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.