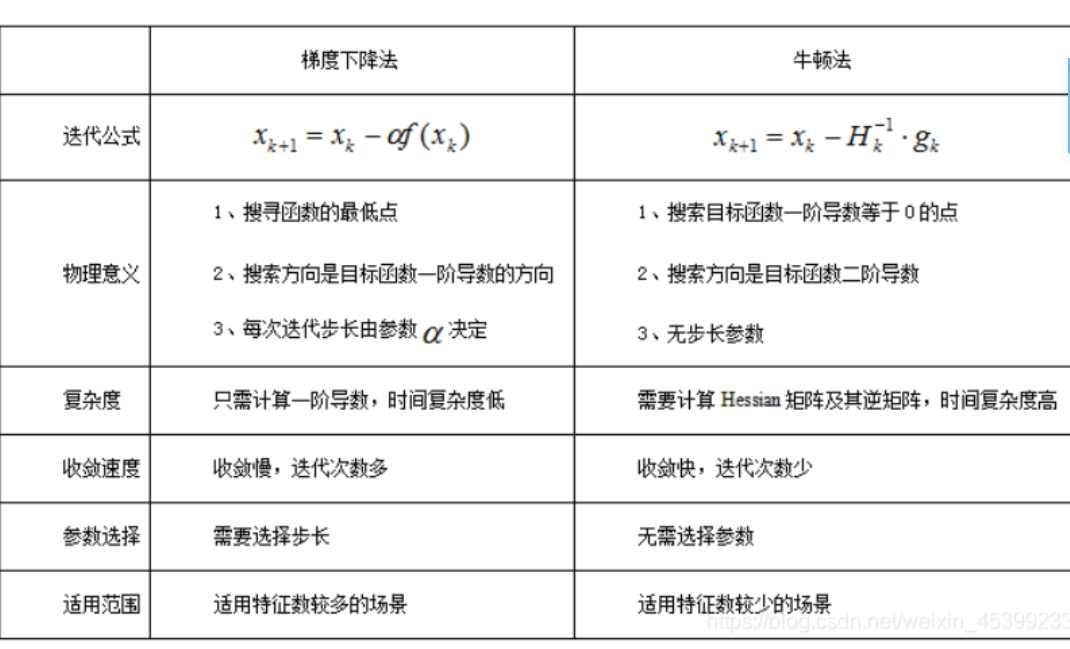
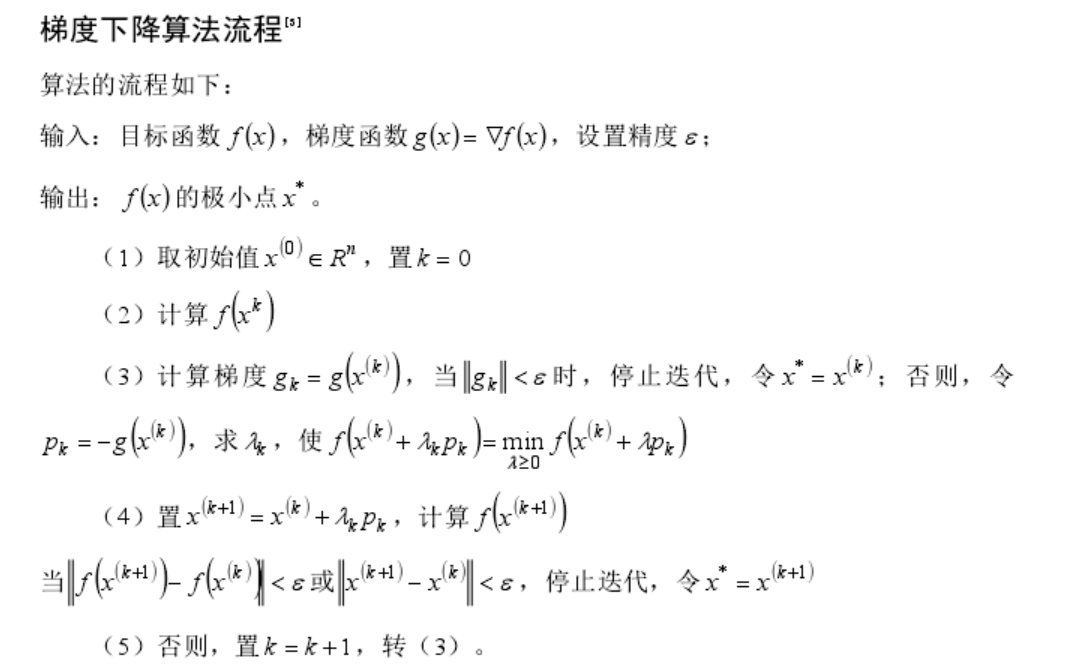
梯度下降的相关：



# 目的

采取梯度下降获得40个本征模式的复数系数。

# 方法介绍：

梯度下降法是一种迭代算法。选取适当的初值x0 ，不断迭代，更新x的值，进行目标函数的极小化，直到收敛。由于负梯度方向是使函数值下降最快的方向，在迭代的每一步，以负梯度方向更新x的值，从而达到减少函数值的目的。

## 梯度下降使用的分类：

Batch Gradient Descent，BGD批量梯度下降 （所有的点都拿来计算）

Stochastic Gradient Descent，SGD 随机梯度下降 （随机挑一个点计算）

Mini-batch gradient descent 小批量梯度下降 （随机选几个点来计算）

前面采用pytorch调用的话，框架里应该是采取的BGD梯度下降法。但是效果不好。BGD自己写梯度公式，涉及的点变量太多，BGD写起来太复杂，所以下面介绍的是SGD。

以下的loss函数采取的是mse函数形式。

# 数据介绍：

### 所求变量：

四十个系数，分为实部虚部，所以定义为k=[40,2]的数组，里面每一个元素都是一个实数变量，所以一共80个变量需要求解。

数据点，40个模式里面取27个点，每个点有三个分量方向，每个分量上有一个复数值，所以数据为point=[40,27,3,2] (2是实部、虚部两个值，都写成实数值好用来梯度下降)

### loss函数:

设置预设点的数值[27,1]，与40个系数的最终加和结果[27,1]进行MSE误差求和。

### 预设点设置：

中心点一个数值1，从距离中心点0.33倍λ位置（λ是光的波长），开始设置多个零点（26个）。

### 求和过程：

某一点为例：

ex分量各自本征模式乘以各自系数求和，ey分量各自本征模式乘以各自系数求和，ez分量各自本征模式乘以各自系数求和，最终得到三个分量方向的各自和，再对三个分量求标量和：

ESUM= EX^2+ EY^2+ EZ^2

### loss函数：

采取的是SGD，取一个点p数据为 [40,1,3,2]。即[40,3,2]

要求的系数k=[40,2]，先随机初始化k。

：求法

ex\_real\_sum=

=系数实数值\*点的实数值 +（-1） \* 系数虚数值\*点的虚数值

= k[0,0]\*p[0, 0,0]+ k[1,0]\*p[1, 0,0]+ k[2,0]\*p[2, 0,0]+```+ k[39,0]\*p[39,0,0]+

(-1)\*( k[0,1]\*p[0,0,1]+ k[1,0]\*p[1,0,1]+ k[2,0]\*p[2,0,1]+```+ k[39,0]\*p[39,0,1])

ex\_img\_sum=

=系数实数值\*点的虚数值 + 系数虚数值\*点的实数值

= k[0,0]\*p[0, 0,0]+ k[1,0]\*p[1, 0,0]+ k[2,0]\*p[2, 0,0]+```+ k[39,0]\*p[39,0,0]+

( k[0,1]\*p[0,0,1]+ k[1,0]\*p[1,0,1]+ k[2,0]\*p[2,0,1]+```+ k[39,0]\*p[39,0,1] )

同理ey,ez两部分

ex\_sum = (ex\_real\_sum^2 + ex\_img\_sum^2)^0.5

ey\_sum = (ey\_real\_sum^2 + ey\_img\_sum^2)^0.5

ez\_sum = (ez\_real\_sum^2 + ez\_img\_sum^2)^0.5

**所以有**

ESUM= EX^2+ EY^2+ EZ^2=

(ex\_real\_sum^2 + ex\_img\_sum^2)+(ey\_real\_sum^2 + ey\_img\_sum^2)+ (ez\_real\_sum^2 + ez\_img\_sum^2)

求ESUM与Epreset的距离作为loss函数。（假设取得这个点的Epreset设置强度是1）

loss = （Epreset -ESUM）^2

=（1 - ESUM）^2

（其实若是BGD就是loss = （Epreset -ESUM）^2 + (Epreset -ESUM）^2 +```多个点）

### 梯度求法：

求第一个的系数元素k[0,0]的偏导

= 2\*(1 - ESUM)\*(-1)\*()

又 = k[0,0]\*p[0, 0,0]+ k[1,0]\*p[1, 0,0]+ k[2,0]\*p[2, 0,0]+```+ k[39,0]\*p[39,0,0]+

(-1)\*( k[0,1]\*p[0,0,1]+ k[1,0]\*p[1,0,1]+ k[2,0]\*p[2,0,1]+```+ k[39,0]\*p[39,0,1])

有p[0, 0,0]

所以= p[0, 0,0]

同理求得，，，

即求得

即求得 = 2\*(1 - ESUM)\*(-1)\*()表达式

为loss函数关于第一个元素k[0,0]系数的偏导数，

同理求得k[40,2]的所有80个元素的偏导数，构成梯度矩阵loss也为[40,2]的数组

### 更新过程：

设置max\_epoch更新次数最大值 例如100000次，可接受最小误差eplison例如0.0001

说明：每一次更新系数只随机取一个点来计算梯度，然后更新

，计算总的误差要计算所有的点与对应预设点的值作剂里求和。

每一次循环选择27个点中的一个点来计算梯度，然后更新所有系数。

当更新次数大于max\_epoch或者计算总的误差小于eplison，停止更新。

已知梯度更新

　k\_new = k\_old− learning\_rate \* loss

### learning\_rate的设置：

learning\_rate 可以设置定值或者变化值

#### 定值：

例如设置learning\_rate=0.001

#### 变化值：

变化值设置方法初始想的有两种:

1. 由大逐渐变小，例如开始设置0.1，当下一次的损失函数得到的值>=上一次的损失函数得到的值时，说明损失函数下山下不去了，此时变化学习率改变为0.01，同理继续更新循环。
2. 大小交替，学习率设置周期变化，每10次循环里可以设置第一次的循环的学习率0.1，然后是中等大小的学习率0.01设置两个循环，然后设置7个循环为小的学习率0.001。即设置一个周期是

[0.1, 0.01, 0.01 , 0.001,0.001,0.001,0.001,0.001,0.001,0.001]依次赋值给每个循环的学习率的值。 这个方法猜测，如果设置合理的话可能可以避免陷入，局部的极小值点。

以上的介绍是采取梯度下降法里面的SGD单个点的求解及更新系数的方式。

若是采取BGD或者Mini-batch gradient descent过程类似，只是，若每一次利用的点越多，梯度写起来就变复杂些。

这个SGD还没实现试过，应该比先前想的梯度下降法BGG更容易写成代码，只是效果不知道好不好。。

二维方向设置零点的求解结果，用的是以前的方法的某次结果。

求解结果：这个是1000精度的图，波导的尺寸还是4000nm，所以图里面的100长度对应一个波长800nm。

设置二维的超振荡

