НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

Звіт

із лабораторної роботи  
з дисципліни «Алгоритми і системи комп’ютерної математики»

на тему

«Метод окантування для знаходження оберненої матриці»

|  |  |
| --- | --- |
| Виконав: | Перевірила: |
| студент групи КМ-51 | асистент кафедри ПМА |
| Галета М.С. | Ковальчук-Химюк Л.О. |
|  |  |
|  |  |
|  |  |

Київ — 2018

Зміст

[ВСТУП 3](#_Toc531559328)

[1 ПОСТАНОВКА ЗАДАЧІ 4](#_Toc531559329)

[2 ОПИС МЕТОДУ 6](#_Toc531559330)

[3 ВЕРИФІКАЦІЯ РОЗРОБЛЕНОЇ ПРОГРАМИ 8](#_Toc531559331)

[3.1 Варіативні вимоги до програмного забезпечення 8](#_Toc531559332)

[3.2 Тестові випадки 8](#_Toc531559333)

[ВИСНОВКИ 9](#_Toc531559334)

[ПЕРЕЛІК ПОСИЛАНЬ 10](#_Toc531559335)

[ДОДАТОК А. Граф потоків керувань програми 11](#_Toc531559336)

[ДОДАТОК Б. Робота програми на Python 12](#_Toc531559337)

[ДОДАТОК В. Лістинг програми 15](#_Toc531559338)

# ВСТУП

Лінійне програмування[1] або лінійна оптимізація (LP, англ. Linear Programming) — метод досягнення найліпшого виходу (такого як найбільший прибуток або найменша вартість) у математичній моделі чиї вимоги представлені через лінійні відношення. Лінійне програмування є особливим випадком математичного програмування (математичної оптимізації).

Більш формально, лінійне програмування є технікою для оптимізації лінійної цільової функції, що обмежена лінійними рівняннями і лінійними нерівностями. Її допустима множина є опуклим поліномом, який є множиною визначеною як перетин скінченної кількості півпростірів, кожен з яких визначає лінійна нерівність. Її цільова функція є дійсно значима афінна функція визначена на цьому багатограннику. Алгоритм лінійного програмування знаходить точку на багатограннику, де ця функція набуває найбільшого чи найменшого значення, якщо така точка існує.

# 1 ПОСТАНОВКА ЗАДАЧІ

Завданням лабораторної роботи є реалізація алгоритму транспортної задачі, що знаходить оптимальний план перевезень, який мінімізує витрати. Матриця тарифів, запаси постачальників та потреби споживачів наведені у Табл. 1.1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Запаси постачальників** | **Матриця тарифів** | | | | |
| 13 | 20 | 26 | 24 | 26 | 29 |
| 17 | 16 | 20 | 29 | 26 | 23 |
| 17 | 4 | 10 | 27 | 30 | 7 |
| 13 | 9 | 16 | 29 | 20 | 3 |
| **Потреби споживачів** | 12 | 12 | 12 | 12 | 12 |

Табл. 1.1 ­– Умова варіанту задачі

Введемо наступні позначення:

1. – кількість одиниць вантажу в -ого постачальника, ;
2. – потреба вантажу -ого споживача, ;
3. – ціна перевезення одиниці вантажу з -ого пункту в -ий;
4. – планована кількість вантажу для перевезення з -ого пункту в -ий.

Цільова функція матиме вигляд:

при обмеженнях:

Програмне забезпечення, розроблюване в рамках виконання лабораторної роботи, повинно задовольняти низку вимог.

До обов’язкових вимог належать:

а) перевірка на некоректне уведення для всіх полів уведення, зокрема:

1) порожнє уведення;

2) синтаксично некоректне уведення (наприклад, рядок «число» в поле для числових коефіцієнтів);

3) уведення спеціальних символів (наприклад, послідовність символів «%#&» в поле для числових коефіцієнтів);

4) уведення чисел, які перевищують максимальний розмір для чисел відповідного типу даних (наприклад, для полей з верхньою і нижньою границею значення більше(по модулю) ніж 3,4\*1034);

б) поява діалогового вікна з відповідним повідомленням у випадку некоректного уведення (наприклад, діалогове вікно, яке повідомляє користувачу, що уведення в поле можливе лише для числових значень);

в) визначення гранично допустимих значень для числових полів.

# 2 ОПИС МЕТОДУ

Метод потенціалів[2] — це метод послідовного покращення плану (перевезень) з використанням другої теореми двоїстості для перевірки оптимальності.

Для застосування методу потенціалів, спочатку треба знайти початковий опорний план. Одним із методів побудови опорного плану є метод північно-західного кута.

Алгоритм методу північно-західного кута:

1. Виконання починається з верхньої лівої клітини транспортної таблиці, тобто зі змінної , якій присвоюється максимальне значення, що допускається обмеженнями на попит і пропозицію.
2. Викреслюється рядок (або стовпець) з повністю реалізованою пропозицією (з задоволеним попитом). Це означає, що у викресленого рядку (стовпці) ми не будемо присвоювати значення іншим змінним (крім змінної, визначеної на першому етапі). Якщо одночасно задовольняються попит і пропозиція, викреслюється лише рядок або тільки стовпець.
3. Якщо не викреслено тільки один рядок або тільки один стовпець, процес зупиняється. В іншому випадку переходимо до клітини праворуч, якщо викреслять стовпець, або до клітини знизу, якщо викреслени й рядок. Потім повертаємось до першого етапу.

Після отримання опорного плану його оптимальність перевіряють за допомогою методу потенціалів.

У методі потенціалів кожному рядку і кожному стовпцю транспортної таблиці ставляться у відповідність числа (потенціали) і . Для кожної базисної змінної , потенціали і задовольняють рівнянню:

Щоб знайти значення потенціалів з цієї системи рівнянь, потрібно присвоїти одному з них довільне значення (зазвичай вважають ) і потім послідовно обчислювати значення інших потенціалів.

Далі, використовуючи знайдені значення потенціалів, для кожної небазисной змінної обчислюються величини

.

Якщо всі ці числа є недодатними то опорний план є оптимальним і розв'язування на цьому завершується. В іншому випадку знаходиться найбільше додатне значення і відповідна йому змінна вводиться в базис. Для визначення змінної, що виводиться з базису будується послідовність:

,

де — змінна, що вводиться в базис, а всі інші змінні є базисними. Окрім цього в цій послідовності при переході на кожному етапі одна координата залишається незмінною і якщо при певному переході незмінною була перша координата, то на наступному незмінною буде друга. Якщо зображувати перехід між змінними на транспортній таблиці стрілками між відповідними клітинами це означає, що переходи можуть бути лише вертикальними чи горизонтальними, але не діагональними, і також після горизонтального переходу має йти вертикальний і навпаки.

Після побудови послідовності можна записати значення відповідних змінних і знайти мінімальне значення серед чисел, що стоять на непарних позиціях. Наступним кроком це число слід додати до всіх змінних, що стоять на парних позиціях і відняти від всіх змінних, що стоять на непарних. Змінна якій відповідало найменше число виводиться з базиса.

У такий спосіб одержується новий опорний план і до нього можна знову застосувати ті ж дії.

3 ВЕРИФІКАЦІЯ РОЗРОБЛЕНОЇ ПРОГРАМИ

3.1 Варіативні вимоги до програмного забезпечення

Програма написана на мові програмування Python. При відкритті програми з’являється вікно, в якому користувачеві пропонується обрати кількість стовпців (постачальників) (від 2 до 10) та кількість рядків (споживачів) (від 2 до 10). Після натискання кнопки ОК відкривається друге вікно, яке містить поля для введення даних для транспортної задачі. На всі поля накладено валідацію шляхом регулярного виразу для введення лише дійсних чисел з проміжку [0; 100000000), тому користувач не зможе ввести літери алфавіту чи недопустимі символи. Якщо користувач залише якесь поле незаповненим, программа виведе повідомлення про те, що всі поля мають бути заповнені

# 3.2 Тестові випадки

Результат виконання програми наведено в Додатку Б. Перевірка результату здійснювалась за допомогою додаткового компоненту «Поиск решения» в MS Excel.



Рис. 3.2.2 – Перевірка результатів в MS Excel

# ВИСНОВКИ

Під час виконання лабораторної роботи було розроблено програмне забезпечення для реалізації методу потенціалів для транспортної задачі.

В результаті роботи програми була розв’язана поставлена задача, точність розв’язку якої було перевірено в MS Excel.

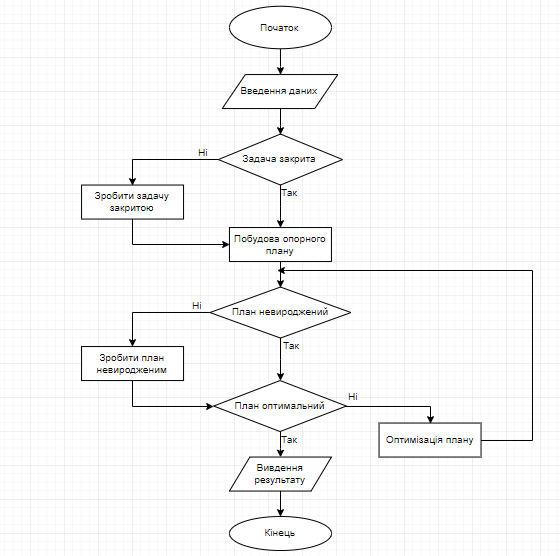
Створений граф потоків керування коду розробленої програми, за допомогою якого формувалися тестові випадки.

# ПЕРЕЛІК ПОСИЛАНЬ

1. [Електронний ресурс] https://uk.wikipedia.org/wiki/Лінійне\_програмування
2. [Електронний ресурс] https://uk.wikipedia.org/wiki/Транспортна\_задача

# ДОДАТОК А. Граф потоків керувань програми

Граф потоку керування



# ДОДАТОК Б. Робота програми на Python

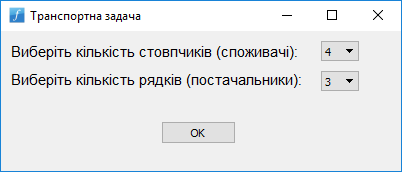


Рис. Б.1 – Результат відкриття програми

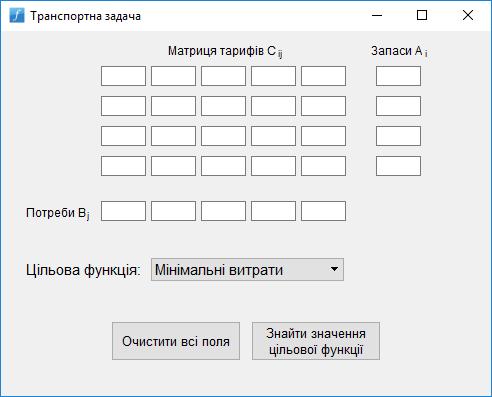


Рис. Б.2 – Результат відкриття вікна для введення даних

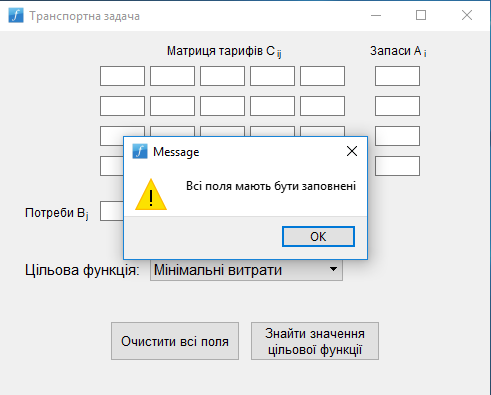


Рис. Б.3 – Результат порожнього введення

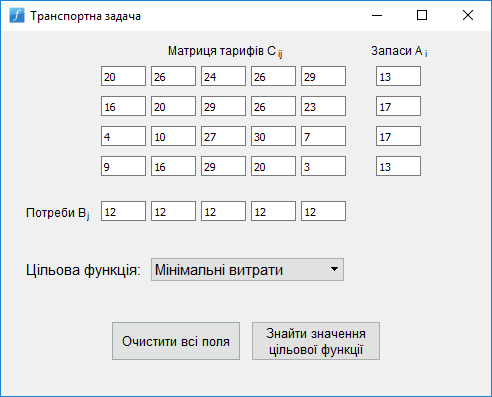


Рис. Б.3 – Результат заповнення даних

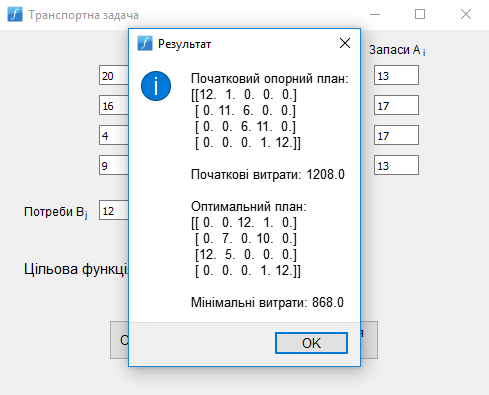


Рис. Б.4 – Виведення результату

# ДОДАТОК В. Лістинг програми

interface.py

import sys

from PyQt5.QtWidgets import QWidget

from PyQt5.QtWidgets import QLabel

from PyQt5.QtWidgets import QComboBox

from PyQt5.QtWidgets import QApplication

from PyQt5.QtWidgets import QPushButton

from PyQt5.QtWidgets import QLineEdit

from PyQt5.QtWidgets import QMessageBox

from PyQt5.QtGui import QIcon, QFont

from PyQt5 import QtCore, QtGui

from transport\_problem import TransportProblem

import numpy as np

class MainWindow(QWidget):

def \_\_init\_\_(self, parent=None):

super().\_\_init\_\_(parent)

self.number\_of\_customers = "4"

self.number\_of\_vendors = "3"

self.secondWin = None

self.initUI()

def initUI(self):

self.lbl\_1 = QLabel(self)

self.lbl\_1.move(10, 12)

self.lbl\_1.setFont(QFont("Arial", 11))

self.lbl\_1.setText("Виберіть кількість cтовпчиків (споживачі): ")

self.combo\_1 = QComboBox(self)

for i in range(9):

self.combo\_1.addItem(str(i+2), i)

self.combo\_1.move(320, 10)

self.combo\_1.setCurrentIndex(2)

self.combo\_1.activated.connect(self.set\_number\_of\_customers)

self.lbl\_2 = QLabel(self)

self.lbl\_2.move(10, 40)

self.lbl\_2.setFont(QFont("Arial", 11))

self.lbl\_2.setText("Виберіть кількість рядків (постачальники): ")

self.combo\_2 = QComboBox(self)

for i in range(9):

self.combo\_2.addItem(str(i+2), i)

self.combo\_2.move(320, 40)

self.combo\_2.setCurrentIndex(1)

self.combo\_2.activated.connect(self.set\_number\_of\_vendors)

self.okbutton = QPushButton("OK", self)

self.okbutton.move(160, 90)

self.okbutton.clicked.connect(self.openWin)

self.setGeometry(150, 150, 400, 140)

self.setWindowTitle('Транспортна задача')

self.setWindowIcon(QIcon('icon.jpg'))

self.show()

def set\_number\_of\_customers(self, index):

self.number\_of\_customers = self.combo\_1.itemText(index)

def set\_number\_of\_vendors(self, index):

self.number\_of\_vendors = self.combo\_2.itemText(index)

def openWin(self):

if self.secondWin is not None:

self.secondWin = None

self.secondWin = SecondWindow(self.number\_of\_customers,

self.number\_of\_vendors)

self.secondWin.show()

class SecondWindow(QWidget):

def \_\_init\_\_(self, number\_of\_customers,

number\_of\_vendors, parent=None):

super().\_\_init\_\_(parent)

self.matrix\_entry = []

self.b = []

self.a = []

self.combo = QComboBox(self)

self.m = int(number\_of\_customers)

self.n = int(number\_of\_vendors)

self.initUI()

def initUI(self):

reg = QtCore.QRegExp("^[0-9]{1,8}(\.[0-9]{1,5})?")

validator = QtGui.QRegExpValidator(reg)

self.C\_str = QLabel(self)

self.C\_str.move((85+50\*(self.m))//2, 12)

self.C\_str.setFont(QFont("Arial", 9))

self.C\_str.setText('Матриця тарифів C')

self.sub\_1 = QLabel(self)

self.sub\_1.move(127+(100+50\*(self.m-1))//2, 17)

self.sub\_1.setFont(QFont("Arial", 7))

self.sub\_1.setText('ij')

for i in range(self.m):

self.matrix\_entry.append([])

for j in range(self.n):

self.matrix\_entry[i].append(QLineEdit(self))

self.matrix\_entry[i][j].resize(45, 20)

self.matrix\_entry[i][j].move(100+50\*i, 35+30\*j)

self.matrix\_entry[i][j].setValidator(validator)

self.b\_str = QLabel(self)

self.b\_str.move(70+50\*(i+2), 12)

self.b\_str.setFont(QFont("Arial", 9))

self.b\_str.setText("Запаси A")

self.sub\_2 = QLabel(self)

self.sub\_2.move(124+50\*(i+2), 17)

self.sub\_2.setFont(QFont("Arial", 7))

self.sub\_2.setText("i")

for j in range(self.n):

self.b.append(QLineEdit(self))

self.b[j].resize(45, 20)

self.b[j].move(75+50\*(i+2), 35+30\*j)

self.b[j].setValidator(validator)

self.a\_str = QLabel(self)

self.a\_str.move(25, 84+30\*j)

self.a\_str.setFont(QFont("Arial", 9))

self.a\_str.setText("Потреби B")

self.sub\_2 = QLabel(self)

self.sub\_2.move(86, 89+30\*j)

self.sub\_2.setFont(QFont("Arial", 7))

self.sub\_2.setText("j")

for i in range(self.m):

self.a.append(QLineEdit(self))

self.a[i].move(100+50\*i, 80+30\*j)

self.a[i].resize(45, 20)

self.a[i].setValidator(validator)

self.func\_label = QLabel(self)

self.func\_label.move(25, 80+30\*(j+2))

self.func\_label.setFont(QFont("Arial", 11))

self.func\_label.setText("Цільова функція:")

self.combo.addItem("Мінімальні витрати", 0)

self.combo.addItem("Максимальний прибуток", 1)

self.combo.move(150, 77+30\*(j+2))

self.combo.setFont(QFont("Arial", 11))

self.combo.setCurrentIndex(0)

but = QPushButton("Знайти значення\nцільової функції", self)

but.resize(130, 40)

but.move((80+70\*(self.m+1))//2, 50+30\*(j+5))

but.setFont(QFont("Arial", 10))

but.clicked.connect(self.solve)

clear\_line\_button = QPushButton("Очистити всі поля", self)

clear\_line\_button.resize(130, 40)

clear\_line\_button.move((70\*(self.m+1)-200)//2,

50+30\*(j+5))

clear\_line\_button.setFont(QFont("Arial", 10))

clear\_line\_button.clicked.connect(self.clear\_line)

self.setGeometry(150, 150, 70+60\*(i+3), 35+30\*(j+8))

self.setWindowTitle('Транспортна задача')

self.setWindowIcon(QIcon('icon.jpg'))

self.show()

def clear\_line(self):

for i in range(self.m):

for j in range(self.n):

self.matrix\_entry[i][j].clear()

for i in range(self.n):

self.b[i].clear()

for i in range(self.m):

self.a[i].clear()

def isEmpty(self):

empty = False

for i in np.reshape(self.matrix\_entry, self.m\*self.n):

if i.text() == "":

empty = True

break

if empty:

return empty

for i in self.b:

if i.text() == "":

empty = True

break

if empty:

return empty

for i in self.a:

if i.text() == "":

empty = True

break

return empty

def solve(self):

if self.isEmpty():

QMessageBox.warning(self, "Message",

"Всі поля мають бути заповнені")

else:

choise = self.combo.currentIndex()

C = np.zeros((self.n, self.m))

for i in range(self.n):

for j in range(self.m):

C[i][j] = float(self.matrix\_entry[j][i].text())

supply = np.zeros(self.n)

demand = np.zeros(self.m)

for i in range(self.n):

supply[i] = float(self.b[i].text())

for i in range(self.m):

demand[i] = float(self.a[i].text())

if choise == 0:

result = TransportProblem(C, supply, demand, self.n, self.m)()

else:

result = TransportProblem(C, supply, demand,

self.n, self.m, isMax=True)()

X\_initial, res\_initial, X, optimal = result

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Information)

if choise == 0:

msg.setText("Початковий опорний план:\n"+

"{}\n\n".format(X\_initial)+

"Початкові витрати: {}\n\n".format(res\_initial)+

"Оптимальний план:\n"+

"{}\n\n".format(X)+

"Мінімальні витрати: {}".format(optimal))

else:

msg.setText("Початковий опорний план:\n"+

"{}\n\n".format(X\_initial)+

"Початковий прибуток: {}\n\n".format(res\_initial)+

"Оптимальний план:\n"+

"{}\n\n".format(X)+

"Максимальний прибуток: {}".format(optimal))

msg.setWindowTitle("Результат")

msg.setFont(QFont("Arial", 10))

msg.exec\_()

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

w = MainWindow()

sys.exit(app.exec\_())

transport\_problem.py

import numpy as np

class TransportProblem:

def \_\_init\_\_(self, C, supply, demand, n, m, isMax=False):

self.C = C

self.supply = supply

self.demand = demand

self.n = n

self.m = m

self.isMax = isMax

self.cycle = None

def \_\_call\_\_(self):

opened\_1 = False

opened\_2 = False

if not self.isClosed(self.supply, self.demand):

delta = np.sum(self.supply) - np.sum(self.demand)

if delta > 0:

opened\_1 = True

self.demand = np.append(self.demand, delta)

self.C = np.append(self.C, np.zeros((self.n, 1)), axis=1)

self.m += 1

else:

opened\_2 = True

self.supply = np.append(self.supply, -delta)

self.C = np.append(self.C, np.zeros((1, self.m)), axis=0)

self.n += 1

X = self.north\_west\_conner\_method(self.n, self.m,

self.supply, self.demand)

X\_initial = np.copy(X)

res\_initial = np.sum(X\*self.C)

for i in range(self.n):

for j in range(self.m):

if X[i][j] == 0:

X[i][j] = np.nan

while True:

if self.isDegenerate(X, self.n, self.m):

X = self.make\_nondegenerate(X, self.n, self.m)

alpha, beta = self.potentials\_calculation(self.C, X)

est = self.cells\_estimation(self.C, self.n, self.m, alpha, beta)

if self.isOptimal(est, self.isMax):

for i in range(self.n):

for j in range(self.m):

if np.isnan(X[i][j]):

X[i][j] = 0

break

row, col = self.max\_estimation\_coords(est, self.isMax)

self.find\_cycle(X, self.n, self.m, row, col)

X = self.recalculation(X, self.cycle, self.n, self.m)

if opened\_1:

return ((X\_initial.T[:-1]).T, res\_initial,

(X.T[:-1]).T, np.sum(X\*self.C))

if opened\_2:

return X\_initial[:-1], res\_initial, X[:-1], np.sum(X\*self.C)

return X\_initial, res\_initial, X, np.sum(X\*self.C)

def isClosed(self, supply, demand):

if np.sum(supply) == np.sum(demand):

return True

else:

return False

def north\_west\_conner\_method(self, n, m, supply, demand):

X = np.zeros((n, m))

for i in range(n):

for j in range(m):

X[i][j] = min(supply[i] - np.sum(X, axis=1)[i],

demand[j] - np.sum(X, axis=0)[j])

return X

def isDegenerate(self, X, n, m):

if np.count\_nonzero(np.isfinite(X)) != n+m-1:

return True

else:

return False

def make\_nondegenerate(self, X, n, m):

delta = n + m - 1 - np.count\_nonzero(np.isfinite(X))

while delta > 0:

b = False

for i in range(n):

for j in range(m):

if np.isnan(X[i][j]):

if not self.find\_cycle(X, n, m, i, j):

X[i][j] = 0

b = True

break

if b:

break

delta -= 1

return X

def potentials\_calculation(self, C, X):

alpha = [None] \* C.shape[0]

beta = [None] \* C.shape[1]

alpha[0] = 0.

for j in range(C.shape[1]):

if np.isfinite(X[0][j]):

beta[j] = C[0][j] - alpha[0]

while (any(u is None for u in alpha[1:]) or

any(v is None for v in beta)):

for i in range(1, C.shape[0]):

for j in range(C.shape[1]):

if (np.isfinite(X[i][j]) and beta[j] is None

and alpha[i] is None):

continue

elif np.isfinite(X[i][j]) and beta[j] is None:

beta[j] = C[i][j] - alpha[i]

elif np.isfinite(X[i][j]) and alpha[i] is None:

alpha[i] = C[i][j] - beta[j]

return np.array(alpha), np.array(beta)

def cells\_estimation(self, C, n, m, alpha, beta):

est = np.zeros((n, m))

for i in range(n):

for j in range(m):

est[i][j] = alpha[i] + beta[j] - C[i][j]

return est

def isOptimal(self, est, isMax):

if isMax:

if (est >= 0).all():

return True

else:

return False

else:

if (est <= 0).all():

return True

else:

return False

def max\_estimation\_coords(self, est, isMax):

if isMax:

row, col = np.unravel\_index(np.argmin(est), est.shape)

else:

row, col = np.unravel\_index(np.argmax(est), est.shape)

return row, col

def find\_cycle(self, X, n, m, row, col):

self.cycle = [[row, col]]

if not self.look\_horizontally(X, self.cycle, n, m, row, col, row, col):

return False

else:

return True

def look\_horizontally(self, X, cycle, n, m, row, col, row\_next, col\_next):

for j in range(m):

if j != col and np.isfinite(X[row][j]):

if j == col\_next:

cycle.append([row, j])

return True

if self.look\_vertically(X, cycle, n, m, row, j,

row\_next, col\_next):

cycle.append([row, j])

return True

return False

def look\_vertically(self, X, cycle, n, m, row, col, row\_next, col\_next):

for i in range(n):

if i != row and np.isfinite(X[i][col]):

if i == row\_next:

cycle.append([i, col])

return True

if self.look\_horizontally(X, cycle, n, m, i, col,

row\_next, col\_next):

cycle.append([i, col])

return True

return False

def recalculation(self, X, cycle, n, m):

min\_elem = np.min([X[i][j] for i, j in cycle[1::2]])

k = cycle[0][0]

l = cycle[0][1]

X[k][l] = min\_elem

cycle\_len = len(cycle)

for i in range(1, cycle\_len):

k = cycle[i][0]

l = cycle[i][1]

if i%2 == 0:

X[k][l] += min\_elem

else:

X[k][l] -= min\_elem

if X[k][l] == 0.:

X[k][l] = np.nan

return X