数学建模入门-期末小作业

魏少杭
学号：20373594
人工智能研究院

2022 年 11 月 29 日

注：本次作业均用 Python 实现

目录

1 练习 1

1.1 代码

练习 1

```
import numpy as np
import matplotlib.pyplot as plt
BEGIN = -5
END = 5
STEPS = 10000
ori3 = np.array([1, -6, 5, -3])
y = np.poly1d(ori3)
x_ = np.linspace(BEGIN, END, STEPS)
y_ = y(x_)
noise = np.random.normal(0, 1, size=x_.shape)
y__ = y_ + noise
# =========================
# 三次多项式
# =========================
p3 = np.polyfit(x_, y__, 3)
# ================================================
# 二阶多项式
# ================================================
p2 = np.polyfit(x_, y__, 2)
# ================================================
# 四阶多项式
# ================================================
p4 = np.polyfit(x_, y__, 4)

print("原始系数: {}\n, 三阶拟合系数: {}\n, 二阶拟合系数: {}\n, 四阶拟合系
    数: {}".format(ori3, p3, p2, p4))

plt.plot(x_, y_, 'r', label='原始函数')
p3 = np.poly1d(p3)
y_p3 = p3(x_)
plt.plot(x_, y_p3, 'b--', label='三阶拟合效果')
p2 = np.poly1d(p2)
y_p2 = p2(x_)
plt.plot(x_, y_p2, 'y--', label='二阶拟合效果')
p4 = np.poly1d(p4)
y_p4 = p4(x_)
plt.plot(x_, y_p4, 'g--', label='四阶拟合效果')
plt.legend()
plt.show()
```

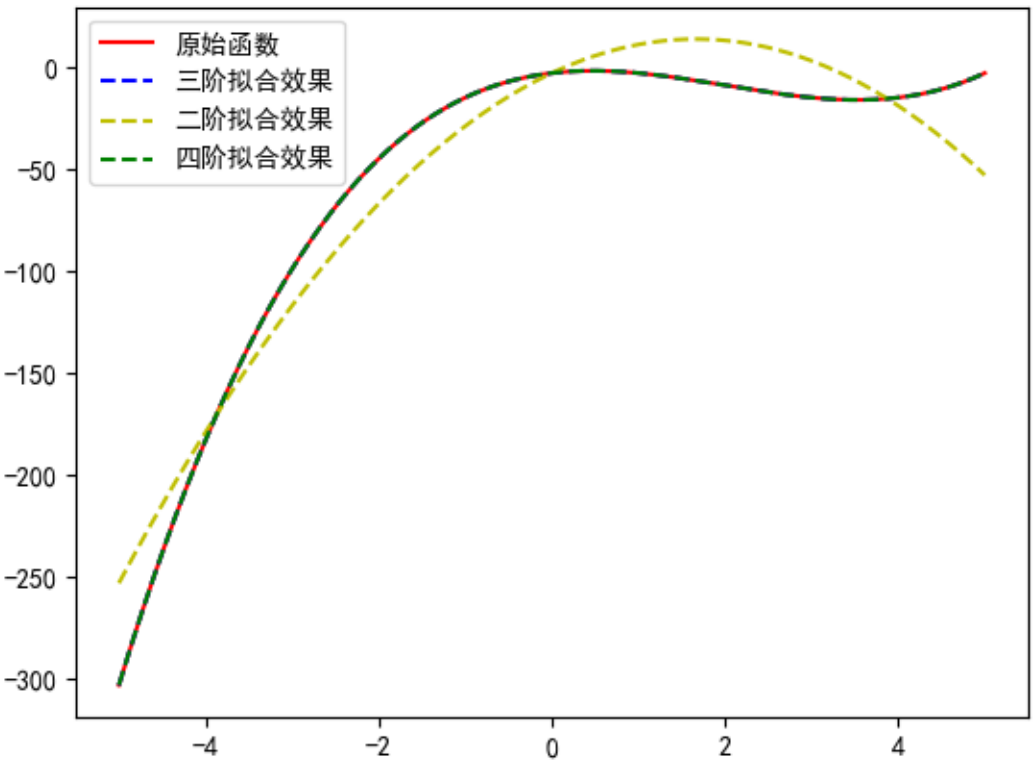

图 1: 练习 1 各阶拟合效果与原始函数对比图

1.2 结果

原始系数: [1, -6 ,5,-3] 三阶拟合系数: [1.00037111,-5.99896795,5.00000599,-3.01698795] 二阶拟合系数:[-5.99896795,20.00857379,-3.01698795] 四阶拟合系数:[2.26810960e-04,1.00037111e+00,-6.00382916e+00,5.00000599e+00 ,-3.00483251e+00]

画图表示如图 1

1.3 分析

通过图形可以看到，三阶、四阶的拟合效果比较好，而二阶的拟合效果一般，与原始函数区别较大。

2 练习 2

2.1 代码

首先定义需要拟合函数的具体形式。然后定义残差函数。在 python 中不需要平方，因为在后续进行拟合的程序中会自动平方进行处理。最后给定一组样本值，并初始化参数，然后送入最小二乘法拟合函数 leastsq 进行求解参数。

练习 2

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import leastsq
# 常数
V = 10

def Fun(params, input):
    V_0, tau = params
    t = input
    return (V - (V - V_0) * np.exp(- t / tau))

def error(params, input, true_val):
    # 这里只写偏差函数，具体平方是在leastsq中计算的
    # 仍然是符合最小二乘法要求
    pred_val = Fun(params, input)
    return pred_val-true_val

def main():
    # 样本数据
    t = np.array([0.5, 1, 2, 3, 4, 5, 7, 9])
    v = np.array([6.36, 6.48, 7.26, 8.22, 8.66, 8.99, 9.43, 9.63])
    # 给定初始的一组参数
    params0 = np.array([10, 1])
    s = Fun(params0, t)
    # 拟合，返回params
    # 传入leastsq参数为：1.残差函数，2.所需确定的函数参数对应的初始值，3.真
        实的输入数据
    params_fit = leastsq(error, params0, args=(t, v))
    # 拟合后的参数数组
    params_fit = params_fit[0]
    V_0, tau = params_fit
    print("V_0: {}, Tau: {}".format(V_0, tau))
    v_pred = Fun(params_fit, t)
    plt.plot(t, v, 'b', label='Sampled True Data')
    plt.plot(t, v_pred, 'o--', label='Fitted Data')
    plt.legend()
```

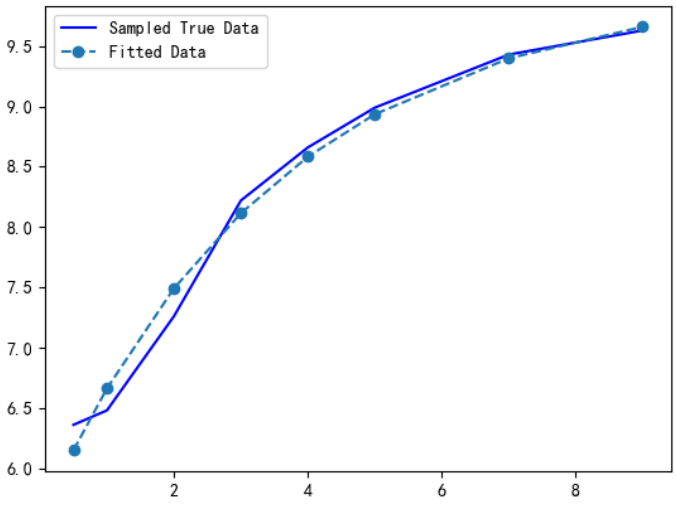

图 2: 练习 2 拟合结果和给定的样本对比图

```
    plt.show()

if __name__ == '__main__':
    main()
```

2.2 结果

拟合后结果为：V_0: 5.557665985114968, Tau: 3.500194038194967。

对比拟合结果和给定的样本如下图 2，其中虚线为拟合结果，深蓝色实线为原始数据的折线图：

3 练习 3

3.1 分析

根据问题，可以列表，得到如下图 3的带多个约束条件的最优化问题，且是一个非线性规划问题：

3.2 求解

3.2.1 题给参数求解

如代码 1 所示，利用题给的 a=50、b=0.2、c=4 得到如下的解。

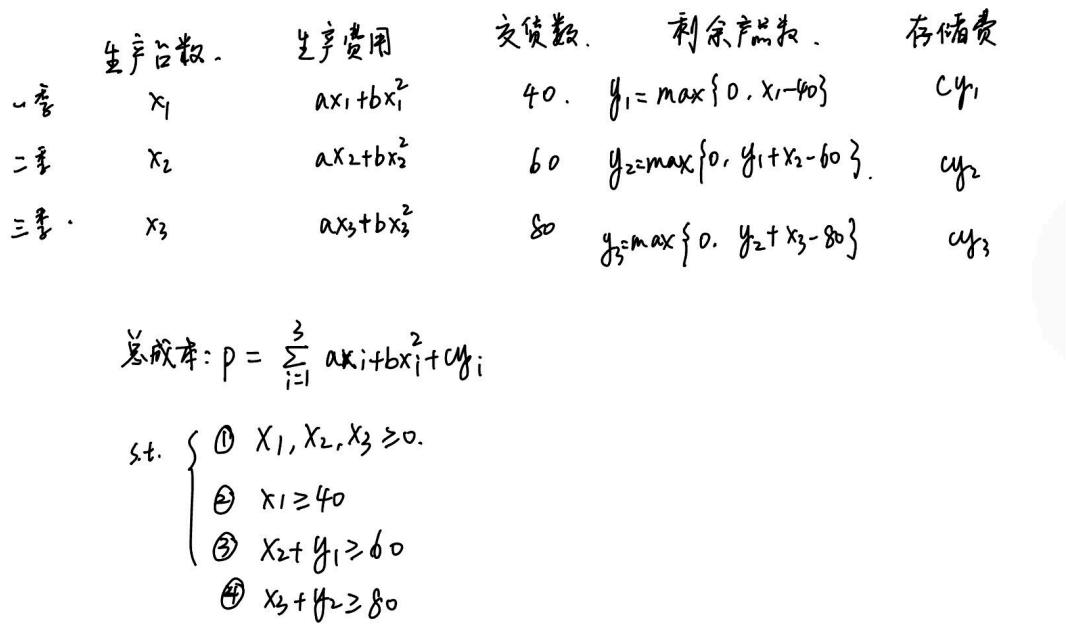

图 3: 练习 3 问题分析和最优化问题的列出

练习 3 代码 1

```python
from scipy.optimize import minimize
import numpy as np

def get_ys(x1, x2, x3):
    y1 = max(0, x1 - 40)
    y2 = max(0, y1 + x2 - 60)
    y3 = max(0, y2 + x3 - 80)
    return y1, y2, y3

def obj_fun(params):
    # x是一个3*2 np.array[[x_1, y_1], [x_2, y_2], [x_3, y_3]]
    # params: a, b, c
    a, b, c = params
    fx = lambda x: a * sum(x[:]) + b * sum(x[:] ** 2) + c * sum(get_ys(x[0],
        x[1], x[2]))
    return fx

def get_constraints(borders):
    # 传入的borders表示等式或者不等式的最小值/最大值
    con1 = {'type': 'ineq', 'fun': lambda x: x[0] - borders[0]}
    con2 = {'type': 'ineq', 'fun': lambda x: x[1] + (get_ys(x[0], x[1], x
       [2]))[0] - borders[1]}
    con3 = {'type': 'ineq', 'fun': lambda x: x[2] + (get_ys(x[0], x[1], x
```

```
        [2]))[1] - borders[2]}
    cons = (con1, con2, con3)
    return cons

def main():
    # 搜索的初始值x1 x2 x3
    x = np.zeros(3)
    # 搜索边界
    bounds = [(0, None) for i in range(3)]
    # 不等式右端
    borders = (40, 60, 80)
    # 约束
    cons = get_constraints(borders)
    # a, b, c
    params = (50, 0.2, 4)
    res = minimize(obj_fun(params), x, method='SLSQP', bounds=bounds,
        constraints=cons)
    if res.success:
        print("Total Cost: {0:.2f}".format(res.fun))
        print("x1: {0:>4.2f}, x2: {1:>4.2f}, x3: {2:>4.2f}".format(res.x[0],
            res.x[1], res.x[2]))
    else:
        print("Sorry, there's no solution!")

if __name__ == '__main__':
    main()
```

结果为:

三个季度的总费用: 11280.00 元;

第一个季度生产 50 台, 第二个季度生产 60 台, 第三个季度生产 70 台发动机。

3.2.2 修改参数后的结果

情况 1: 若 (a, b, c) = (50, 0.2, 1000), 结果为:

三个季度的总费用: 11320.00 元;

第一个季度生产 40 台, 第二个季度生产 60 台, 第三个季度生产 80 台发动机。

可以看到, 如果存储单位成本 c 比较大的时候, 企业会尽可能只满足当前季度订单要求, 而不去多生产设备来存放, 来减少费用支出。

情况 2: 若 (a, b, c) = (50, 10, 4), 结果为:

三个季度的总费用: 117159.20 元;

第一个季度生产 60 台, 第二个季度生产 60 台, 第三个季度生产 60 台发动机。

可以看到, 在平方次项生产成本本身比较高的情况下, 企业是愿意按照季度均匀生产的 (均值不等式原理), 而忽略掉可能产生的存储费用。

情况 3：若 (a, b, c) = (1000, 0.2, 4)，结果为：

三个季度的总费用: 182320.00 元;

第一个季度生产 40 台, 第二个季度生产 60 台, 第三个季度生产 80 台发动机。

可以看到，在一次项生产成本比较高的情况下，平方项成本的影响相对变小，那么会想方设法减小存储成本，因此按照当季度的订单要求生产而不生产多余发动机。

4 练习 4

4.1 代码

练习 4

```python
import numpy as np
import matplotlib.pyplot as plt

"""
    使用k-means动态聚类算法对数据进行分析
"""
# 数据点个数
N = 8
# 类别数
CN = 3

def get_labels(A, C):
    def Euclid_dis(a, c):
        return np.sqrt(sum((a - c) ** 2))
    labels = np.zeros(N, dtype=int)
    for i in range(N):
        labels[i] = np.argmin([Euclid_dis(A[i], c) for c in C])
    return labels

def get_clusters(labels):
    C1_data = np.argwhere(labels == 0).reshape(-1)    # 由于返回的时候是按照多维思路来返回的所以是一个二维数组，而二维下标才是想要的值
    C2_data = np.argwhere(labels == 1).reshape(-1)
    C3_data = np.argwhere(labels == 2).reshape(-1)
    return C1_data, C2_data, C3_data


def get_Centers(A, labels):
    C1_data, C2_data, C3_data = get_clusters(labels)
    C1 = np.mean([A[i] for i in C1_data], axis=0)
    C2 = np.mean([A[i] for i in C2_data], axis=0)
    C3 = np.mean([A[i] for i in C3_data], axis=0)
```

```
    C = np.array([C1, C2, C3])
    return C

def main():
    # 被聚类的几个点
    A = np.array([[2, 10], [2, 5], [8, 4], [5,8],
         [7, 5], [6, 4], [1, 2], [4, 9]])
    # 初始化3个聚类中心
    C1, C2, C3 = A[0], A[3], A[6]
    C = np.array([C1, C2, C3])
    loop_time = 0
    done = False
    while not done:
        labels = get_labels(A, C)
        New_C = get_Centers(A, labels)
        if loop_time == 0:
            print("第一轮后: \n第一个聚类中心({},{})\t第二个聚类中心({},{})\
                t第三个聚类中心({}{})".format(
                New_C[0][0], New_C[0][1], New_C[1][0], New_C[1][1], New_C
                    [2][0], New_C[2][1]
            ))
        if New_C.all() == C.all():
            done = True
        C == New_C
        loop_time += 1
    C1_data, C2_data, C3_data = get_clusters(labels)
    print("最后的三个簇:")
    print("第一类:",end=' ')
    for i in C1_data:
        print('A{}'.format(i),'({},{})'.format(A[i][0], A[i][1]),end=', ')
    print()
    print("第二类:",end=' ')
    for i in C2_data:
        print('A{}'.format(i),'({},{})'.format(A[i][0], A[i][1]),end=', ')
    print()
    print("第三类:",end=' ')
    for i in C3_data:
        print('A{}'.format(i),'({},{})'.format(A[i][0], A[i][1]),end=', ')

    # numpy花式索引，用逗号隔开
    plt.scatter(A[C1_data, 0], A[C1_data, 1],s = 300, c='b', alpha=0.5,
        label='第一类')
    plt.scatter(A[C2_data, 0], A[C2_data, 1],s = 100, c='r', alpha=0.5,
        label='第二类')
```

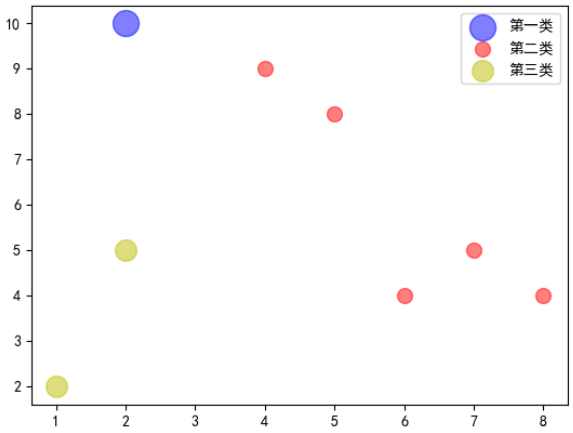

图 4: 练习 4 聚类可视化结果

```
72      plt.scatter(A[C3_data, 0], A[C3_data, 1],s = 200, c='y', alpha=0.5,
            label='第三类')
73      plt.legend()
74      plt.show()
75
76  if __name__ == '__main__':
77      main()
```

4.2 结果

第一轮后:

第一个聚类中心 (2.0,10.0)，第二个聚类中心 (6.0,6.0)，第三个聚类中心 (1.53.5)

最后的三个簇:

第一类: A0 (2,10)

第二类: A2 (8,4), A3 (5,8), A4 (7,5), A5 (6,4), A7 (4,9)

第三类: A1 (2,5), A6 (1,2)

作图如图 4所示:

5 练习 5

5.1 代码

使用了 prim 算法生成最小生成树:

练习 5

```
1  import numpy as np
```

```
NONE = 1000

def prim(V, E, sel, wait, T):
    min_ = []
    for i in sel:
        possible_j = [j for j in range(6) if j not in sel]
        possible_edges = [E[i][j] for j in possible_j]
        edges_i = np.array([[j, edge] for j, edge in zip(possible_j,
            possible_edges)])
        min_j_index = np.argmin(edges_i[:, 1])
        min_j = edges_i[min_j_index][0]
        min_.append([min_j, E[i][min_j], i])
    min_ = np.array(min_)
    min_from = min_[np.argmin(min_[:,1])][2]
    new_sel = min_[np.argmin(min_[:,1])][0]
    edge_sel = E[min_from][new_sel]
    T.append((min_from, new_sel, edge_sel))

    sel.append(new_sel)
    wait.remove(new_sel)
    v_from = V[min_from]
    v_to = V[new_sel]
    print(v_from, '---{}---'.format(edge_sel), v_to)


def main():
    V = ['L', 'M', 'N', 'Pa', 'Pe', 'T']
    E = np.array([
        [NONE, 56, 35, 21, 51, 60],
        [56, NONE, 21, 57, 78, 70],
        [35, 21, NONE, 36, 68, 68],
        [21, 57, 36, NONE, 51, 61],
        [51, 78, 68, 51, NONE, 13],
        [60, 70, 68, 61 ,13, NONE]
    ])
    sel = [0]
    wait = [i for i in range(6) if i not in sel]
    T = []
    while wait:
        prim(V, E, sel, wait, T)
    print(T)
    print('Sum Weights:{}'.format(sum(np.array(T)[:, 2])))

if __name__ == '__main__':
    main()
```

5.2 结果

得到的最小生成树为:

L —21— Pa

L —35— N

N —21— M

L —51— Pe

Pe —13— T

总权值: 141