# 메타물질 안테나

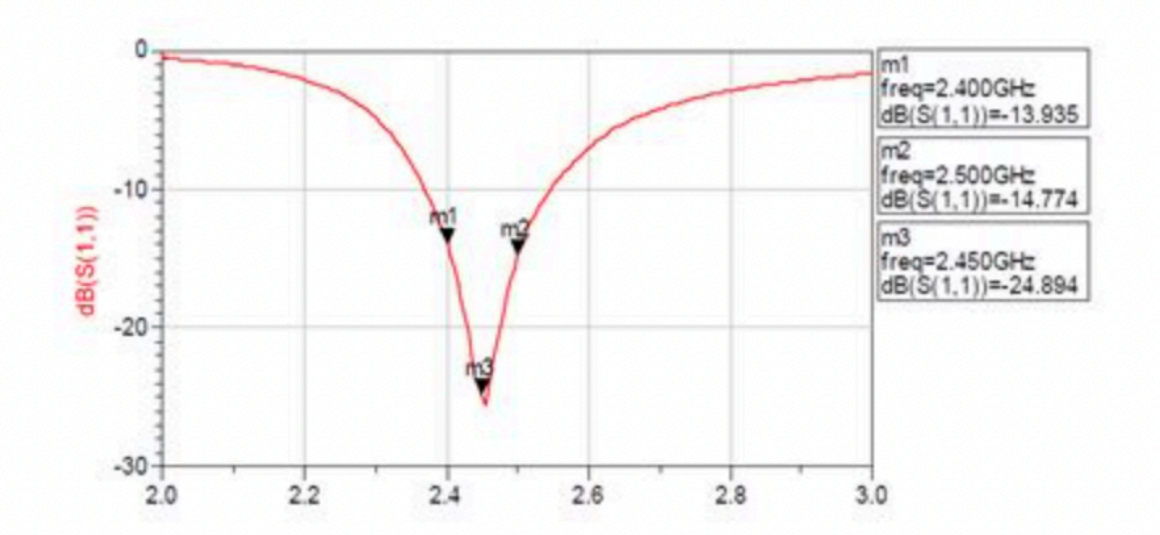
메타물질 안테나의 전자기 시뮬레이션 결과 분석

정지혁 / 202102934

컴퓨터 알고리즘

2023년 6월 6일 ~ 2023년 6월 22일

# 목차




## Metamarterial이란 무엇인가.

우리가 흔히 사용하는 Microstrip antenna는 크기, 무게, 성능, 설치 용이성 및 공기역학적 profile 이 제약이 있는 고성능 항공기, 우주선, 위성 및 미사일 응용 분야에 적용되는 low-profile 안테나이다. 민간 응용 프로그램에는 이동 통신 및 무선 연결이 이에 포함된다.

'Metamaterial'이란 빛이나 전자기파의 파장보다 훨씬 작은 크기의 금속이나 유전물질을 메타 원자의 주기적인 배열로 설계해 만든 것으로 가시광선이나 마이크로파를 굴절시켜서 물체를 보이지 않게 하는 성질을 갖는 물질이다. 메타물질이 적용될 수 있는 분야는 매우 다양하며 항공우주산업, 센서 감지와 사회기반시설의 모니터링, 스마트 태양에너지 관리, 군중통제, 레이돔, 전쟁시의 고주파 통신, 고 이득 안테나의 렌즈, 초음파센서의 개선, 지진피해 방지 건물 등의 매우 다양한 분야에 적용된다.

Metamarterial은 독특한 전기적 특성을 가진 단위 셀의 특정 배치이다. 이러한 물질은 자연에 존재하지 않는 특성을 나타내는데, 메타물질의 성향, 전기적 허용율 및 자기 투자율은 음수이기 때문이다. 이는 전자기 복사가 예상과 다르게 동작하도록 하는 음의 굴절을 생성한다.

메타물질로 만든 안테나는 안테나의 복사력을 증대시키며, 특히 음의 투자율을 가지는 메타물질로 작동 주파수를 조절하거나 미세한 크기를 가지는 안테나를 만들 수 있다.

이번 글 에서는 이러한 장점을 가지는 메타물질 안테나의 전자기 시뮬레이션 결과에서 안테나의 전압 정재파 비율에 따른 반사 손실을 분석해보겠다.

## 안테나의 전압 정재파 비율과 반사 손실

반사 손실과 VSWR 모두 안테나의 매칭 상태를 측정하는 데 사용되는 파라미터이다. **반사 손실**과 **VSWR** 사이에는 수치적인 관계가 있다.

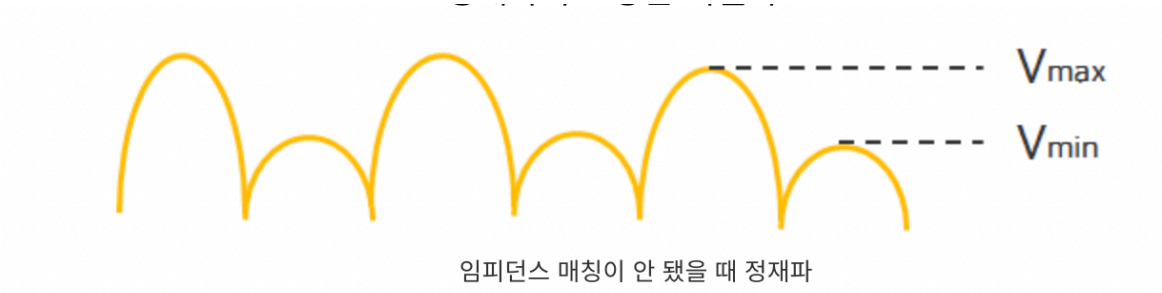
**반사 손실**(Return Loss)은 Reflective Loss라고도 알려져 있으며, 반사된 신호의 정도를 나타내는 파라미터이다. 입사 전력의 어느 정도가 신호원으로 다시 반사되는 지를 나타내는 수치이다.
반사 손실이 클수록 신호의 전송 효율은 높다는 것을 의미한다 (전송 장치로의 전력과 신호 강도의 손실이 줄어든다.)

반사 손실 = 입사 전력 / 반사 전력, dB단위

(그림1)

**VSWR(**Voltage Standing Wave Ratio)은 전압 정재파비로, 전송 선로를 따라 형성된 정재파의 최대 전압 대 최소 전압의 비율이다.

(그림2)




임피던스 매칭이 안 됐을 때 정재파

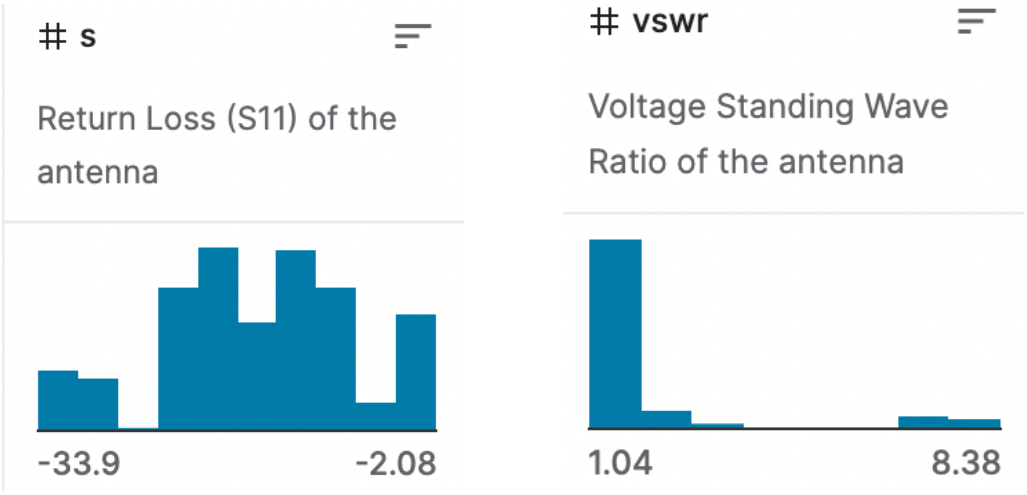
즉, 여기서 VSWR은 최대 진폭과 최소 진폭의 전압의 비를 나타내는 말이다. ( VSWR = Vmax/Vmin )
반사 손실과 VSWR 사이에는 상관관계가 있는데, 이는 아래와 같다.

반사 손실 = -20log ((VSWR-1) / (VSWR + 1))

VSWR이 작을수록 안테나의 매칭 성능이 좋아서 안테나에서의 출력이 더 높을 수 있다는 것을 나타낸다. 최소 VSWR은 1.0로, 이는 안테나로 부터 반사되는 전력이 없는 이상적인 값을 의미한다.

DataSet 소개

FotonTech의 소프트웨어 엔지니어 Renan Machado(플로리아노폴리스, 산타카타리나 주, 브라질)가 kaggle에 업로드(4Y ago)한 Metamaterial Antennas데이터 셋을 가지고 분석을 하였다.
(Open datasbase License를 따른다.)
column으로 시뮬레이션 결과로 나온 Metamarterial에 따른 SRR배열에 관한 정보와 Antenna gain, VSWR, bandwidth, S11(반사 손실), 안테나에 의해 방사되는 전력, 안테나에 허용되는 전력 등이 있다.




데이터 셋 불러오기(GoCSV.java)

```
package LinearRegression;

import java.io.*;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;

public class GoCSV {

   private String filePath;
   private BufferedReader bufferedReader;
   private List<String[]> readCSV;

   private int index;

   //This constructor is for read CSV File
   public GoCSV(String filePath) throws IOException {
      this.filePath = filePath;
```

```
        bufferedReader = new BufferedReader(new InputStreamReader(new
FileInputStream(this.filePath), "UTF-8"));
        readCSV = new ArrayList<>();

        makeList(bufferedReader);
        this.index = 1;     //1열(domain)빼고  반환
    }

    public void makeList(BufferedReader bufferedReader) throws IOException {
        String line = null;
        while((line = bufferedReader.readLine())!=null) {
            String[] lineContents = line.split(",(?=([^\"]*\"[^\"]*\")*[^\"]*$)",-1);
            String[] arr = {lineContents[8], lineContents[10]};  //사용할 컬럼(8번째, 10번째)만 배열에 담음
            //readCSV.add(lineContents);                          //csv파일 전체 반환
            readCSV.add(arr);
        }
    }
        //한 행을 읽음
        public String[] nextRead(){
        if(readCSV.size() == index){
            return null;
        }
        return readCSV.get(index++);
    }
}


// 호출 MetamarterialAntenna .java
```

---

```
StringTokenizer st;
GoCSV goCSV = new GoCSV("/Users/jihyeok/정지혁  프로젝트/study/Algorithm/src/
LinearRegression/antenna.csv"); //파일 저장한 절대 경로
String[] line=null;
vswr = new ArrayList<>();
s11 = new ArrayList<>();
while((line = goCSV.nextRead())!=null){
    vswr.add(Double.parseDouble(line[0]));
    s11.add(Double.parseDouble(line[1]));
    /*for(String a : line){
        System.out.print(a+" ");
    }
    System.out.println();
    */
}
```

# Linear Model

입력 특성에 대한 선형함수를 만들어 예측을 수행하는 모델이다. 다양한 선형모델이 존재하며 분류와 회귀에 모두 사용 가능하다.이 때 우리가 찾아낼 수 있는 가장 직관적이고 간단한 모델인 선(line)을 이용해서 데이터와 비교해 해당 데이터를 가장 잘 설명할 수있는 선을 찾는 분석이 바로 선형회귀분석(Linear Regression)이다.

Y = ax+b와 같은 회귀 직선에서 기울기(a)와 절편(b)에 따라서 선의 모양이 정해진다.

$$y = w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_px_p + b$$

실제로 분석하는 데이터의 특성의 개수에 따라 위와 같은 식으로 표현되며,

P : 특성의 개수

w: 가중치, 계수(학습이 되면서 예측이 잘되는 방향으로 변함)

b: 절편

과 같다.

## 두 변수간의 관계를 가장 잘 설명하는 매개변수 a와 b를 추정

회귀 직선 y=ax+b에서 회귀 계수(a)와 절편(b)를 어떻게 구할 수 있을까?

$$\beta_1 = \frac{\sum(x-\bar{x})(y-\bar{y})}{\sum(x-\bar{x})^2}$$

회귀 계수는 다음과 같은 식으로 구해 진다. 순서쌍 (x,y)가 있을 때, x 값을 x평균으로 뺀것과 y 값을 y평균으로 뺀 값을 순차적으로 모두 더한값에서 x값에서 x의 평균을 뺀 값의 제곱을 모두 더한 값을 나눠준다. 절편은 회귀 계수를 구한 후에, 주어진 식에 y의 평균, x의 평균을 대입하여 구한다.

절편과 회귀 계수 구하기 MetamarterialAntenna.java

```
static void calcRegLine(){
   double mx=0,my=0;
   int p = vswr.size();
   for(int i=0 ;i<p ; i++){
      mx+= vswr.get(i);
      my+=s11.get(i);
   }
   mx = mx/p;
   my = my/p;

   double x=0;
   double y=0;

   for(int i=0 ; i<p ; i++){
        x += Math.pow((vswr.get(i) - mx) , 2);   // ((x - x의  평균)의  제곱)의  합
        y += (vswr.get(i) - mx) * (s11.get(i) - my);
   }
   a = x / y;
   b = my - a*mx;

        //소수점  2자리에서 반올림
        a = Math.round(a*10)/10.0;
   b = Math.round(b*10)/10.0;
}
```

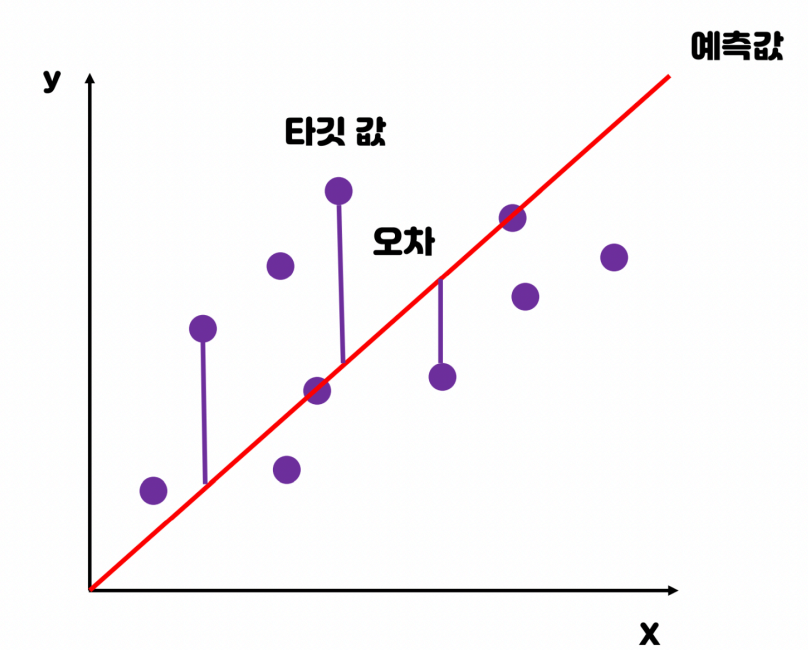
호출 MetamarterialAntenna.java

```
calcRegLine();
System.out.println("Regression Line : y = "+a+"x + (“+b+")");
```

```
Regression Line : y = 21.1x + (-60.3)
```

# 손실함수

평균 제곱 오차(MSE)는 회귀에서 자주 사용되는 손실 함수이다. MSE 즉, 평균 제곱 오차에 하여 알아보자, 앞서 진행한 과정처럼 회귀 직선을 그어 데이터를 예측하면 모든 값을 정확하게 예측할 수 없다. 예측 값들이 직선 위에 일치하는게 아니고 그 주위에 분포하는데, 실제 데이터와 예측값들의 차이를 손실이라고 한다.




예측값과 실제값의 차이를 줄이기 위하여 오차를 구한다. 무수히 많은 오차값을 구하는 과정에서, 양의 방향으로 발생한 손실과 음의 방향으로 발생한 손실이 서로 상쇄되지 않도록 오차를 제곱하여 더한다. 이러한 방식으로 구하는 것을 MSE(평균 제곱 오차)라고 한다.

$$cost = \frac{1}{m}\sum_{i=1}^{m}(H(x_i) - y_i)^2$$

$$H(x) = Wx + b$$

H(x) : 예측값

y : 실제값

w:가중치

cost : 오차

MSE구하기 MetamarterialAntenna.java

---

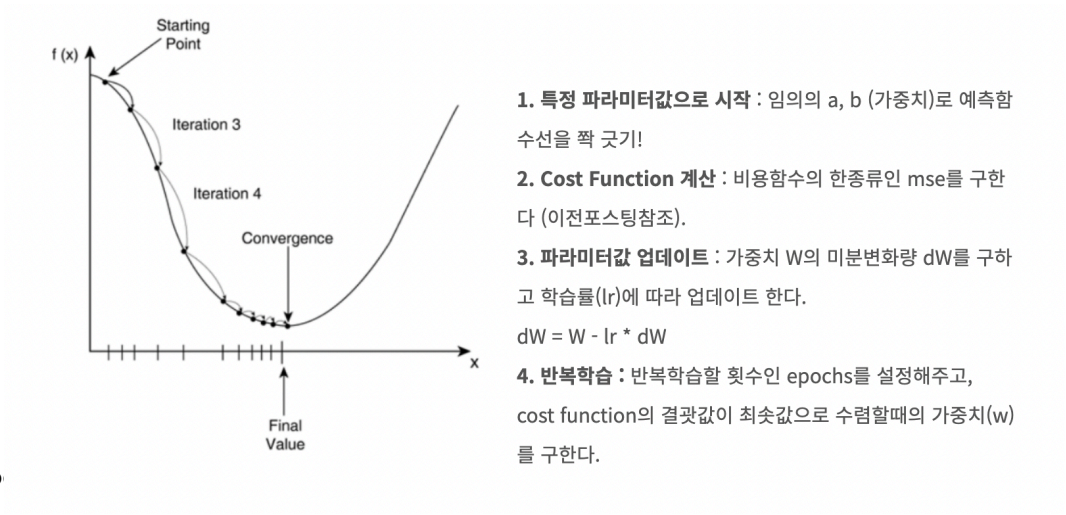
```
static double calcMSE(){
   double y_hat;
   int p = vswr.size();
   double mse = 0.0;

   for(int i=0 ; i<p ; i++){
      y_hat = a*vswr.get(i) + b;                    //예측값
      mse += Math.pow((y_hat - s11.get(i)), 2);    //(예측값. - 실제 값)^2의 합
   }

   mse = mse/p;

   //return mse;
   return Math.round(mse*10)/10.0;                  // 2째 자리에서 반올림
}
```

# 손실함수를 최소화하는 a와 b

여기서 a는 선형회귀 모델에서 직선의 기울기이며, 이는 가중치(w)로 표현되기도 한다. 손실함수는 알고리즘의 오차를 나타내는데, 이 오차가 최소가 되는 지점의 가중치를 찾아 최적의 예측함수를 찾는 것이 목표이다. 즉, MSE를 가장 적게 만들어 주는 직선의 변수들을 찾는 것이 목표이다.

먼저 MSE의 오차를 수정하는 대표적인 방법인 경사하강법을 구현하여 손실함수를 최소화 해 보겠다.

(그림3)




**1. 특정 파라미터값으로 시작** : 임의의 a, b (가중치)로 예측함수선을 쫙 긋기!

**2. Cost Function 계산** : 비용함수의 한종류인 mse를 구한다 (이전포스팅참조).

**3. 파라미터값 업데이트** : 가중치 W의 미분변화량 dW를 구하고 학습률(lr)에 따라 업데이트 한다.

dW = W - lr * dW

**4. 반복학습 :** 반복학습할 횟수인 epochs를 설정해주고, cost function의 결괏값이 최솟값으로 수렴할때의 가중치(w)를 구한다.

Rep

경사하강법으로 최적의 a(=w)와 b찾기 MetamarterialAntenna.java

```
static void gradientDescent(double lr, int epochs){
   a=0.0;
   b=0.0;
   ArrayList<Double> err;

   for(int i=0 ; i<epochs ; i++){
      err = new ArrayList<>();
      double sum=0.0;
      double sum_err=0.0;

      for(int j=0 ; j<vswr.size() ; j++){
         double y_hat = a*vswr.get(j)+b;
         err.add(y_hat - s11.get(j));

         sum_err += err.get(j);

         sum += err.get(j) * vswr.get(j);
      }

            //오차함수를 a, b로 미분한 값, 학습률에 따라 이동한 a, b의 변화량을 구하기 위함.

      double a_diff = (2.0/vswr.size()) * sum;
      double b_diff = (2.0/vswr.size()) * sum_err;

      a -= lr * a_diff;
      b -= lr * b_diff;
      if(i%100 == 0){
         System.out.println("epochs: "+i+", 기울기: "+a+", 절편: "+b);
      }
   }
}
```

실행 MetamarterialAntenna.java

```
gradientDescent(0.03, 2001);
```

반복 횟수(epochs)는 2001회, 학습률(lr)0.03

```
Regression Line : y = 21.1x + (-60.3)
MSE : 1684.3
epochs: 0, 기울기: -2.013919309906102, 절편: -0.966296855968533
epochs: 100, 기울기: -0.2588307695910805, 절편: -15.08980748471233
epochs: 200, 기울기: 0.024149499330982968, 절편: -16.11967614522586
epochs: 300, 기울기: 0.04560923233772813, 절편: -16.197775959643387
epochs: 400, 기울기: 0.047236625450594844, 절편: -16.203698638180875
epochs: 500, 기울기: 0.04736003837204087, 절편: -16.204147782923666
epochs: 600, 기울기: 0.04736939735792935, 절편: -16.2041818436951
epochs: 700, 기울기: 0.047370107094120824, 절편: -16.20418442668459
epochs: 800, 기울기: 0.047370160916776184, 절편: -16.204184622564913
epochs: 900, 기울기: 0.047370164998402436, 절편: -16.20418463741944
epochs: 1000, 기울기: 0.04737016530793087, 절편: -16.204184638545932
epochs: 1100, 기울기: 0.04737016533140444, 절편: -16.20418463863136
epochs: 1200, 기울기: 0.04737016533318261, 절편: -16.20418463863783
epochs: 1300, 기울기: 0.047370165333314146, 절편: -16.204184638638306
epochs: 1400, 기울기: 0.047370165333314146, 절편: -16.204184638638306
epochs: 1500, 기울기: 0.047370165333314146, 절편: -16.204184638638306
epochs: 1600, 기울기: 0.047370165333314146, 절편: -16.204184638638306
epochs: 1700, 기울기: 0.047370165333314146, 절편: -16.204184638638306
epochs: 1800, 기울기: 0.047370165333314146, 절편: -16.204184638638306
epochs: 1900, 기울기: 0.047370165333314146, 절편: -16.204184638638306
epochs: 2000, 기울기: 0.047370165333314146, 절편: -16.204184638638306
MSE : 62.2
```

다음과 같이 기울기는 0.047370165333314146, 절편은 -16.204184638638306에 수렴한다. 처음 수식을 통해 구한 Regression Line보다 손실 함수 최적화가 많이 된 것을 확인했다.

다음은 SA알고리즘을 통하여 손실함수(MSE)를 최소화하는 a와 b를 찾아보자.

SA알고리즘을 이용하여 손실함수(MSE)를 최소화하는 a와 b를 찾기

---

```
//Simaulated Annelaring
public static void SimulatedAnnealing() {
   double temperature = 1000;
   double coolingFactor = 0.995;

   //수식으로 Regression Line의 a와 b에서 bset가  어떻게  나오는지  알기  위해 a와 b로  초기화
       double cur_a = a, cur_b = b;
   double best_a= 0.0, best_b = 0.0;


   for (double t = temperature; t > 1; t *= coolingFactor) {
      double next_a = cur_a * Math.random();
      double next_b = cur_b * Math.random();

      if (Math.random() < probability(calcMSE(cur_a, cur_b), calcMSE(next_a, next_b), t)) {
         cur_a = next_a;
         cur_b = next_b;
      }

      if (calcMSE(cur_a, cur_b) < calcMSE(best_a, best_b)) {
         best_a = cur_a;
         best_b = cur_b;
      }

   }
   System.out.println("Final a(=w): "+best_a+", Final b: "+best_b+", Final MSE: " +
calcMSE(best_a, best_b));
}

//두 번째  MSE가 첫 번째  MSE보다 짧은 경우 첫 번째  MSE를 유지.  그렇지 않으면 두 번째 둘러보기를 수락
할 확률을 반환.

public static double probability(double f1, double f2, double temp) {
   if (f2 < f1) return 1;
   //System.out.println(Math.exp((f1 - f2) / temp));
   return Math.exp((f1 - f2) / temp);
}
```

실행문

---

```
calcRegLine();   //수식으로  구한 회귀 직선의 a와 b에서 bset가  어떻게  나오는지  알기  위해  실행했다.
SimulatedAnnealing();
```

시작온도(temperature = 1000), 냉각률(coolingFactor = 0.995)

---

```
Final a(=w): 0.008456321485042181, Final b: -0.05955998566404075, Final MSE: 320.3
```

시작온도(temperature = 2000), 냉각률(coolingFactor = 0.5)

---

```
Final a(=w): 1.2306466739365787, Final b: -14.6541259546308, Final MSE: 83.6
```

시작온도(temperature = 3000), 냉각률(coolingFactor = 0.4)

---

```
Final a(=w): 0.0402350633933724, Final b: -15.429705906438913, Final MSE: 62.8
```

시작 온도를 높이고 냉각률을 낮추어 주어 더 많은 횟수를 반복시키고, local bset에서도 다른 변수를 찾아볼 확률을 높였더니 global best에 좀 가까워 진 것 같다.

여러 번 실행 해 보며 나오는 여러 해를 보았지만 경사 하강법으로 구한 MSE(=62.2)보다 작게 나오진 않았다...

```
 double next_a = cur_a * Math.random();
double next_b = cur_b * Math.random();
```

부분에서 변화율을 조금 조정하면 좀 더 최적해에 근사하게 나오지 않을까해서 시도해 보았지만 알 수 없는 에러가 발생했다.

# 전체 코드

GoCSV.java

```
package LinearRegression;

import java.io.*;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;

public class GoCSV {

   private String filePath;
   private BufferedReader bufferedReader;
   private List<String[]> readCSV;

   private int index;

   //This constructor is for read CSV File
   public GoCSV(String filePath) throws IOException {
      this.filePath = filePath;
      bufferedReader = new BufferedReader(new InputStreamReader(new
FileInputStream(this.filePath), "UTF-8"));
      readCSV = new ArrayList<>();

      makeList(bufferedReader);
      this.index = 1;     //1열(domain)빼고  반환
   }

   public void makeList(BufferedReader bufferedReader) throws IOException {
      String line = null;
      while((line = bufferedReader.readLine())!=null) {
         String[] lineContents = line.split(",(?=([^\"]*\"[^\"]*\")*[^\"]*$)",-1);
         String[] arr = {lineContents[8], lineContents[10]};
         //readCSV.add(lineContents);
         readCSV.add(arr);
      }
   }
       //한 행을 읽음
       public String[] nextRead(){
      if(readCSV.size() == index){
         return null;
```

```
        }
        return readCSV.get(index++);
    }
}
```

MetamarterialAntenna.java

---

```
package LinearRegression;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.StringTokenizer;

public class MetamarterialAntenna {
    static ArrayList<Double> vswr;
    static ArrayList<Double> s11;
    static double a, b;

    public static void main(String[] args) throws IOException {
        StringTokenizer st;
        GoCSV goCSV = new GoCSV("/Users/jihyeok/정지혁  프로젝트/study/Algorithm/
src/LinearRegression/antenna.csv");
        String[] line = null;
        vswr = new ArrayList<>();
        s11 = new ArrayList<>();
        while ((line = goCSV.nextRead()) != null) {
            vswr.add(Double.parseDouble(line[0]));
            s11.add(Double.parseDouble(line[1]));
            /*for(String a : line){
                System.out.print(a+" ");
            }
            System.out.println();
            */
        }
        Collections.sort(vswr);

        calcRegLine();
        System.out.println("Regression Line : y = " + a + "x + (" + b + ")");
        System.out.println("MSE : " + calcMSE(a, b));

        gradientDescent(0.03, 2001);
```

```
        System.out.println("MSE : " + calcMSE(a, b));

        calcRegLine();
        SimulatedAnnealing();
    }

    static void calcRegLine() {
        double mx = 0, my = 0;
        int p = vswr.size();
        for (int i = 0; i < p; i++) {
            mx += vswr.get(i);
            my += s11.get(i);
        }
        mx = mx / p;
        my = my / p;

        double x = 0;
        double y = 0;

        for (int i = 0; i < p; i++) {
            x += Math.pow((vswr.get(i) - mx), 2);   // ((x - x의 평균)의 제곱)의 합
                    y += (vswr.get(i) - mx) * (s11.get(i) - my);
        }
        a = x / y;
        b = my - a * mx;

        //소수점 2자리에서 반올림
                a = Math.round(a * 10) / 10.0;
        b = Math.round(b * 10) / 10.0;
    }

    static double calcMSE(double a, double b) {
        double y_hat;
        int p = vswr.size();
        double mse = 0.0;

        for (int i = 0; i < p; i++) {
            y_hat = a * vswr.get(i) + b;
            mse += Math.pow((y_hat - s11.get(i)), 2);
        }
        mse = mse / p;
        //return mse;
        return Math.round(mse * 10) / 10.0;
    }

    static void gradientDescent(double lr, int epochs) {
```

```
        a = 0.0;
        b = 0.0;
        ArrayList<Double> err;

        for (int i = 0; i < epochs; i++) {
            err = new ArrayList<>();
            double sum = 0.0;
            double sum_err = 0.0;

            for (int j = 0; j < vswr.size(); j++) {
                double y_hat = a * vswr.get(j) + b;
                err.add(y_hat - s11.get(j));

                sum_err += err.get(j);

                sum += err.get(j) * vswr.get(j);
            }

            //오차함수를 a, b로 미분한 값, 학습률에 따라 이동한 a, b의 변화량을 구하기 위함.

            double a_diff = (2.0 / vswr.size()) * sum;
            double b_diff = (2.0 / vswr.size()) * sum_err;

            a -= lr * a_diff;
            b -= lr * b_diff;
            if (i % 100 == 0) {
                System.out.println("epochs: " + i + ", 기울기: " + a + ", 절편: " + b);
            }
        }
    }

    //Simaulated Annelaring
    public static void SimulatedAnnealing() {
        double temperature = 3000;
        double coolingFactor = 0.4;

        //수식으로 Regression Line의 a와 b에서 bset가 어떻게 나오는지 알기 위해 a와 b로 초
기화
                double cur_a = a, cur_b = b;
        double best_a= 0.0, best_b = 0.0;


        for (double t = temperature; t > 1; t *= coolingFactor) {
            double next_a = cur_a * Math.random();
            double next_b = cur_b * Math.random();
```

```
        if (Math.random() < probability(calcMSE(cur_a, cur_b), calcMSE(next_a, next_b),
t)) {
            cur_a = next_a;
            cur_b = next_b;
        }

        if (calcMSE(cur_a, cur_b) < calcMSE(best_a, best_b)) {
            best_a = cur_a;
            best_b = cur_b;
        }

    }
    System.out.println("Final a(=w): "+best_a+", Final b: "+best_b+", Final MSE: " +
calcMSE(best_a, best_b));
  }

    //두 번째 MSE가 첫 번째 MSE보다 짧은 경우 첫 번째 MSE를 유지. 그렇지 않으면 두 번째
둘러보기를 수락할 확률을 반환.

  public static double probability(double f1, double f2, double temp) {
    if (f2 < f1) return 1;
    //System.out.println(Math.exp((f1 - f2) / temp));
    return Math.exp((f1 - f2) / temp);
  }
}
```

## 출처

### 그림

(그림1) - TechForum (한국 기술지원 포럼) - 칩 안테나 선택 시 반사 손실대 VSWR
https://forum.digikey.com/t/return-loss-vswr/19144/1
(그림2) - [RF기초-6탄] VSWR(정재파비)란? / RF열무의 라이프/ 2020.8.17
https://rf-yeolmu.tistory.com/9
(그림3) - 모두의 딥러닝 4장. 오차수정하기: 경사하강법 / 효니루 / 2020.9.29
https://bookandmed.tistory.com/12

## 참고 글


위키백과 - 메타물질
https://ko.wikipedia.org/wiki%EB%A9%94%ED%83%80_%EB%AC%BC%EC%A7%88

TechForum (한국 기술지원 포럼) - 칩 안테나 선택 시 반사 손실대 VSWR
https://forum.digikey.com/t/return-loss-vswr/19144/1

Linear Model(선형회귀) - ayleen9506 / https://velog.io/@ayleen9506/Linear-Model-%EC%84%A0%ED%98%95%ED%9A%8C%EA%B7%80

Linear Regression - 단순선형회귀 정리, 회귀계수와 절편 구하기 / **dlskawns**·2021년 8월 11일 /
https://velog.io/@dlskawns/Linear-Regression
%EB%8B%A8%EC%88%9C%EC%84%A0%ED%98%95%ED%9A%8C%EA%B7%80-
%EB%8B%A4%EC%A4%91%EC%84%A0%ED%98%95%ED%9A%8C%EA%B7%80-
%EA%B0%84%EB%8B%A8%ED%95%9C-%EC%A0%95%EB%A6%AC