Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э.Баумана)
Мытищинский филиал

ФАКУЛЬТЕТ КОСМИЧЕСКИЙ

КАФЕДРА К-1 САУ

ЛАБОРАТОРНАЯ РАБОТА №10

ПО ДИСЦИПЛИНЕ
“ЧИСЛЕННЫЕ МЕТОДЫ”

НА ТЕМУ:

«Решение задач оптимизации»

Студент К1-61Б	**21.05.24**	**Тимофеев К. А.**
(Группа)	(Подпись, дата)	(ФИО)
Руководитель		**Чернова Т.В.**
	(Подпись, дата)	(ФИО)

2024 г.

Теория

Задачи оптимального планирования, связанные с отысканием оптимума заданной целевой функции (линейной формы) при наличии ограничений в виде линейных уравнений или линейных неравенств относятся к задачам линейного программирования.
Линейное программирование – это направление математического программирования, изучающее методы решения экстремальных задач, которые характеризуются линейной зависимостью между переменными и линейным критерием.

В общем случае задача сводится к поиску экстремумов некой функции при заданных ограничениях.

$$F = {}_{j=0}^{n} c_j \quad x_j \to \min / \max$$

$$\begin{array}{cccccc} a_{ij} & x_j + & \cdots & + a_{in} & x_n & b_i \\ & \vdots & \ddots & \ddots & & \vdots \\ a_{mj} & x_j + & \cdots & a_{mn} & x_n & b_m \end{array}$$

Задачи

Задача 1

Для производства двух видов изделия А и В предприятие использует три вида сырья. Нормы расхода сырья на изготовление единицы продукции данного вида представлены в таблице.

Виды сырья	Нормы расхода сырья на одно изделие, кг		Общее количество сырья, кг
	A	*B*	
I	12	4	300
II	4	4	120
III	3	12	252
Прибыль (одно изделие)	30	40	

Требуется составить план выпуска изделий А и В, при котором прибыль предприятия от их реализации максимальна.

Задача 2

Аналогично задаче 1 опроксимировать функцию прибыли по заданным значениям.

12.	160	100	0,2	0,1	0,2	0,5	0,17	0,16	100	180	90

Задачи решить графически и с использованием внешних систем расчёта.

Ход работы

Реализация графического решения имеет достаточно громоздкий вид, опишем его словесно, чтобы не углубляться в тонкости работы с компьютерной графикой.

Для графического решения указанной задачи оптимизации необходимо построить прямые, уравнения которых находят из системы ограничений, меняя знак больше-равно на равно. Точки пересечения прямых назовём узлами. Затем проводят график функции F(x0, x1,…,xn), который параллельным переносом сдвигают первого пересечения с узлом. Найденный в таком случае узел будет являться искомым экстремумом функции, а его координаты соответственно искомыми значениями параметров функции.

Реализация алгоритма

```
def lineEqCoef(xy : list[list[float]]) -> list[float]:
    xa, ya = xy[0]
    xb, yb = xy[1]
    # Canonical equation
    '''
        (x-xa)/(xb-xa) = (y-ya)/(yb-ya)
        x/(xb-xa) - xa/(xb-xa) = y/(yb-ya) - ya/(yb-ya)
        y = x*(yb-ya)/(xb-xa) + (yb-ya)*(- xa/(xb-xa) + ya/(yb-ya))
        y = x*a + b
    '''
    # then coefficients is
    a = (yb-ya)/(xb-xa)
    b = (yb-ya)*(- xa/(xb-xa) + ya/(yb-ya))
    return [a, b]

def Crossing(ab1, ab2):
    A = np.array([[-ab1[0], 1], [-ab2[0], 1]])
    B = np.array([ab1[1], ab2[1]])
    return np.linalg.solve(A, B)

def searchHighestPoint(points : dict):
    dotX = 0
    dotY = 0
    for i in points.keys():
        if dotY <= points[i][1]:
            dotY = points[i][1]; dotX = points[i][0]
    return [dotX, dotY]
```

```
def lineParallelTransfer(transferLine : list[list[float]], point_where_transfer : list[float]) -> list[float]:
        # Canonical equation
    '''
        dx, dy - translation offset
        ((x+dx)-xa)/(xb-xa) = (y-ya)/(yb-ya)
        (x+dx)/(xb-xa) - xa/(xb-xa) = y/(yb-ya) - ya/(yb-ya)
        y = (x+dx)*(yb-ya)/(xb-xa) + (yb-ya)*(- xa/(xb-xa) + ya/(yb-ya))
        y = x*(yb-ya)/(xb-xa) + dx*(yb-ya)/(xb-xa) + (yb-ya)*(- xa/(xb-xa) + ya/(yb-ya))
    '''
    xa, ya = transferLine[0];    xb, yb = transferLine[1]

    a, b = lineEqCoef(transferLine)

    dx = 0
    dy = 0

    x = np.linspace(0, 100, 1000).round(1)
    y = []
    for i in x:
        y.append(a*i+b)
        if i == point_where_transfer[0]:
            dy = point_where_transfer[1]-y[-1]
        if y[-1] == point_where_transfer[1]:
            dx = point_where_transfer[0]-i

    A = (yb-ya)/(xb-xa)
    B = dx*(yb-ya)/(xb-xa) + (yb-ya)*(- xa/(xb-xa) + ya/(yb-ya))

    return [A, B]

def graphOptimize(df : pd.DataFrame, pltLimits):
    # expect dataframe with n columns of dots
    n = len(df.columns)
    # [[x1, y1], [x2, y2]]
    lines = [df[i] for i in df.columns]
    ab = [lineEqCoef(i) for i in lines]
    a = [i[0] for i in ab]
    b = [i[1] for i in ab]

    fig, (ax) = plt.subplots()

    x = np.linspace(pltLimits[0][0], pltLimits[0][1], 1000).round(4)
    y : list[list[float]] = []

    ax.set_xlim(pltLimits[0])
    ax.set_ylim(pltLimits[1])

    colors = ["#"+''.join([random.choice('0123456789ABCDEF') for j in range(6)]) for i in range(n-1)]

    for j in range(n-1):
        y.append([a[j]*i+b[j] for i in x])
        ax.plot(x, y[j], color=colors[j], linewidth=1, label=str(j+1)+' line')
        ax.stackplot(x, y[j], color=colors[j], alpha = 0.18)

    y.append([a[n-1]*i+b[n-1] for i in x])
    ax.plot(x, y[n-1], '--', linewidth=1.2, color='black', label='Level line')

    cross_points = {}
    k = 1
    for i in range(n-1):
        for j in range(i, n-1):
            if j == i : continue
            cross_points[str(i+1)+' : '+str(j+1)] = Crossing([a[i], b[i]], [a[j], b[j]])
            ax.scatter(cross_points[str(i+1)+' : '+str(j+1)][0], cross_points[str(i+1)+' : '+str(j+1)][1], linewidth=0.7, marker='o', 
color='black')
        k+=1

    highest_cross_point = searchHighestPoint(cross_points)

    newLevelLineCoef = lineParallelTransfer(df['levelline'], highest_cross_point)

    ax.plot(x, [newLevelLineCoef[0]*i + newLevelLineCoef[1] for i in x], '--', linewidth=1.2, color='red', label='New level line')

    plt.grid();   plt.legend(loc='upper right'); plt.show()
```

Компактнее решение можно отыскать используя заранее разработанные системы, например, в библиотеке (модуле) для анализа данных SciPy существует метод optimize.linprog(…), который позволяет упростить взаимодействие со сложными методами оптимизации.

https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.html

```
scipy.optimize.linprog(c, A_ub=None, b_ub=None, A_eq=None, b_eq=None, bounds=(0, None), method='highs', callback=None, options=None, x0=None, integrality=None)
```

Linear programming: minimize a linear objective function subject to linear equality and inequality constraints. [source]

Linear programming solves problems of the following form:

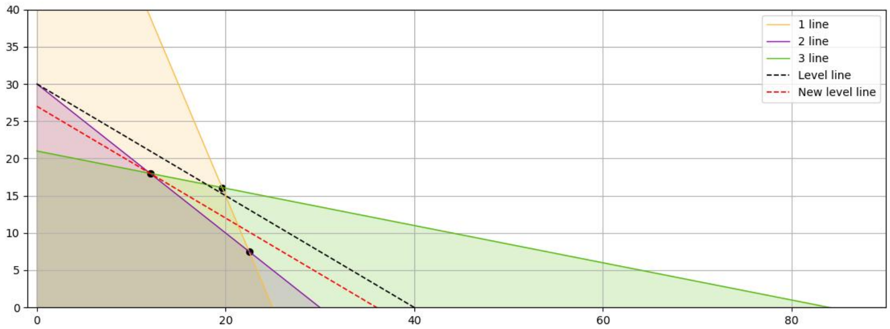
$$
\begin{aligned}
\min_x \ & c^T x \\
\text{such that } & A_{ub} x \leq b_{ub}, \\
& A_{eq} x = b_{eq}, \\
& l \leq x \leq u,
\end{aligned}
$$

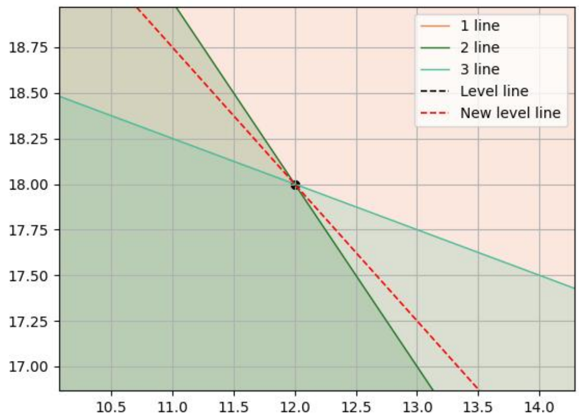
where x is a vector of decision variables; c, b_{ub}, b_{eq}, l, and u are vectors; and A_{ub} and A_{eq} are matrices.

Важно отметить, что данный метод, как и многие другие реализации, выполняют поиск только минимума функции, поэтому данные заранее необходимо подготовить, если требуется найти максимум.

Задача 1

Графическое решение





Вывод linprog():

```
= RESTART: C:\Users\Админ\Documents\Labs\Math\NumericMethods\Lab10\lb10.py
        message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
        success: True
         status: 0
            fun: -1079.9999999999998
              x: [ 1.200e+01  1.800e+01]
            nit: 3
          lower:  residual: [ 1.200e+01  1.800e+01]
                 marginals: [ 0.000e+00  0.000e+00]
          upper:  residual: [       inf        inf]
                 marginals: [ 0.000e+00  0.000e+00]
          eqlin:  residual: []
                 marginals: []
        ineqlin:  residual: [ 8.400e+01  0.000e+00  0.000e+00]
                 marginals: [-0.000e+00 -6.667e+00 -1.111e+00]
 mip_node_count: 0
 mip_dual_bound: 0.0
        mip_gap: 0.0
1079.9999999999998
[12. 18.]
```

Задача 2

Вывод linprog(…)

```
=== RESTART: C:\Users\Админ\Documents\Labs\Math\NumericMethods\Lab10\lb10.py ===
        message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
        success: True
         status: 0
            fun: -81333.33333333333
              x: [ 4.667e+02  6.667e+01]
            nit: 3
          lower:  residual: [ 4.667e+02  6.667e+01]
                 marginals: [ 0.000e+00  0.000e+00]
          upper:  residual: [       inf        inf]
                 marginals: [ 0.000e+00  0.000e+00]
          eqlin:  residual: []
                 marginals: []
        ineqlin:  residual: [ 0.000e+00  5.333e+01  0.000e+00]
                 marginals: [-5.733e+02 -0.000e+00 -2.667e+02]
 mip_node_count: 0
 mip_dual_bound: 0.0
        mip_gap: 0.0
81333.33333333333
[466.66666667  66.66666667]
```