**《最优化技术》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2019级、计算机科学与技术、计科01班** | | | **姓名** | **谢双骏** |
| **实验题目** | **平时作业** | | | | | |
| **实验时间** | **2021年5月27日** | | **实验地点** | **DS3401** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 ■设计性 □综合性** | | |
| **教师评价：**  **□算法/实验过程正确；□源程序/实验内容提交 □程序结构/实验步骤合理；**  **□实验结果正确； □语法、语义正确； □报告规范；**  **其他：**  **评价教师签名：** | | | | | | |
| **一、实验目的**  **平时作业，检测学习效果** | | | | | | |
| **二、实验内容**  **平时作业_00** | | | | | | |
| 1. **实验过程或算法（源程序）** 2. **梯度下降法拟合曲线**   **1.基本思想：**  设拟合的曲线为：  然后的设解为：  首先表示出拟合曲线的y值也就是：  然后设置目标函数：  然后表示出梯度的导数：  最后根据迭代公式：  在满足迭代条件前一直迭代，不断更新梯度和拟合曲线的y值，最后得到  **2.代码**  # coding:utf-8 import numpy as np import matplotlib.pyplot as plt from numpy.core.numeric import ones # 显示中文 plt.rcParams['font.sans-serif']=['SimHei'] plt.rcParams['axes.unicode\_minus'] = False # 梯度计算 def grad\_Calculation(p, x, y):  temp = np.dot(x, p) - y  result = 1 / m \* np.dot(np.transpose(x),temp)  return result # 损失度计算 def loss\_degree(p, x, y):  temp = np.dot(x, p) - y  result = 1 / (2\*m) \* np.dot(np.transpose(temp), temp)  return result # 输入点的个数，可以自定义 m = 10 # 构造矩阵 #x0 x0 = np.ones((m,1)) #x1 x1 = np.arange(1, m+1).reshape(m, 1) #拼接x0和x1 x = np.hstack((x0, x1)) #生成y的值 y = np.random.randint(0, 30, size=(m, 1)) # 对y进行排序的操作，这样的点分布对于我们进行线性回归的拟合很有帮助 y.sort(axis=0) # 设置学习率 a = 0.01 # 存储迭代次数 generation = 0 # 设置解的初始值 p = np.array([1, 1]).reshape(2, 1) # 计算初始的梯度 grad = grad\_Calculation(p, x, y) # 计算初始的损失值 loss = loss\_degree(p, x, y) # 存储损失值，方便之后画图 loss\_store = [] # 先扁平化得到一维矩阵再转换为列表 loss = loss\_degree(p,x,y).flatten().tolist() # 这里用extend不用append loss\_store.extend(loss) generation += 1 # 设置可允许的误差的范围 e = 0.00001 # 开始迭代 #当梯度的值大于误差的时候，一直执行 while np.all(np.absolute(grad) > e):  # 迭代公式  p = p - a \* grad  # 更新grad  grad = grad\_Calculation(p, x, y)  #更新loss  loss = loss\_degree(p, x, y)  loss = loss\_degree(p,x,y).flatten().tolist()  loss\_store.extend(loss)  #迭代次数+1  generation += 1 # 画图  #左右两个子图 fig,ax = plt.subplots(1,2) fig.suptitle('运用梯度下降算法进行线性回归拟合的图像') fig.dpi = 150 # 得到最佳的p [p0, p1] = p print('最优的两个参数分别为：'+str(round(float(p0),4))+"和"+str(round(float(p1),4)))  # 拟合图像 ax[0].set\_title('梯度下降法拟合图像') ax[0].scatter(x1,y,color='orange') ax[0].plot(x1, p0 + p1\*x1, color='blue') ax[0].grid(False) # 绘制网格线 ax[0].set\_xlabel("x轴") ax[0].set\_ylabel("y轴") # 损失值图像 ax[1].set\_title('损失值图像') li = list(range(generation)) print('迭代次数为：', generation) ax[1].plot(li,loss\_store,color='red') ax[1].grid(True) ax[1].set\_xlabel("迭代次数") ax[1].set\_ylabel("损失值") plt.show()   1. **单纯形法求解**   **1.基本思想**  单纯形法的基本思想就是首先找出一个初始基本可行解，然后检验它是不是最优的，如果不是，转移到另一个基本可行解（找出更大的目标函数值）；如果是，就结束迭代，输出结果。  YB$P{5{_DX`[$M%924`3II6  在求解的部分，运用了人工迭代单纯性表进行求解。   1. **设备分配问题**   **1.基本思想**  这个问题是多阶段决策过程优化，一共有3个用户，所以划分为3个阶段。A,B,C三个用户编号为1，2，3。  我们使用表示分配给第k个用户到第n个用户的设备数量。  使用表示分配给第k个用户的设备数量。  表示第k个用户分配数量时的利润  由此我们可以得到递推的方程：   1. **金矿挖掘问题**   **1.基本思想**  假设worker[i]是挖掘第i个金矿所需的工人， Gold[i]是挖掘第[i]个金矿的收益，dp[i][j]为前i座金矿，现有j个工人的时候，可以挖掘的最大黄金储备量，那么递推关系为：  （1）如果现有的工人数目比当前金矿的所需挖掘人数多，那么： dp[i][j]=max{dp[i-1][j],dp[i-1][j-worker[i]]+Gold[i]}  也就是前i座金矿，现有j个工人可以挖掘的最大黄金储备量等于前i-1座，现有j个工人的可挖掘储备量（不挖第i座金矿）和前i-1座，现有j-当前金矿挖掘人数的储备量 +当前金矿的储备量（挖第i座金矿）这两者之间的最大值  （2）如果现有的工人数目小于当前金矿的所需挖掘人数，那么： dp[i][j]=dp[i-1][j]  （3）如果j==0,或者i==0.也就是没有工人和没有金矿，那么相应的 dp[i][j]==0  **2.代码**  # 自定义金矿列表，第一行表示每一个金矿的存储量，第二行表示挖掘该金矿需要的工人数目， Gold\_mine= [[200, 300, 350, 400, 500], [3, 4, 3, 5, 5]] # 输入工人的数目，方便之后dp列表大小的设定 worker = int(input("请输入工人数量：")) # 标记金矿的数目，方便之后dp列表大小的设定 num = len(Gold\_mine[0]) # 动态规划列表 dp = [] # 记录选择了哪些金矿的列表 select = [] # 初始化dp列表 for i in range(num + 1):  dp.append([])  for j in range(worker + 1):  dp[i].append(0) # 从第一座金矿到最后一座金矿开始遍历 for i in range(1, num + 1):  # 每一座金矿从没有工人到全部工人开始遍历  for j in range(1, worker + 1):  # 如果当前工人数目大于金矿所需的挖掘人数  if j - Gold\_mine[1][i - 1] >= 0:  dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - Gold\_mine[1][i - 1]] + Gold\_mine[0][i - 1])  # 现有工人不够挖掘当前金矿  else:  dp[i][j] = dp[i - 1][j] # 输出可以挖掘的最多的黄金储备 print("可以挖掘的最多的黄金储备是：" + str(dp[num][worker])) # 反向搜索寻找挖掘了那些金矿，原理同0，1背包的选取了哪些物品 while worker > 0 and num > 0:  # 如果工人数目大于现在的金矿的工人需求  if worker - Gold\_mine[1][num - 1] >= 0:  # 如果挖现在金矿的收入比不挖高，那么就把现在的金矿标号i加入select列表，否则不作为  # noinspection PyTypeChecker  if dp[num - 1][worker - Gold\_mine[1][num - 1]] + Gold\_mine[0][num - 1] > dp[num - 1][worker]:  select.append(num)  worker -= Gold\_mine[1][num - 1]  # 无论挖还是不挖，金矿标号-1  num -= 1  # 如果工人数目比现在的金矿的工人需求小，那么肯定不挖，金矿标号-1  else:  num -= 1 # 由于是从后往前寻找，顺序是反向的，运用reverse函数倒置列表，变成正序 select.reverse() # 根据情况输出挖掘了哪些金矿 if len(select) == 0:  print("应该一座金矿都不挖") else:  print("应当挖掘第 ", end='')  for e in select:  print(str(e) + " ", end='')  print("座金矿") | | | | | | |
| 1. **实验结果及分析和（或）源程序调试过程** 2. **梯度下降法**   **1.结果**  **（1）控制台**  最优的两个参数分别为：1.5997和2.2546  迭代次数为： 3390  **（2）图像**    **2.调试过程**  （1）在实验的过程中，最开始并没有使用矩阵的方法去表示数据，而是使用普通列表，然后自己设计算法，最后发现代码的复杂高，并且结果总是出错。后来使用矩阵存储数据，大大简化了代码的复杂度，降低了出错的概率。  （2）在画图的过程中，发现中文无法输出，改变编码方式也无济于事，最后在网上查找资料后发现，只需要补充：  plt.rcParams['font.sans-serif']=['SimHei'] plt.rcParams['axes.unicode\_minus'] = False  即可解决问题   1. **单纯形法**   **1.求解**   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 标准化处理：  约束矩阵：  列出单纯形表： | | | | | | | | | |  | | | 1 | 2 | 1 | 0 | 0 |  | |  | 基变量 |  |  |  |  |  |  | | 0 |  | 15 | 2 | -3 | 2 | 1 | 0 | -5 | | 0 |  | 20 | 1/3 | 1 | 5 | 0 | 1 | 20 | |  | | | 1 | 2 | 1 | 0 | 0 |  | | 初始基本可行解为：  目标函数的最大值为：  进基，出基： | | | | | | | | | | *0* |  | 75 | 3 | 0 | 17 | 1 | 3 | 25 | | *2* |  | 20 | 1/3 | 1 | 5 | 0 | 1 | 60 | |  | | | 1/3 | 0 | -9 | 0 | -2 |  | | 此时基本可行解为：  目标函数的最大值为：  进基，出基： | | | | | | | | | | *1* |  | *25* | 1 | 0 | 17/3 | 1/3 | 1 |  | | *2* |  | *35/3* | 0 | 1 | 28/9 | -1/9 | 2/3 |  | |  | | | 0 | 0 | -98/9 | -1/0 | -7/3 |  | | 检验数都小于0，停止迭代，最优解为：  目标函数的最大值为： | | | | | | | | |   2.  标准化处理：  约束矩阵：  A中没有单位阵可作为初始基，采用大从法，加入人工变量a，变为：  约束矩阵：  列出单纯形表：   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | | | 1 | 1 | -2 | 0 | 0 | -M |  | |  | 基变量 |  |  |  |  |  |  | a | | 0 |  | 5 | 3 | 1 | -1 | 1 | 0 | 0 | 5/3 | | -M | a | 7 | 1 | -4 | 1 | 0 | -1 | 1 | 7 | |  | | | 1+M | -4M | M-2 | 0 | -M | 0 |  |   初始基本可行解为：  进基，出基：   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 1 |  | 5/3 | 1 | 1/3 | -1/3 | 1/3 | 0 | 0 | -5 | | -M | a | 16/3 | 0 | -13/3 | 4/3 | -1/3 | -1 | 1 | 4 | |  | | | 0 | 2/3-13/3M | 4M/3-5/3 | -1/3-M/3 | -M | 0 |  |   此时基本可行解为：  进基，a出基：   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | *1* |  | **3** | 1 | -3/4 | 0 | 1/4 | -1/4 | 1/4 |  | | *-2* |  | *4* | 0 | -13/4 | 1 | -1/4 | -3/4 | 3/4 |  | |  | | | 0 | -19/4 | 0 | -3/4 | -5/4 | 5/4-M |  |   检验数均小于0，停止迭代，有无穷多最优解，其中一个为：  目标函数的最大值为：  **2.调试过程**  最开始的时候，并没有使用的单纯形表进行操作，只是单纯的每一次求取数据进行迭代，发现这样的方法不仅笨重，而且非常容易出错，效果很差。后来改用单纯形表进行操作，大大简化了运算的难度，并且出错的概率降低了，即使出错了，也能发现错误出现在哪里。   1. **设备分配问题**   **1.求解**  很明显这是一个逆推的方法，，很容易理解，也就是第四个到第三个的设备数量，很明显是不存在的，也就是0。  当k=3的时候：  根据题目所提供的利润表，我们可以得到如下的数据,其中是最优解   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |  | 0 | 5 | 10 | 12 | 14 | 16 | 17 |   接下来继续递推k=2的时候：  可以得到递推表格和如下：   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |  |  | |  | | | | | | | | 0 | 0 |  |  |  |  |  |  | 0 | 0 | | 1 | 5 | 3 |  |  |  |  |  | 5 | 0 | | 2 | 10 | 8 | 8 |  |  |  |  | 10 | 0 | | 3 | 12 | 13 | 13 | 11 |  |  |  | 13 | 1或2 | | 4 | 14 | 15 | 18 | 16 | 15 |  |  | 18 | 2 | | 5 | 16 | 17 | 20 | 21 | 20 | 17 |  | 21 | 3 | | 6 | 17 | 18 | 22 | 23 | 25 | 22 | 18 | 25 | 4 |   接下来继续递推k=2的时候：  可以得到递推表格和如下：   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |  |  | |  |  | | | | | | | | 6 | 25 | 25 | 27 | 25 | 24 | 21 | 19 | 27 | 2 |   因为推到第一层的时候，已经不用给更前的用户考虑留存设备了，因此只有设备数量这一个值，所以此时剩下多少设备全部分配给第一个用户就是最大的利润。  最后再从前往后依次递推答案就可以了：  因此有  当时：  当时：  所以A,B,C用户每个人两套设备的利润是最大的。  **2.调试过程**  因为是手动表格求解，调试过程都是一些运算的错误，所以暂不列出。   1. **金矿挖掘问题**   **1.结果**  请输入工人数量：10  可以挖掘的最多的黄金储备是：900  应当挖掘第 4 5 座金矿  **2.调试过程**  （1）最开始的时候使用普通的函数递归，虽然实现起来没有问题，但是自顶向下的递归算法肯定会有重复的计算，所以还是改进为了数组存储数据，然后自底向上执行。  （2）根据递归的条件找出最大的金矿储备是非常容易的，但是寻找挖掘了哪些金矿有一些难度。我尝试了添加其他数组记录轨迹的方法，由于不太熟悉，未能实现。后来发现，根据dp的数组就可以进行寻迹的操作。原理和01背包选取了哪些物品是一样的。具体实现的原理参见源代码注释。 | | | | | | |