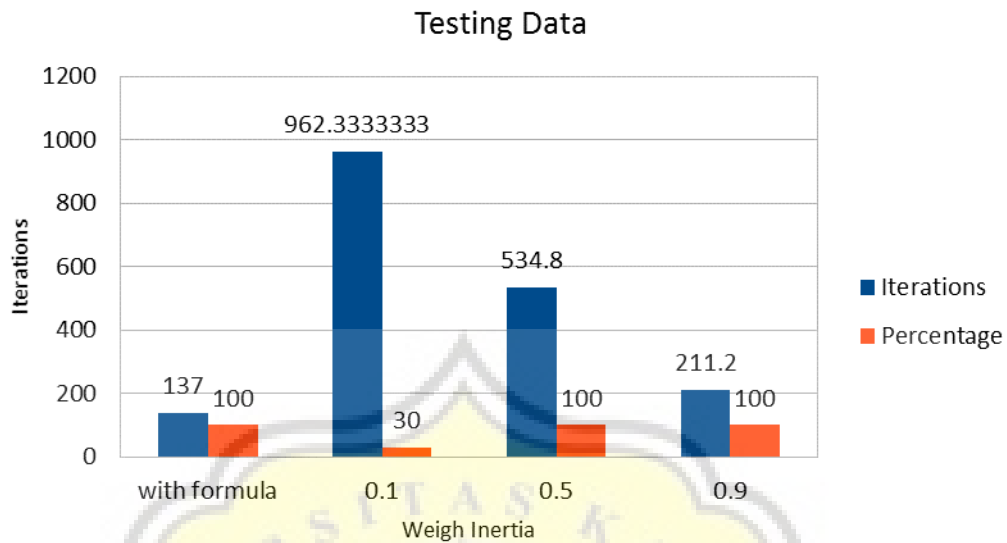


Figure 32. Data Testing when Change the Weigh Inertia. Another Parameters are 0.1.

When change random value, it get better value with bigger value too. The iterations become smaller and the percentage become bigger if using bigger value. It can get the result faster with random value. But it still better without random value because fixed number still can be better.

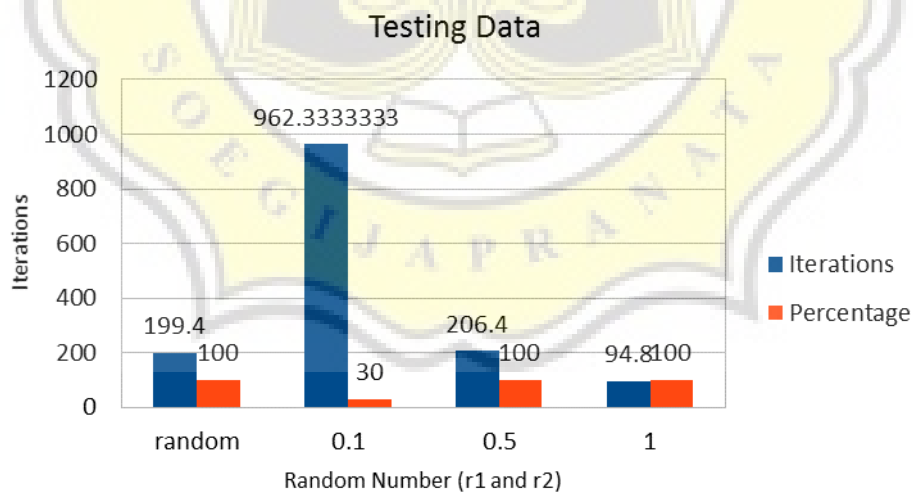


Figure 33. Testing Data when Change the Random Number. Another Parameters are 0.1.

And this is when change the learning rates value. Same as another testing, it better with a bigger value. The iterations are smaller. But it become more slower and worse if give weigh too small to the global learning rates (c2). It become better if the global learning rates value have a bigger value.

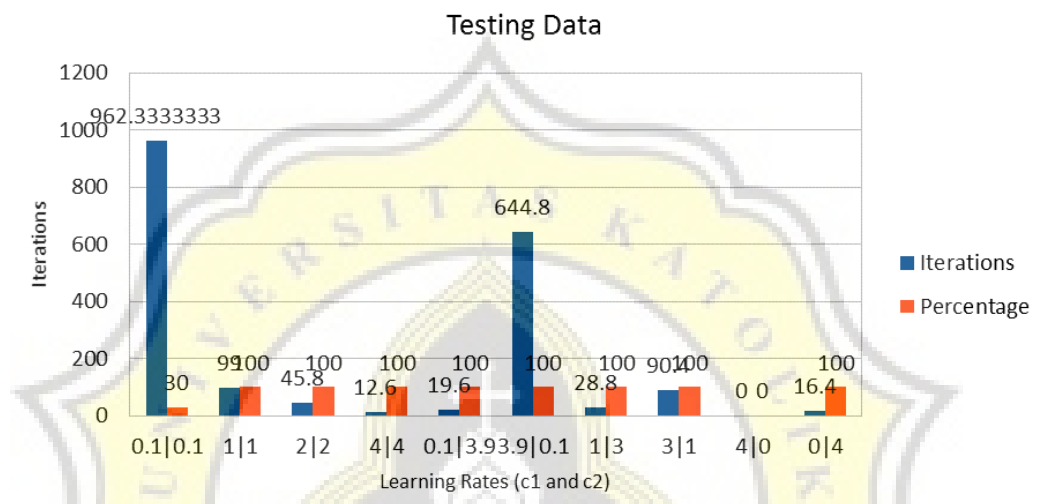


Figure 34. Testing Data when Change the Learning Rates. Another Parameters are 0.1.

As the result above, it not always become better if all of the parameter bigger or smaller. It need some adjustment to set the parameter. If the one is small, the other need to be bigger. If the one is bigger, the other need to be smaller.

The second polynomial function is :
 $5x^5 + x^6 - x^4 + 3x^7 - x^3 - 4x^2 - 7x + 1000$. First, it will be tested with 1000 amount of particles, 1000 maximum iterations, r1 and r2 using random number, weigh inertia with the formula, and the learning rates 2.0 for all of them (c1 and c2).

This is the testing result when changing the particle amount value. The result not changing too much. The iterations and the percentage not change too much although the amount value already changed.

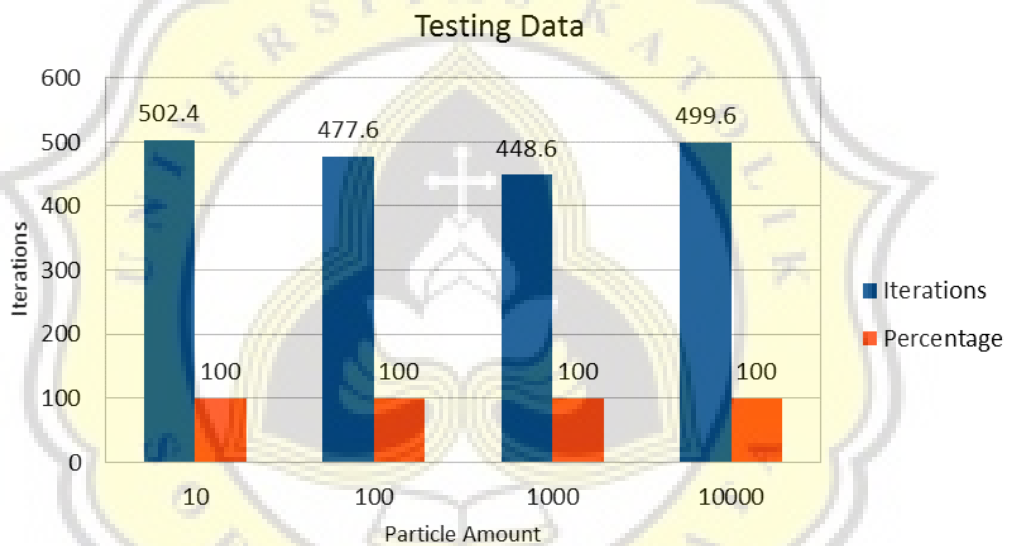


Figure 35. Testing Data when Change the Particle Amount with Different Problem

When changing the maximum iteration value, it does not get the value with smaller value. It can get the result with 1000 or more value, but it is slower because the iterations are bigger.

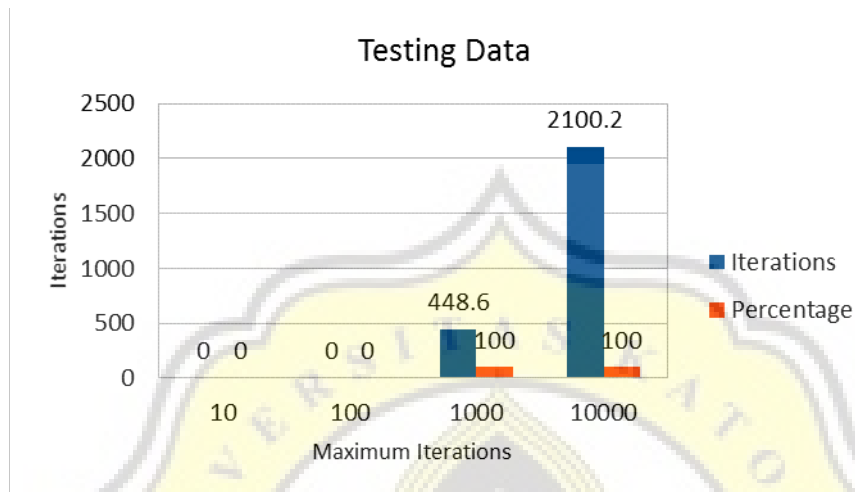


Figure 36. Testing Data when Change the Maximum Iterations with Different Problem

When changing the weigh inertia value, it get better result and faster when use the smaller value, just like another polynomial function. Up to 0,9 value, it cannot get the result. And It become slower if using the formula because the iterations are bigger.

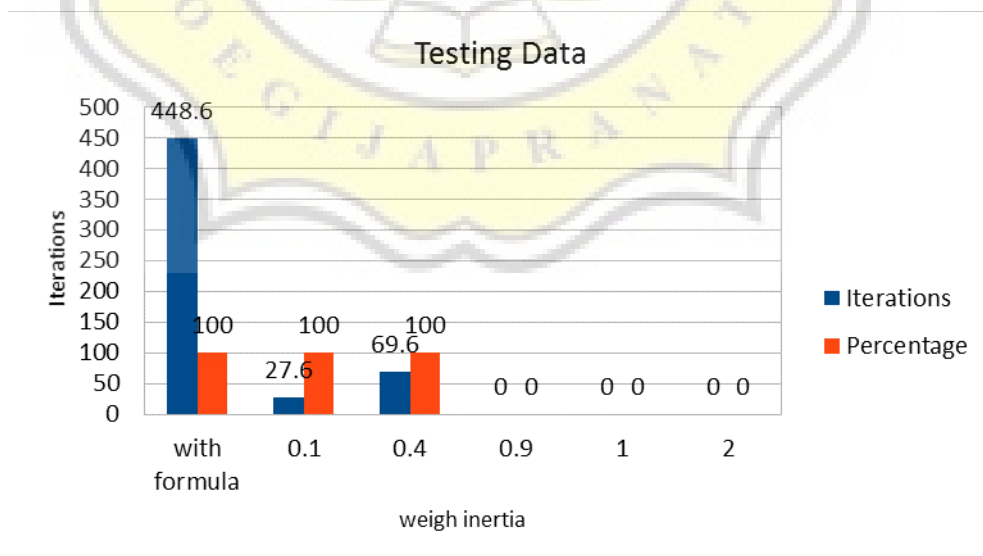


Figure 37. Testing Data when Change the Weigh Inertia with Different Problem

If change the random value, it got better result with bigger value, but it not change too much with smaller value because the iterations almost as same as another. It does not get the result when use up to 0.9 value.

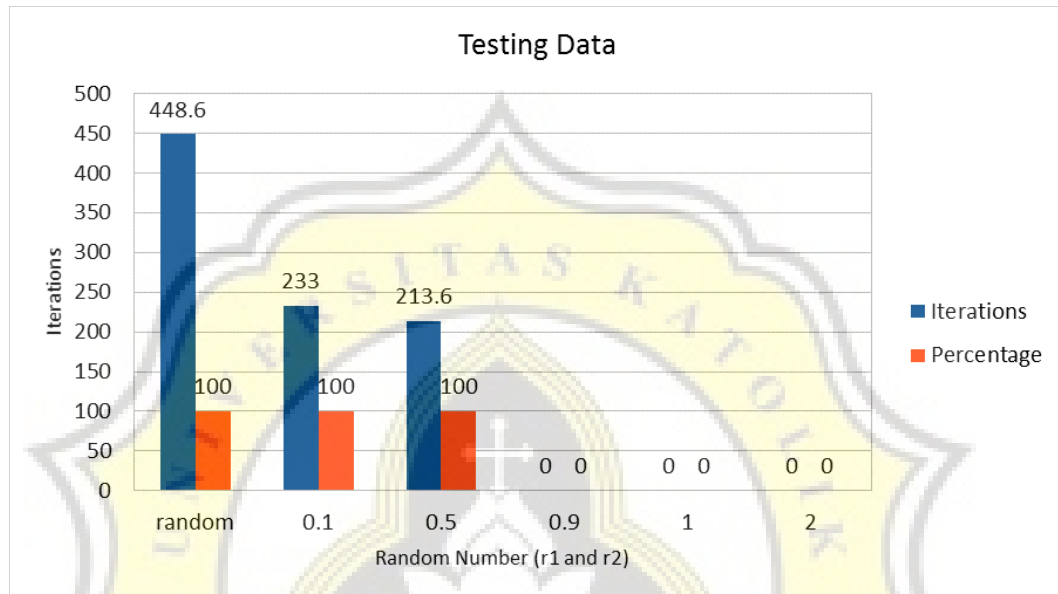


Figure 38. Testing Data when Change the Random Value with Different Problem

If change the learning rates value, it can search more faster when use smaller value because the iterations are smaller. When use bigger value, the change to get the result become smaller. If make the different value between c_1 and c_2 , it will be better if give weigh more to the particle learning rates (c_1) because the iterations are fewer when the c_1 have more value than c_2 .

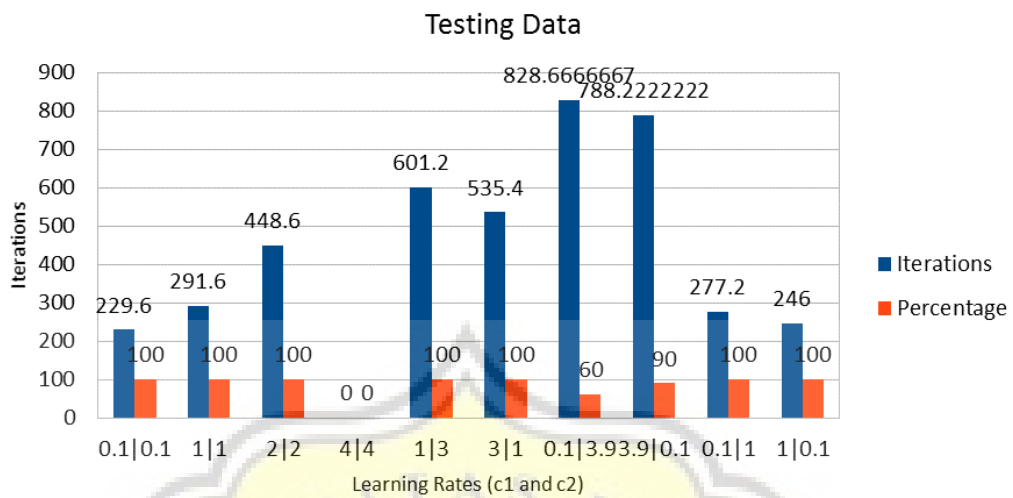


Figure 39. Testing Data when Change the Learning Rates with Different Problem

From the result above, it shows that the different problem of polynomial function not affecting too much with the parameters. But it become more slower if solving the more complicated problem than the easier problem.