梯度下降法

分析

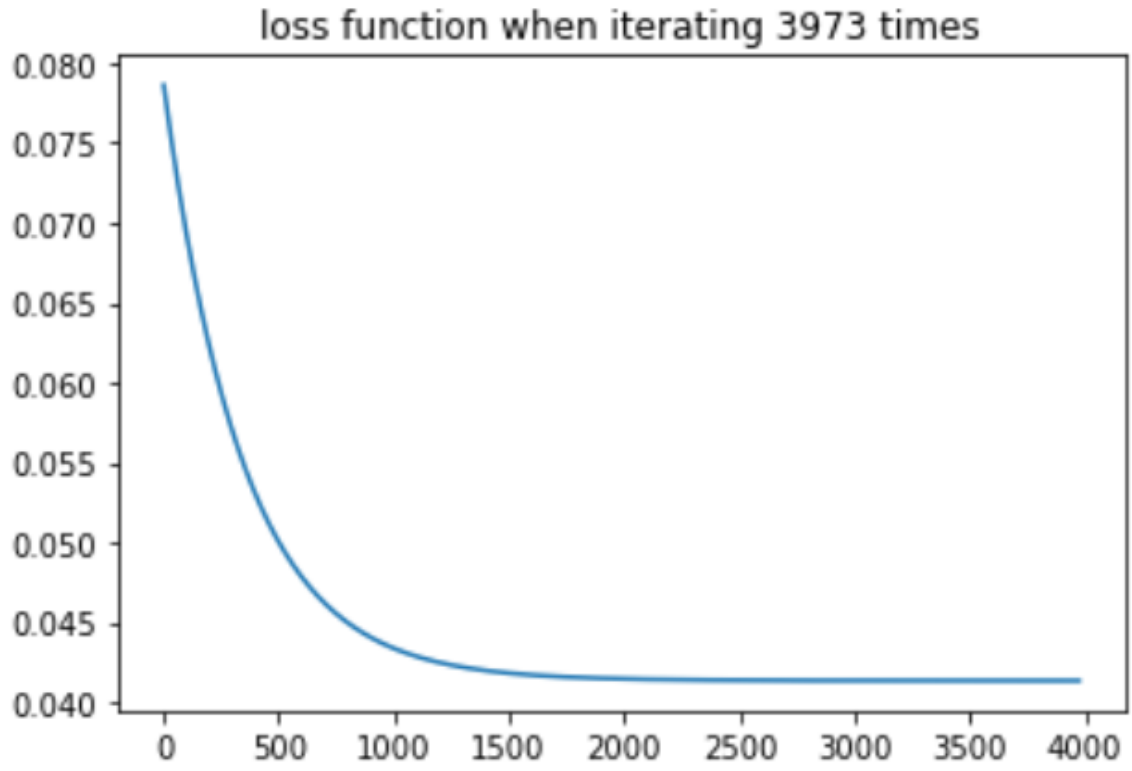
选取损失函数

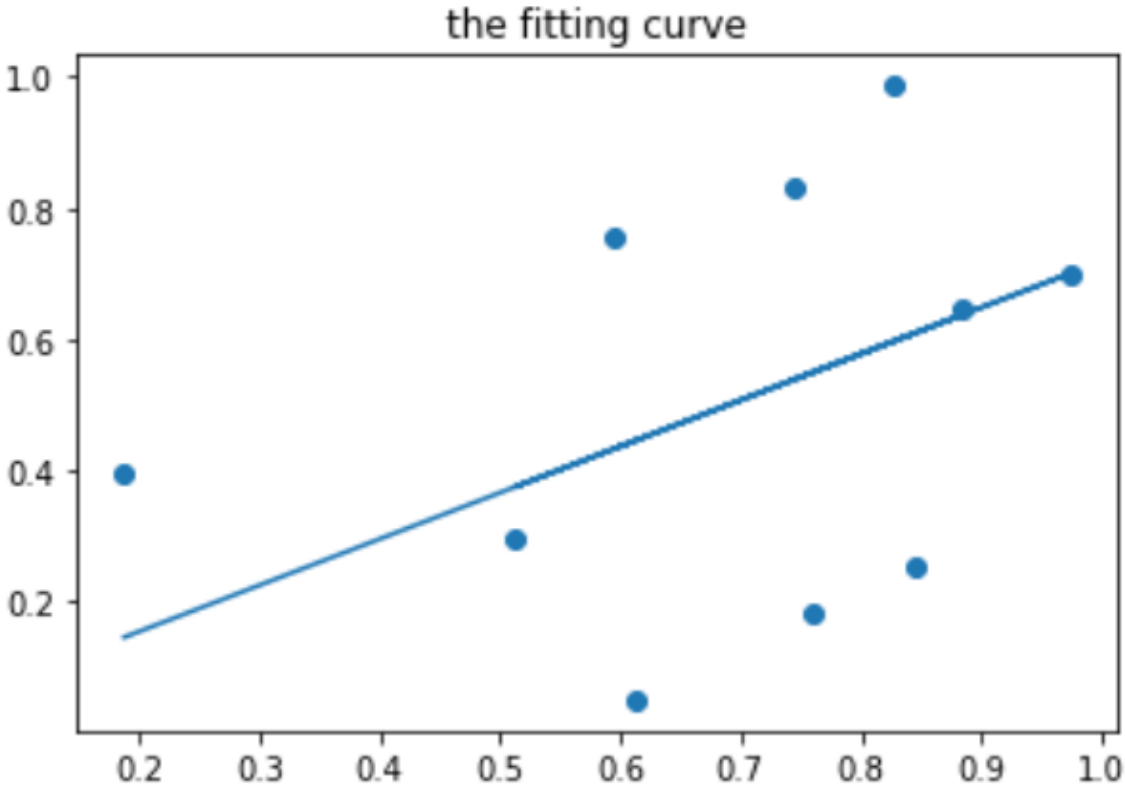
$$Loss = \frac{1}{2}(wx + b - y)^2 \tag{1}$$

程序

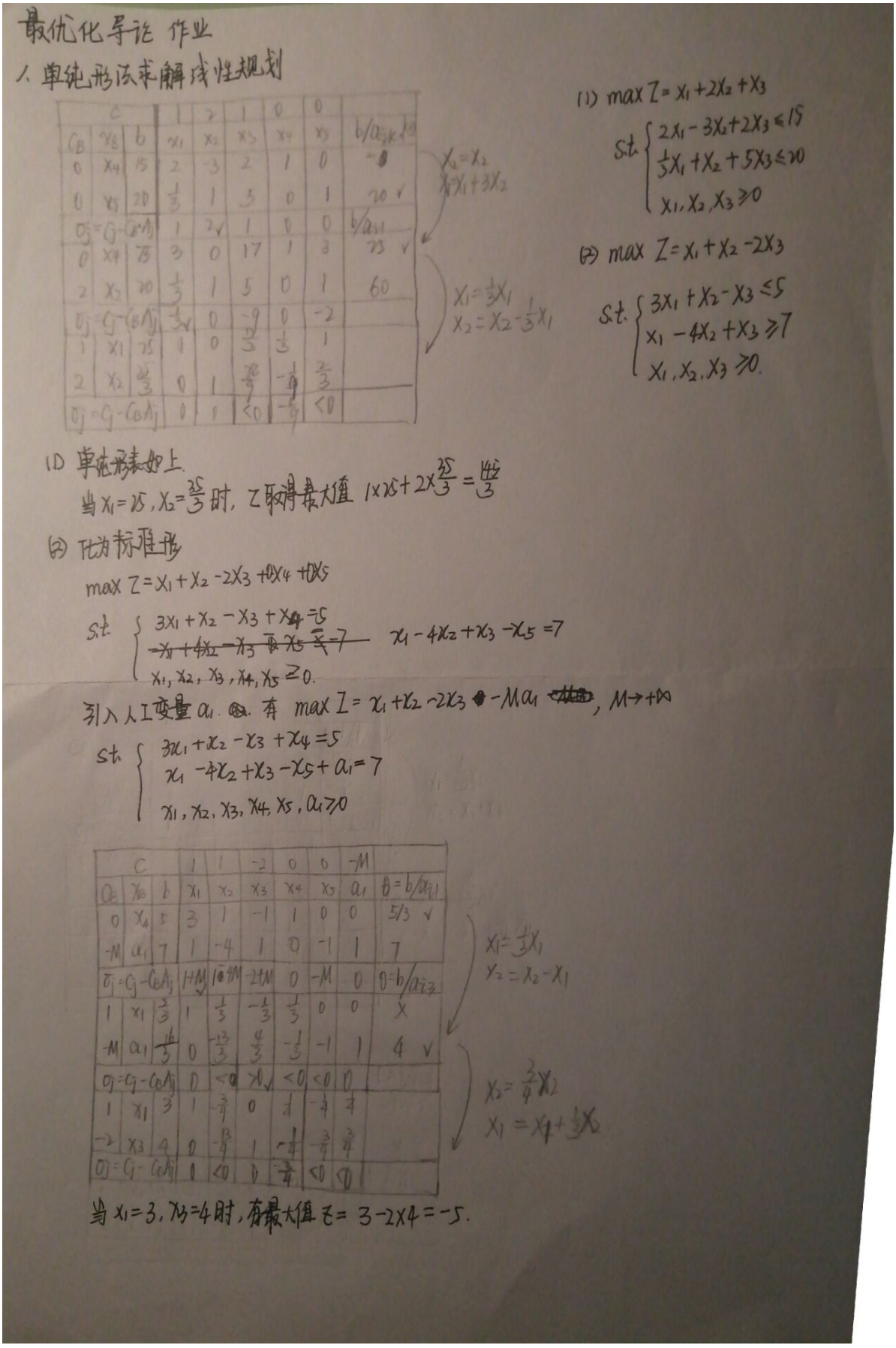
```python
import numpy as np
import matplotlib.pyplot as plt
# 参数定义
x = np.random.random(10)
y = np.random.random(10)
lr = 0.001
iter_num = 5000
e = 0.000001

def loss_fun(w):
    return np.mean((w[0]*x+w[1]-y)**2)/2

def df(w):
    return np.mean([x*(w[0]*x+w[1]-y),w[0]*x+w[1]-y])

def tiduxiajiang(w1):
    loss = []
    for i in range(1,iter_num):
        loss.append(loss_fun(w1))
        w2 = w1 - lr*df(w1)
        if np.sqrt(np.sum((w1-w2)**2)) < e:
            #     print(loss)
            plt.figure(1)
            plt.plot(range(1,i+1),loss,"-")
            plt.title("loss function when iterating {:d} times".format(i))
            return (w1+w2)/2
        w1 = w2
    plt.figure(1)
    plt.plot(range(1,i+1),loss,"-")
    plt.title("loss function in the end")
    return 0


if __name__=="__main__":
    # np.random.seed(612) # 设置随机数种子
    w0 = np.random.random(2)
    w = tiduxiajiang(w0)
    plt.figure(2)
    plt.scatter(x,y)
    plt.plot(x,x*w[0]+w[1],'-')
    plt.title('the fitting curve')
```

结果


loss function when iterating 3973 times
0.080
0.075
0.070
0.065
0.060
0.055
0.050
0.045
0.040
0
500
1000
1500
2000
2500
3000
3500
4000


the fitting curve
1.0
0.8
0.6
0.4
0.2
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

单纯形法


最优化导论 作业
一、单纯形法求解线性规划
(1) max Z = x1 + 2x2 + x3
s.t. 2x1 - 3x2 + 2x3 ≤ 15
1/3 x1 + x2 + 5x3 ≤ 20
x1, x2, x3 ≥ 0
(2) max Z = x1 + x2 - 2x3
s.t. 3x1 + x2 - x3 ≤ 5
x1 - 4x2 + x3 ≥ 7
x1, x2, x3 ≥ 0.
x2 = x2
x4 = x4 + 3x2
x1 = 1/3 x1
x2 = x2 - 1/3 x1
(1) 单纯形表如上.
当 x1 = 15, x2 = 25/3 时, Z 取得最大值 1×15 + 2×25/3 = 115/3
(2) 化为标准形
max Z = x1 + x2 - 2x3 + 0x4 + 0x5
s.t. 3x1 + x2 - x3 + x4 = 5
x1 - 4x2 + x3 - x5 = 7
x1, x2, x3, x4, x5 ≥ 0.
引入人工变量 a1. 有 max Z = x1 + x2 - 2x3 - Ma1, M → +∞
s.t. 3x1 + x2 - x3 + x4 = 5
x1 - 4x2 + x3 - x5 + a1 = 7
x1, x2, x3, x4, x5, a1 ≥ 0
x1 = 1/3 x1
x2 = x2 - x1
x3 = 1/4 x2
x1 = x1 + 1/3 x2
当 x1 = 3, x3 = 4 时, 有最大值 Z = 3 - 2×4 = -5.

动态规划

1、方案设计题

设$dp[i][j]$表示j台设备分配给前i个人的可获利润；

$p[i][j]$表示j台设备分配给第i个人的可获利润；

则有动态转移方程如下

$$dp[i][j] = max_{\ 0\le k\le j}\{\ p[i][k] + dp[i-1][j-k]\ \} \tag{2}$$

动态转移过程如下，按照$i:1\rightarrow 3, j:6\rightarrow 0$的顺序从上至下，从右至左维护矩阵。

	设备数j						
分配前i人	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	4	9	12	14	16	19
最优策略	0	1	2	3	4	5	6
2	0	4	9	12	17	20	24
最优策略	0,0	1,0	2,0	1,2 2,1 3,0	2,2	2,3 3,2	2,4
3	0	5	10	14	19	22	27
最优策略	0,0,0	0,0,1	0,0,2	1,0,2 2,0,1	2,0,2	2,2,1	2,2,2

有矩阵结果知，$dp[3][6]=27$，即当设备分配如$(A,B,C)=(2,2,2)$时，具有最大总利润27万元。

2、编程题

分析

由题意知，金矿要么不挖，要么全挖，为**01背包**问题，其中

$C=10$	1	2	3	4	5
w	3	4	3	5	5
v	200	300	350	400	500

程序

```cpp
#include<bits/stdc++.h>
using namespace std;
//01背包问题
const int N = 5, C = 10;
int w[]={0,3,4,3,5,5};
int v[]={0,200,300,350,400,500};
//int dp[C+1];  // 用一维矩阵维护
```

```
int dp[N+1][C+1]; // 二维矩阵维护
int path[N+1];

void package01(){
    for (int i=1;i<=N;i++){
        for (int j=w[i];j<=C;j++){
            dp[i][j] = dp[i-1][j];
            if(dp[i-1][j] < dp[i-1][j-w[i]] + v[i])
                dp[i][j] = dp[i-1][j-w[i]] + v[i];
        }
    }
    printf("最优策略为：");
    int t=C;
    for(int i=N;i>=1;i--){
        if(dp[i][t]==dp[i-1][t-w[i]]+v[i]){
            cout << "1 ";
            t = t-w[i];
        }else cout << "0 ";

    }
    printf("\n总价值为：%d",dp[N][C]);
}
int main(){
    package01();
    return 0;
}
```

结果

```
最优策略为：1 1 0 0 0
总价值为：900
```