**МИНОБРНАУКИ РОССИИ**

федеральное государственное бюджетное образовательное учреждение высшего образования

**«Национальный исследовательский университет «МЭИ»**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ЛАБОРАТОРНАЯ РАБОТА №3

По курсу: «Методы решения задач оптимизации»

Тема: «Динамическое программирование»

|  |  |
| --- | --- |
| Выполнил: | Волков М.Л. |
| Вариант: | 1 |
| Группа: | Э-13м-23 |
| Проверил: | Нухулов С.М. |

Москва, 2024 г.

**Предварительный отчет**

**Цель:** получение практических навыков работы с методом решения задач динамического программирования с количеством неизвестных больше трех.

**Задание:**

1. Написать алгоритм метода динамического программирования на языке Python для поставленной задачи;
2. Решить поставленную задачу с использованием написанного алгоритма;

**Формулировка задачи:**

Персональный энергоблок (ПЭБ) имеет в своем составе аккумуляторную батарею емкости capacity [Вт∙ч] и уровнем заряда initCharge Вт∙ч. Цена за электроэнергию в течении дня изменяется согласно почасовому графику priceSchedule, нагрузка потребителя, подключенного к ПЭБ, изменяется согласно почасовому графику loadSchedule, также подключен потребитель с постоянной нагрузкой constantLoad [Вт∙ч]. ПЭБ способен каждый час либо заряжать свою аккумуляторную батарею (от 1 до 4 кВт∙ч), покупая электроэнергию из сети, либо разряжать (от 1 до 4 [кВт∙ч]) – продавая излишки электроэнергии в сеть, либо не производить торговых операций вовсе. Необходимо спланировать график торговых операций на следующий день, имея перечисленную информацию, на каждый час так, чтобы суммарное вознаграждение к концу дня было максимальным, а оставшийся заряд аккумуляторной батареи был выше значения targetCharge [Вт∙ч].

**Исходные данные:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Вар** | **capacity** | **initCharge** | **priceSchedule** | **loadSchedule** | **constantLoad** | **targetCharge** |
| 1 | 16000 | 6000 | 1.5, 1.5, 1.5, 1.5,  1.5, 1.5, 2, 3, 5,  5, 5, 4.5, 3, 3, 3,  3, 4.5, 5, 7, 9,  11, 12, 8, 4 | 480, 320, 320, 360, 360,  360, 420, 920, 1200, 720,  680, 720, 800, 820, 960,  1200, 1380, 1380, 1520,  1800, 1920, 1920, 1640, 1020 | 400 | 4800 |

**Отчет**

**Написание алгоритма динамического программирования:**

Ссылка на репозиторий: <https://github.com/Aglomiras/LR3_Optimize/blob/master/part2.py>

Код:

Без плановых торгов:

import matplotlib.pyplot as plt  
  
  
capacity = 16000 *# емкость персонального энергоблока (ПЭБ)*initCharge = 6000 *# уровень заряда ПЭБ*'''Почасовая цена за электроэнергию'''  
priceSchedule = [1.5, 1.5, 1.5, 1.5, 1.5, 1.5,  
 2.0, 3.0, 5.0, 5.0, 5.0, 4.5,  
 3.0, 3.0, 3.0, 3.0, 4.5, 5.0,  
 7.0, 9.0, 11.0, 12.0, 8.0, 4.0]  
  
'''Почасовое потребление электроэнергии'''  
loadSchedule = [480, 320, 320, 360, 360, 360,  
 420, 920, 1200, 720, 680, 720,  
 800, 820, 960, 1200, 1380, 1380,  
 1520, 1800, 1920, 1920, 1640, 1020]  
  
constantLoad = 400 *# потребитель с постоянной нагрузкой*targetCharge = 4800 *# конечный заряд аккумулятора*'''Продажа/покупка электроэнергии'''  
maxEnergy = 4000  
minEnergy = 1000  
  
'''Инициализация массивов хранящих уровень электроэнергии батареи и затраты на покупку энергии'''  
resLevelEnergy = [0] \* len(loadSchedule)  
resExpenses = [0] \* len(loadSchedule)  
  
'''Словарь промежуточных данных'''  
data = {  
 *# 'levelEnergy': resLevelEnergy,  
 # 'Expenses': resExpenses*}  
  
  
'''Расчет уровня заряда ПЭБ на данный час и затрат на покупку энергии'''  
def calculate\_simple(cons\_power, index, resLevelEnergy\_mass, resExpenses\_mass):  
 global initCharge  
 global targetCharge  
  
 res\_energy = 0  
 if targetCharge > (initCharge - cons\_power - constantLoad):  
 *# Покупка электроэнергии* if cons\_power % minEnergy != 0:  
 res\_energy = (cons\_power / minEnergy) \* 1000 + 1000  
 else:  
 res\_energy = (cons\_power / minEnergy) \* 1000  
  
 initCharge = initCharge + res\_energy - cons\_power - constantLoad  
 else:  
 initCharge = initCharge - cons\_power - constantLoad  
 *# Запись результатов шага рачсета* resLevelEnergy\_mass[index] = initCharge  
 resExpenses\_mass[index] = -res\_energy \* priceSchedule[index]  
  
  
'''Визуализация расчетов'''  
def visual\_calculate\_simple(time\_list, energy\_list, gold\_list):  
 fig, ax = plt.subplots(nrows=1, ncols=2)  
  
 fig.set\_figheight(10)  
 fig.set\_figwidth(20)  
  
 ax[0].plot(time\_list, energy\_list, c='orange')  
 ax[1].plot(time\_list, gold\_list, c='red')  
  
 ax[0].axis(xmin=0, xmax=25, ymin=0, ymax=16000)  
 ax[1].axis(xmin=0, xmax=25, ymin=0, ymax=-50000)  
  
 ax[0].set\_title('levelEnergy ПЭБ')  
 ax[1].set\_title('Expenses')  
 ax[0].set\_xlabel('time, ч')  
 ax[1].set\_xlabel('time, ч')  
 ax[0].set\_ylabel('energy, кВТ\*ч')  
 ax[1].set\_ylabel('gold, руб')  
  
 fig.suptitle('Результаты дня без плановых торгов')  
 plt.show()  
  
  
'''Опыт без плановых торгов'''  
def res\_calculate\_simple(time\_list):  
 *# Предварительная очистка массивов* data.clear()  
 data["levelEnergy"] = resLevelEnergy  
 data["Expenses"] = resExpenses  
  
 *# Расчет энергопотребления за сутки* for i in range(len(loadSchedule)):  
 calculate\_simple(loadSchedule[i], i, data["levelEnergy"], data["Expenses"])  
  
 summExpenses = 0  
 for i in range(len(data["Expenses"])):  
 summExpenses += data["Expenses"][i]  
 print("Суммарные затраты на покупку электроэнергии: ", summExpenses)  
  
 visual\_calculate\_simple(time\_list, data["levelEnergy"], data["Expenses"])  
  
  
'''Временная ось'''  
time = []  
for i in range(len(loadSchedule)):  
 time.append(i + 1)  
  
  
def visual\_consumer(time\_mass, power\_mass):  
 plt.figure()  
 plt.title('График потребления нагрузки')  
 plt.grid()  
 plt.plot(time\_mass, power\_mass, 'tab:red')  
 plt.show()  
  
 return 0  
  
  
*# visual\_consumer(time, loadSchedule)*res\_calculate\_simple(time)

С плановыми торгами:

import matplotlib.pyplot as plt  
  
# ----------------------------------------------------------------------------------------------------------------------  
# ----------------------------------------Блок инициализации исходных данных--------------------------------------------  
# ----------------------------------------------------------------------------------------------------------------------  
capacity = 16000 # емкость персонального энергоблока (ПЭБ)  
initCharge = 6000 # уровень заряда ПЭБ  
  
'''Почасовая цена за электроэнергию'''  
priceSchedule = [1.5, 1.5, 1.5, 1.5, 1.5, 1.5,  
 2.0, 3.0, 5.0, 5.0, 5.0, 4.5,  
 3.0, 3.0, 3.0, 3.0, 4.5, 5.0,  
 7.0, 9.0, 11.0, 12.0, 8.0, 4.0]  
  
'''Почасовое потребление электроэнергии'''  
loadSchedule = [480, 320, 320, 360, 360, 360,  
 420, 920, 1200, 720, 680, 720,  
 800, 820, 960, 1200, 1380, 1380,  
 1520, 1800, 1920, 1920, 1640, 1020]  
  
constantLoad = 400 # потребитель с постоянной нагрузкой  
targetCharge = 4800 # конечный заряд аккумулятора  
  
'''Продажа/покупка электроэнергии'''  
maxEnergy = 4000  
minEnergy = 1000  
  
'''Инициализация массивов хранящих уровень электроэнергии батареи и затраты на покупку энергии'''  
resLevelEnergy = [0] \* len(loadSchedule)  
resExpenses = [0] \* len(loadSchedule)  
  
'''Временная ось'''  
time = [0]  
for i in range(len(loadSchedule)):  
 time.append(i + 1)  
  
  
# ----------------------------------------------------------------------------------------------------------------------  
# -------------------------------Класс таблицы задачи динамического программирования------------------------------------  
# ----------------------------------------------------------------------------------------------------------------------  
class Data\_Dynamic\_Program:  
 # ------------------------------------------------------------------------------------------------------------------  
 # Конструктор класса  
 # ------------------------------------------------------------------------------------------------------------------  
 def \_\_init\_\_(self, num\_hour, const\_charge, load\_charge, load\_price, last\_income, last\_level\_charge,  
 trading\_operation):  
 # --------------------------------------------------------------------------------------------------------------  
 # Данные текущего часа  
 # --------------------------------------------------------------------------------------------------------------  
 self.hour = num\_hour # Номер текущего часа  
 self.const\_charge = const\_charge # Величина постоянно потребляемой энергии  
 self.load\_charge = load\_charge # Нагрузка потребителей на текущий час  
 self.load\_price = load\_price # Тариф на электроэнергию на данный час  
 # --------------------------------------------------------------------------------------------------------------  
 # Данные предыдущего часа  
 # --------------------------------------------------------------------------------------------------------------  
 self.last\_income = last\_income # Доход с предыдущей итерации [Массив]  
 self.last\_level\_charge = last\_level\_charge # Оставшийся заряд ПЭБ с предыдущей итерации [Массив]  
 self.trading\_operation = trading\_operation # Торговая операция (покупка/продажа) с предыдущей итерации [Массив]  
 # --------------------------------------------------------------------------------------------------------------  
 # Расчет конечных состояний часа  
 # --------------------------------------------------------------------------------------------------------------  
 self.now\_last\_income = list() # Возможные варианты дохода при торгах на текущем часе  
 self.now\_last\_level\_charge = list() # Варианты оставшегося заряда ПЭБ после окончания часа  
 self.now\_trading\_operation = list() # Варианты торговых операций на текущий час  
  
 # ------------------------------------------------------------------------------------------------------------------  
 # Метод создания начальной таблицы задачи динамического программирования  
 # ------------------------------------------------------------------------------------------------------------------  
 def table\_Data\_Initial(self):  
 # --------------------------------------------------------------------------------------------------------------  
 # Расчет всех возможных торговых операций (от -4000 до 4000) на первом шаге  
 # [+] - покупка энергии  
 # [-] - продажа энергии  
 # --------------------------------------------------------------------------------------------------------------  
 for i in range(-4, 5):  
 operation = i \* 1000  
 end\_energy = self.last\_level\_charge[self.hour] + operation - self.const\_charge - self.load\_charge  
  
 if 0 <= end\_energy <= capacity:  
 self.now\_last\_income.append(- operation \* self.load\_price)  
 self.now\_last\_level\_charge.append(end\_energy)  
 self.now\_trading\_operation.append(operation)  
  
 # ------------------------------------------------------------------------------------------------------------------  
 # Метод дополнения таблицы задачи динамического программирования  
 # ------------------------------------------------------------------------------------------------------------------  
 def table\_Data\_Update(self):  
 # --------------------------------------------------------------------------------------------------------------  
 # Создание хэш-таблицы (словаря) для хранения промежуточных данных  
 # Условная структура:  
 # data[end\_energy] = {  
 # "operation": None,  
 # "income": None,  
 # "end\_energy": None  
 # }  
 # end\_energy - конечный уровень заряда по окончанию часа, исполняет роль ключа, так как по конечному уровню  
 # заряда легко отслеживать все варианты возможных событий  
 # --------------------------------------------------------------------------------------------------------------  
 data = dict()  
  
 # --------------------------------------------------------------------------------------------------------------  
 # Расчет всех возможных торговых операций (от -4000 до 4000) на первом шаге  
 # [+] - покупка энергии  
 # [-] - продажа энергии  
 # --------------------------------------------------------------------------------------------------------------  
 for i in range(-4, 5):  
 operation = i \* 1000 # Торговая операция с энергией  
 income = - operation \* self.load\_price # Стоимость энергии при данной торговой операции  
  
 # ----------------------------------------------------------------------------------------------------------  
 # Определение максимально прибыльного варианта в данный расчетный час  
 # ----------------------------------------------------------------------------------------------------------  
 for j in range(len(self.last\_level\_charge)):  
 # ------------------------------------------------------------------------------------------------------  
 # end\_energy - уровень энергии ПЭБ  
 # end\_income - общий доход, полученный после i операции, при j варианте прошлого этапа расчета  
 # data\_set - хранит промежуточные значения  
 # ------------------------------------------------------------------------------------------------------  
 end\_energy = self.last\_level\_charge[j] + operation - self.const\_charge - self.load\_charge  
 end\_income = self.last\_income[j] + income  
  
 data\_set = [operation, end\_income,  
 end\_energy] # Хранит торговую операцию, ее доход, ее конечную энергию  
  
 # ------------------------------------------------------------------------------------------------------  
 # Проверка на соответствие заданным условиям диапазона мощности  
 # ------------------------------------------------------------------------------------------------------  
 if 0 <= end\_energy <= capacity:  
 # --------------------------------------------------------------------------------------------------  
 # Проверка, что данный уровень заряда после i операции, при j исходе прошлого расчета уже имеется в  
 # словаре. Если такого конечного уровня заряда ранее не было, то записываем его в словарь. Если  
 # такой уровень заряда уже имеется, то записываем максимально выгодный из двух вариантов  
 # --------------------------------------------------------------------------------------------------  
 if end\_energy in data:  
 if data[end\_energy][1] < end\_income:  
 data[end\_energy] = data\_set  
 else:  
 data[end\_energy] = data\_set  
  
 # --------------------------------------------------------------------------------------------------------------  
 # Запись оптимальных вариантов расчета данного часа для каждого варианта операции  
 # Формируем листы вариантов дохода, конечного уровня заряда, торговых операций для следующего часа расчета  
 # --------------------------------------------------------------------------------------------------------------  
 for i in data.keys():  
 self.now\_last\_income.append(data[i][1])  
 self.now\_last\_level\_charge.append(data[i][2])  
 self.now\_trading\_operation.append(data[i][0])  
  
 # ------------------------------------------------------------------------------------------------------------------  
 # Метод вывода таблицы данного часа  
 # ------------------------------------------------------------------------------------------------------------------  
 def to\_string\_table(self):  
 print("Текущий час: ", self.hour + 1, "\n",  
 "Нагрузка текущего часа: ", self.load\_charge, "\n",  
 "Постоянное потребление: ", self.const\_charge, "\n",  
 "Тариф на электроэнергию в текущий час: ", self.load\_price, "\n",  
 "Возможные варианты торговых операций: ", self.now\_trading\_operation, "\n",  
 "Возможны варианты заряда ПЭБ на конец часа: ", self.now\_last\_level\_charge, "\n",  
 "Возможный доход: ", self.now\_last\_income, "\n"  
 )  
  
  
# ----------------------------------------------------------------------------------------------------------------------  
# --------------------------------Блок расчета задачи динамического программирования------------------------------------  
# ----------------------------------------------------------------------------------------------------------------------  
def calculation(const\_charge, load\_charge, load\_price, initChar, targetChar):  
 # ------------------------------------------------------------------------------------------------------------------  
 # Создадим лист, который будет содержать в себя все таблицы (по каждому часу: индексу). Каждая из этих таблиц будет  
 # представлять из себя экземпляр класса Data\_Dynamic\_Program со всеми полями  
 # ------------------------------------------------------------------------------------------------------------------  
 payslips = list()  
  
 # ------------------------------------------------------------------------------------------------------------------  
 # Записываем в лист таблицы расчета по каждому часу  
 # ------------------------------------------------------------------------------------------------------------------  
 for i in range(0, len(loadSchedule), 1):  
 if i == 0:  
 data\_table = Data\_Dynamic\_Program(i, const\_charge, load\_charge[i], load\_price[i], [0], [initChar], [0])  
 data\_table.table\_Data\_Initial()  
 else:  
 data\_table = Data\_Dynamic\_Program(i, const\_charge, load\_charge[i], load\_price[i],  
 payslips[i - 1].now\_last\_income,  
 payslips[i - 1].now\_last\_level\_charge,  
 payslips[i - 1].now\_trading\_operation)  
 data\_table.table\_Data\_Update()  
  
 # --------------------------------------------------------------------------------------------------------------  
 # Вывод таблицы за данный час в консоль  
 # --------------------------------------------------------------------------------------------------------------  
 data\_table.to\_string\_table()  
 payslips.append(data\_table)  
  
 # ------------------------------------------------------------------------------------------------------------------  
 # Поиск наилучшего варианта  
 # Создаем вспомогательные массивы для хранения значений дохода, энергии в конце часа и торговой операции  
 # Эти массивы будут использованы для построения графиков дохода и нагрузки  
 # ------------------------------------------------------------------------------------------------------------------  
 inc\_mass = []  
 energy\_mass = []  
 operation\_mass = []  
 for i in range(len(loadSchedule) - 1, -1, -1):  
 if i == len(loadSchedule) - 1:  
 # ----------------------------------------------------------------------------------------------------------  
 # Сортировка листа вариантов конечного заряда ПЭБ:  
 # Максимальная прибыль будет достигнута в том случае, когда в последний час будет продана максимально  
 # возможная мощность и оставшийся заряд ПЭБ будет максимально близким к предельному значению ПЭБ  
 # ----------------------------------------------------------------------------------------------------------  
 help\_end\_energy = [] # Вспомогательный массив  
 for j in range(len(payslips[i].now\_last\_level\_charge)):  
 val\_energy = payslips[i].now\_last\_level\_charge[j]  
 # ------------------------------------------------------------------------------------------------------  
 # Отбираем только те варианты конечной мощности, которые больше предельного значения  
 # ------------------------------------------------------------------------------------------------------  
 if targetChar < val\_energy:  
 help\_end\_energy.append(val\_energy)  
  
 # ----------------------------------------------------------------------------------------------------------  
 # Находим минимальное значения оставшейся энергии из всех возможных вариантов конечного состояния ПЭБ,  
 # после находим соответствующее ему значение дохода от торговых операций  
 # ----------------------------------------------------------------------------------------------------------  
 min\_satisfy = min(help\_end\_energy) # Минимальное значение оставшегося заряда  
 index\_max\_income = payslips[i].now\_last\_level\_charge.index(min\_satisfy) # Индекс этого заряда в массиве  
  
 # ----------------------------------------------------------------------------------------------------------  
 # Запись значений в массивы  
 # Первые значения элементов в массивах соответствуют самым поздним часам дня (инверсированные массивы)  
 # ----------------------------------------------------------------------------------------------------------  
 inc\_mass.append(payslips[i].now\_last\_income[index\_max\_income])  
 energy\_mass.append(min\_satisfy)  
 operation\_mass.append(payslips[i].now\_trading\_operation[index\_max\_income])  
 else:  
 # ----------------------------------------------------------------------------------------------------------  
 # Выполнение обратной торговой операции для восстановления последовательности энергии ПЭБ в конце каждого  
 # часа, а также нахождение индекса этого элемента для нахождения соответствующего дохода  
 # ----------------------------------------------------------------------------------------------------------  
 last\_energy = energy\_mass[-1] - operation\_mass[-1] + const\_charge + load\_charge[i + 1]  
 index\_max\_income = payslips[i].now\_last\_level\_charge.index(last\_energy)  
  
 # ----------------------------------------------------------------------------------------------------------  
 # Запись значений в массивы  
 # Первые значения элементов в массивах соответствуют самым поздним часам дня (инверсированные массивы)  
 # ----------------------------------------------------------------------------------------------------------  
 inc\_mass.append(payslips[i].now\_last\_income[index\_max\_income])  
 energy\_mass.append(last\_energy)  
 operation\_mass.append(payslips[i].now\_trading\_operation[index\_max\_income])  
  
 # ------------------------------------------------------------------------------------------------------------------  
 # Добавления начальных значений дохода и энергии  
 # ------------------------------------------------------------------------------------------------------------------  
 inc\_mass.append(0)  
 energy\_mass.append(initChar)  
  
 return inc\_mass[::-1], energy\_mass[::-1]  
  
  
# ----------------------------------------------------------------------------------------------------------------------  
# ------------------------------Блок визуализации задачи динамического программирования---------------------------------  
# ----------------------------------------------------------------------------------------------------------------------  
def visual\_consumer(time\_mass, mass\_val, name\_graf, num\_graf, name\_y, color):  
 plt.subplot(2, 1, num\_graf)  
 plt.grid(True, color="grey", linewidth="1.4", linestyle="-.")  
 plt.title(name\_graf, fontsize=10)  
 plt.ylabel(name\_y, fontsize=14)  
 plt.xlabel('time, ч', fontsize=10)  
 plt.plot(time\_mass, mass\_val, 'r', c=color, linewidth=3, linestyle="-")  
  
  
# ----------------------------------------------------------------------------------------------------------------------  
# Расчет максимального дохода данной задачи  
# ----------------------------------------------------------------------------------------------------------------------  
income, energy = calculation(constantLoad, loadSchedule, priceSchedule, initCharge, targetCharge)  
print("Изменение дохода: ", income)  
print("Изменение заряда ПЭБ: ", energy)  
  
fig, ax = plt.subplots(nrows=1, ncols=2)  
fig.set\_figheight(8) # Высота  
fig.set\_figwidth(16) # Длина  
plt.subplots\_adjust(wspace=10, hspace=0.2, left=0.06, right=0.98, top=0.96, bottom=0.1) # Отступы по краям  
visual\_consumer(time, energy, "levelEnergy ПЭБ", 1, "energy, кВТ\*ч", "red")  
visual\_consumer(time, income, "Expenses", 2, "gold, руб", "blue")  
plt.show()

**Вывод:**

В ходе выполнения лабораторной работы был разработан алгоритм расчета торговых операций для изменения заряда ПЭБ, для получения максимальной денежной выгоды и поддержания заряда батареи на определенном уровне, то есть были получены практические навыки решения задач динамического программирования.