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.

1. 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.

1. 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.

1. 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]]

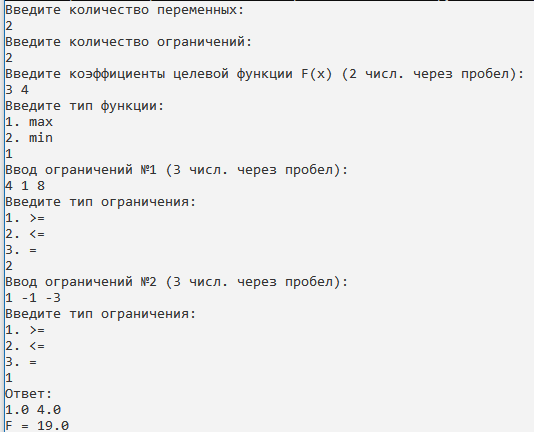
f\_max = limits\_b[-1]

print('Ответ:')

print(\*result\_x[:sv\_num])

print(f"F = {f\_max}")

1. Экранные формы



1. 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.