Министерство образования и науки Российской Федерации
Федеральное агентство по образованию
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Отчет по лабораторной работе №1 дисциплины «Исследование операций»

Выполнил студент группы ИВТ-31_____/Крючков И. С/
Проверил_____/Коржавина А. С./

Киров 2022

1. Цель работы

Закрепить на практике знания о симплексном методе решения задач линейного программирования и получить навыки его программной реализации.

2. Задание

Написать программу, реализующую симплексный метод либо одну из его модификаций – метод искусственного базиса, двойственный симплекс и т. д. Необходимо рассмотреть все возможные случаи, например, когда задача не имеет решений, имеет множество решений и т.д.

3. Описание симплекс-метода

Симплекс-метод позволяет эффективно найти оптимальное решение, избегая простой перебор всех возможных угловых точек. Основной принцип метода: вычисления начинаются с какого-то «стартового» базисного решения, а затем ведется поиск решений, «улучшающих» значение целевой функции. Это возможно только в том случае, если возрастание какой-то переменной приведет к увеличению значения функционала.

4. Листинг программы

```
# Ввод данных

v_num = int(input("Введите количество переменных: \n"))
l_num = int(input("Введите количество ограничений: \n"))

sv_num = v_num

f = [0] * v_num
result_x = [0] * v_num

f_raw = input(f"Введите коэффициенты целевой функции F(x) ({v_num} числ. через пробел):
\n").split()

f = list(map(float, f_raw))

ok = 0
while ok == 0:
    f_type = int(input(("Введите тип функции:\n"
                    "1. max\n"
                    "2. min\n")))
    if(f_type == 1 or f_type == 2):
        ok = 1

limits_a = [[0] * v_num] * l_num
limits_b = [0] * l_num
limits_type = [1] * l_num
```

```
for i in range(l_num):
    ok = 0
    while ok == 0:
        limits_raw = input(f"Ввод ограничений №{i+1} ({v_num+1} числ. через пробел):
\n").split()
        if(len(limits_raw) == v_num+1):
            ok = 1

    limits_tmp = list(map(float, limits_raw))
    limits_a[i] = limits_tmp[:-1]
    limits_b[i] = limits_tmp[-1]

    ok = 0
    while ok == 0:
        l_type = int(input(("Введите тип ограничения:\n"
                        "1. >=\n"
                        "2. <=\n"
                        "3. =\n")))
        if(l_type == 1 or l_type == 2 or l_type == 3):
            ok = 1

    limits_type[i] = l_type


def comp(a, b):
    if f_type == 1:
        return a < b
    else:
        return a > b

# Приведение к каноническому виду
# 1 >=
# 2 <=
k_type = 1 if f_type == 2 else 2
un_k_type = f_type

for i, lim in enumerate(limits_a):
    if limits_type[i] == 1:
        limits_a[i] = list(map(lambda x: -x, lim))
        limits_b[i] = -limits_b[i]
        limits_type[i] = k_type


    if limits_type[i] != 3:
        for ai, _ in enumerate(limits_a):
            limits_a[ai].append(0)

        v_num += 1

        limits_a[i][-1] = 1
        f.append(0)
        result_x.append(0)

limits_b.append(0)

min_max = min if f_type == 1 else max

def get_basis(st = 0):

    vbasis = [-1]*l_num
    bs_zero = l_num

    cols_t = list(zip(*limits_a))
    cols = [list(sb) for sb in cols_t]
```

```
z_cols_ids = []
oe_cols_ids = []
nz_cols_ids = []

for cid, col in enumerate(cols):

    z_f = False
    cnt = 0
    el_id = 0

    oe_cnt = 0
    oe_id = 0

    nz_cnt = 0

    for i, v in enumerate(col):

        if v == 1:
            cnt += 1
            el_id = i

            oe_cnt += 1
            oe_id = i
        elif v != 0:
            cnt = 0
            z_f = True

            oe_cnt += 1
            oe_id = i

        if v == 0:
            nz_cnt += 1

    if cnt == 1 and z_f == False:
        z_cols_ids.append((el_id, cid))

    if oe_cnt == 1:
        oe_cols_ids.append((oe_id, cid))

    if nz_cnt == 0:
        nz_cols_ids.append(cid)


for lavid, lav in enumerate(limits_a):
    added = 0

    for idld, ld in reversed(list(enumerate(lav))):
        if ld == 1:
            for zv in z_cols_ids:
                if idld == zv[1]:
                    vbasis[lavid] = idld
                    bs_zero -= 1
                    added = 1
                    break

        if added == 1:
            break

if bs_zero == 0:
    return vbasis

for bsid, bs in enumerate(vbasis):
```

```
        if bs == -1:
            for ldiv, ldata in enumerate(limits_a[bsid]):
                added = 0
                for oe in oe_cols_ids:
                    if bsid == oe[0] and ldiv == oe[1]:
                        vbasis[bsid] = ldiv
                        bs_zero -= 1
                        added = 1

                        limits_b[bsid] /= limits_a[bsid][ldiv]

                        for ldi, ldd in enumerate(limits_a[bsid]):
                            limits_a[bsid][ldi] /= limits_a[bsid][ldiv]

                        break

            if added == 1:
                break

    if bs_zero == 0:
        return vbasis

    for bsid, bs in enumerate(vbasis):
        if bs == -1:
            for nz in nz_cols_ids:
                if nz not in vbasis:
                    dtmp = limits_a[bsid][nz]

                    limits_b[bsid] /= dtmp

                    for ldi, ldd in enumerate(limits_a[bsid]):
                        limits_a[bsid][ldi] /= dtmp


                    for ltid, ltd in enumerate(limits_a):
                        if ltid != bsid:
                            mdt = ltd[nz]
                            for ldi, ldd in enumerate(ltd):
                                limits_a[ltid][ldi] -= limits_a[bsid][ldi] * mdt

                            limits_b[ltid] -= limits_b[bsid] * mdt

                    vbasis[bsid] = nz
                    bs_zero -= 1
                    break

    return vbasis


def get_delta(bs):
    delta = [0]*v_num
    limits_b[-1] = 0

    cols_t = list(zip(*limits_a))
    cols = [list(sb) for sb in cols_t]

    for did, d in enumerate(delta):
        for bid, base in enumerate(bs):
            delta[did] += f[base] * cols[did][bid]
        delta[did] -= f[did]

    for bid, base in enumerate(bs):
        limits_b[-1] += f[base] * limits_b[bid]
    limits_b[-1] -= f[-1]
```

```
    return delta


def rem_nfc(bs):

    def num_negative(l):
        nn = 0
        for lb in l:
            if lb < 0:
                nn += 1
        return nn


    rne = False
    while rne == False and num_negative(limits_b[:-1]) > 0:

        nvs = [(i, v) for i, v in enumerate(limits_b[:-1]) if v < 0]
        mrow = max(nvs,key=lambda x: abs(x[1]))[0]

        if num_negative(limits_a[mrow]) == 0:
            rne = True
            break

        nvs = [(i, v) for i, v in enumerate(limits_a[mrow]) if v < 0]

        mcol = max(nvs,key=lambda x: abs(x[1]))[0]

        dtmp = limits_a[mrow][mcol]

        limits_b[mrow] /= dtmp

        for ldi, ldd in enumerate(limits_a[mrow]):
            limits_a[mrow][ldi] /= dtmp


        for ltid, ltd in enumerate(limits_a):
            if ltid != mrow:
                mdt = ltd[mcol]
                for ldi, ldd in enumerate(ltd):
                    limits_a[ltid][ldi] -= limits_a[mrow][ldi] * mdt

                limits_b[ltid] -= limits_b[mrow] * mdt

        bs[mrow] = mcol

    return rne, bs


# основной цикл вычислений

fst = 1

while True:

    basis = get_basis(fst)
    fst = 0

    err, basis = rem_nfc(basis)

    if err:
        print("Нет решений")
        exit(1)
```

```
    dlt = get_delta(basis)

    norm = 0
    for v in dlt:
        if comp(v, 0):
            norm += 1

    if norm == 0:
        break

    #разрешающий столбец
    rc_id = min_max(range(len(dlt)), key=dlt.__getitem__)
    rc = []
    for i in range(l_num):
        rc.append(limits_a[i][rc_id])

    bi_d_rc = []
    for di, (bi, rci) in enumerate(zip(limits_b[:-1], rc)):
        if rci != 0:
             if bi/rci >= 0:
                bi_d_rc.append((di, bi/rci))

    if len(bi_d_rc) == 0:
        print("Оптимальное решение отсутствует")
        exit(1)

    row, _ = min(bi_d_rc, key=lambda x: x[1])

    limits_b[row] /= limits_a[row][rc_id]
    limits_a[row] = list(map(lambda x: x / limits_a[row][rc_id], limits_a[row]))


    av = limits_a[row][rc_id]

    for i, lim in enumerate(limits_a):
        if i != row:

            b_bv = limits_a[i][rc_id]
            bv = limits_a[i][rc_id]

            mab = bv/av
            for idl, _ in enumerate(lim):
                limits_a[i][idl] -= limits_a[row][idl]*mab

            limits_b[i] -= limits_b[row]*(b_bv/av)

# получение результата

limits_a.append(dlt)

cols_t = list(zip(*limits_a))
cols = [list(sb) for sb in cols_t]

nd_cols_ids = []
for cid, col in enumerate(cols):
    cnt = 0
    el_id = 0
    for i, v in enumerate(col):
        if v != 0:
            cnt += 1
            el_id = i

    if cnt == 1:
        nd_cols_ids.append((el_id, cid))
```

```
nd_cols_ids.sort()

for col in nd_cols_ids:
    result_x[col[1]] = limits_b[col[0]]/limits_a[col[0]][col[1]]

f_max = limits_b[-1]

print('Ответ:')
print(*result_x[:sv_num])
print(f"F = {f_max}")
```

5. Экранные формы

```
Введите количество переменных:
2
Введите количество ограничений:
2
Введите коэффициенты целевой функции F(x) (2 числ. через пробел):
3 4
Введите тип функции:
1. max
2. min
1
Ввод ограничений №1 (3 числ. через пробел):
4 1 8
Введите тип ограничения:
1. >=
2. <=
3. =
2
Ввод ограничений №2 (3 числ. через пробел):
1 -1 -3
Введите тип ограничения:
1. >=
2. <=
3. =
1
Ответ:
1.0 4.0
F = 19.0
```

6. Вывод

В ходе выполнения лабораторной работы был изучен метод симплекса для решения оптимизационных задач линейного программирования. Метод позволяет найти значения переменных, при которых целевая функция достигает максимума или минимума при заданных ограничениях.