一、 程式設計

RBFN 參照作業 2 的方法設計

Step1. 根據粒子群大小，隨機產生出 n 組(p+2)*J+1 維向量(θ,w_1 , w_2 ,.., w_J , m_{11}, m_{12},.., m_{1p}, m_{21}, m_{22},.., m_{2p},.., m_{J1}, m_{J2},..., m_{Jp}, σ_1, σ_2, ..., σ_J)，每組向量皆視為一個粒子的位置，粒子的速度初始為零

Step2. 將粒子位置帶入 RBFN 求出每一粒子的評估值(fitness)

Step3. 每一粒子與其自身最佳評估值比較，若新的評估值較粒子的最佳評估值佳，則以新的位置及評估值取代粒子的最佳解位置及評估值

Step4. 每一粒子與群體最佳評估值比較，若粒子的最佳評估值較群體的最佳評估值佳，則以粒子的最佳解位置及評估值取代群體最佳解位置及評估值

Step5. 以下列式子更新每一粒子的速度及位置

$$\begin{cases} \underline{v}_i(t) = \underline{v}_i(t-1) + \varphi_1(\underline{p}_i(t) - \underline{x}_i(t-1)) + \varphi_2(\underline{p}_g(t) - \underline{x}_i(t-1)) \\ \underline{x}_i(t) = \underline{x}_i(t-1) + \underline{v}_i(t) \end{cases}$$

$V_i(t)$: 時間 t 時，第 i 個粒子的速度

$X_i(t)$: 時間 t 時，第 i 個粒子的位置

$\varphi 1, \varphi 2$: 對自身跟群體的學習比例

$P_i(t)$: 時間 t 時，第 i 個粒子的最佳解位置

$P_g(t)$: 時間 t 時，群體的最佳解位置

Step6. 重複 Step2~Step5，直到移動次數符合要求

二、 主要程式碼

Step1.

```
for (int i = 0; i < groupNum; i++) {
        double[] velocities = new double[Dimension];
        Particle p = new Particle(locations[i], velocities);
        particles.add(p);
}

for (int m = 0; m < itrNum; m++) {
```

Step2 and Step3.

```
for (Particle p : particles) {
        double fitness = rbfn.computeEofN(p.getLocation());
        double BestFitness = p.getBestFitness();
        if (fitness < BestFitness) {
                p.setBestFitness(fitness);
                p.setPBest(p.getLocation());
        }
}
```

Step4.

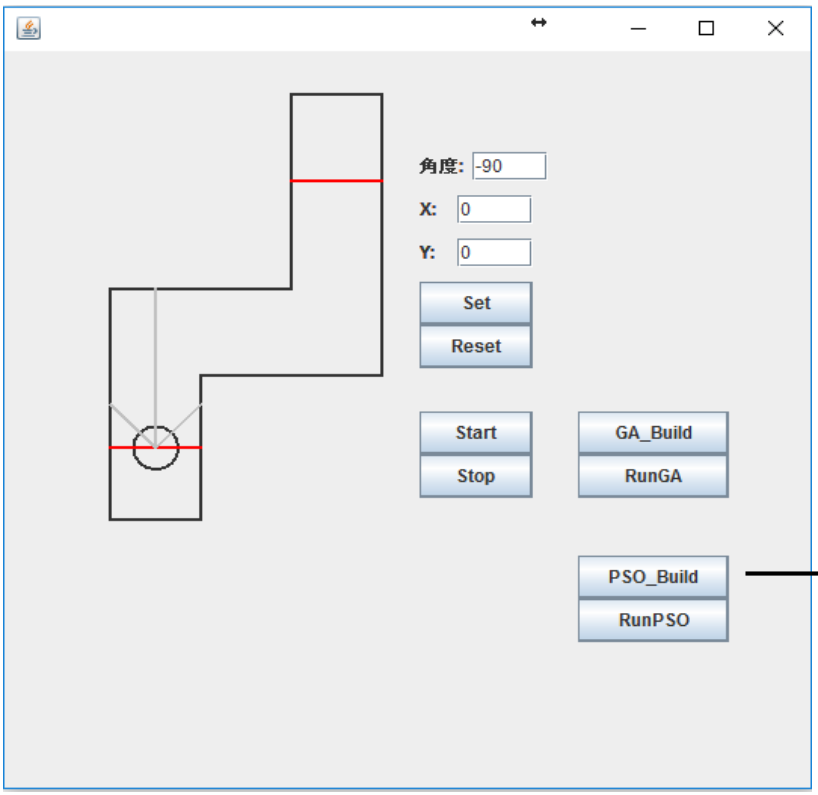
```
for (Particle p : particles) {
        double pFitness = p.getBestFitness();
        if (pFitness < gFitness) {
                gFitness = pFitness;
                for (int i = 0; i < Dimension; i++) {
                        gBest[i] = p.getPBest()[i];

                }
        }
}
```

Step5.

```
        double weight = 0.1;
        for (Particle p : particles) {
                double[] velocity = p.getVelocity();
                double[] location = p.getLocation();
                double[] pBest = p.getPBest();
                double phi_1 = Math.random();
                double phi_2 = Math.random();
                for (int i = 0; i < Dimension; i++) {
                        velocity[i] = velocity[i] + weight * phi_1 * (pBest[i] - location[i])
                                        + weight * phi_2 * (gBest[i] - location[i]);
                        location[i] = location[i] + velocity[i];
                }
                p.setVelocity(velocity);
                p.setLocation(location);
        }
}
```

三、 執行畫面

主畫面



PSO_Build

備註：

1. Error(n)小於誤差值時才會停止重新隨機產生出新的族群
2. 演化完畢後的結果有機率會無法走到終點
3. 壓縮檔中附有 Demo 影片