Міністерство освіти й науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Кафедра автоматизації проектування енергетичних процесів і систем

ЗВІТ

Про виконання лабораторної роботи №3

З дисципліни «Комп’ютерне моделювання та оптимізація»

**Методи послідовних наближень для оптимізації функції багатьох змінних**

TI-92 Черноусова Дениса

Перевірив проф. д.т.н. Шушура О. М.

Київ – 2021



g = 3

k = 2

g – остання цифра у номері студентського квитка, а k – передостання.

Розробити алгоритми та програмне забезпечення для розв’язку наведених задач оптимізації. Алгоритми представити у вигляді блок-схем або діаграм діяльності UML. Програмне забезпечення розробити на будь-якій сучасній мові програмування. Знайти точне значення розв’язку задачі за допомогою математичних бібліотек та порівняти його зі значенням, отриманим в результаті роботи розробленого програмного забезпечення.

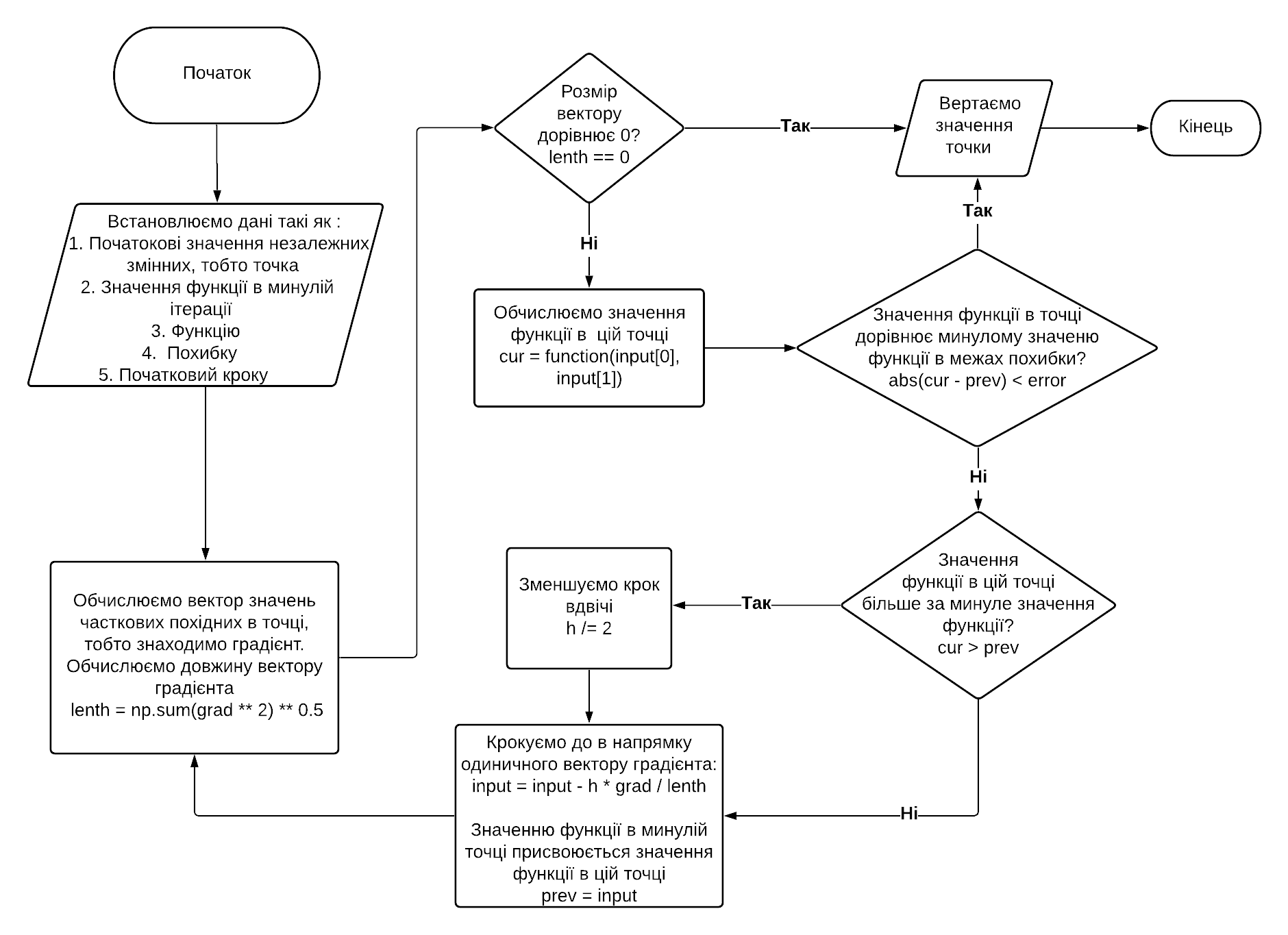
Задачі:

1. Для функції f(x,y) = х2+(y-g)2 -2kgx + k знайти точку локального мінімуму (де g – остання цифра у номері студентського квитка, а k – передостання) методами:

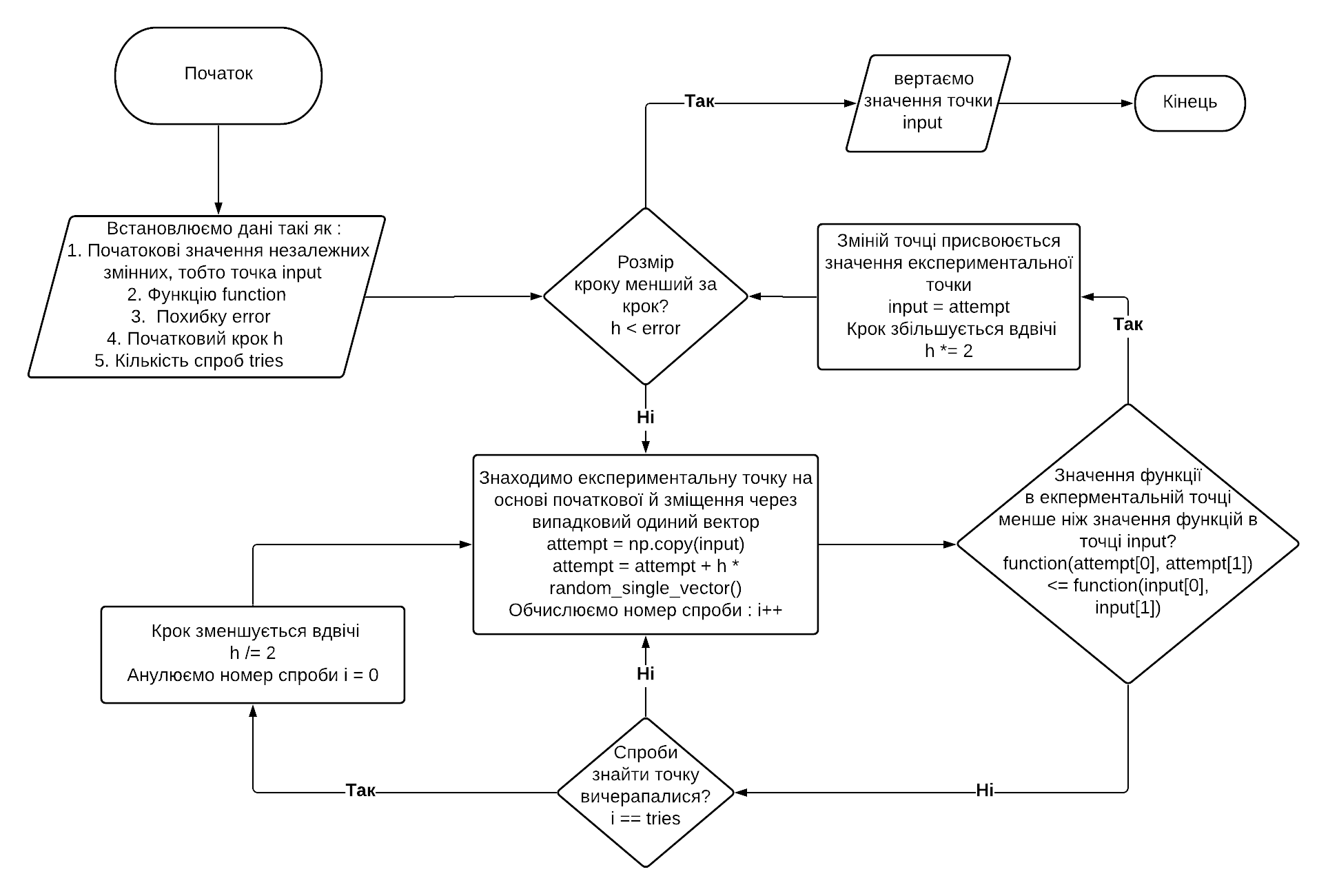
1. градієнтним методом з поділом кроку навпіл (у якості початкової точки взяти (kg+2,g-3), початкове значення кроку h=1, абсолютна точність пошуку має дорівнювати 0,01)
2. адаптивний метод випадкового пошуку для багатовимірного випадку (параметри методу визначити самостійно для отримання розв’язку з точністю, близькою до градієнтного методу)

**Алгоритми**

Градієнтний метод

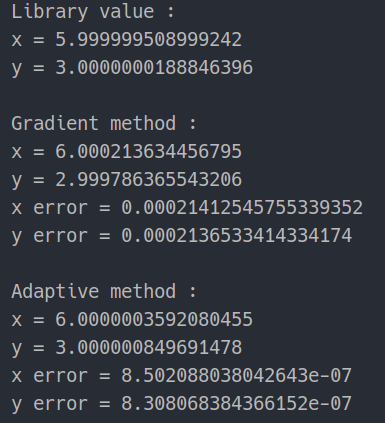


Адаптивний метод

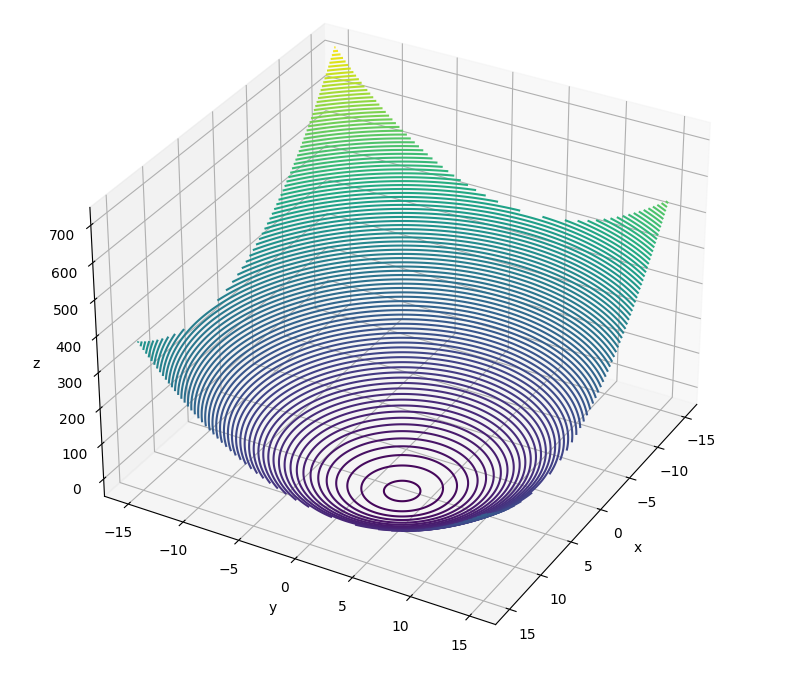


**Результати роботи**

Результати обчислень



Графік функції



**Лістинг**

main.py

*from* adaptive *import* start\_adaptive  
*from* gradient\_descent *import* start\_gradient  
*from* libs *import* \*  
  
lib\_x, lib\_y = lib\_calculate()  
print("Library value :")  
print("x =", lib\_x, "\ny =", lib\_y)  
  
grad\_x, grad\_y = start\_gradient(np.array([x0, y0]), 1)  
  
print("\nGradient method :")  
print("x =", grad\_x, "\ny =", grad\_y)  
print("x error =", abs(grad\_x - lib\_x), "\ny error =", abs(grad\_y - lib\_y))  
  
adap\_x, adap\_y = start\_adaptive(np.array([x0, y0]))  
  
print("\nAdaptive method :")  
print("x =", adap\_x, "\ny =", adap\_y)  
print("x error =", abs(adap\_x - lib\_x), "\ny error =", abs(adap\_y - lib\_y))  
  
draw()

libs.py

*from* matplotlib *import* pyplot *as* plt  
*from* scipy *import* optimize  
  
*from* function *import* \*  
  
  
*def* lib\_calculate():  
 optimization = optimize.minimize(function, x0, args=(x0,))  
 best\_x = optimization.x[0]  
 optimization = optimize.minimize(*lambda* x, y: function(y, x), y0, args=(y0,))  
 best\_y = optimization.x[0]  
 *return* best\_x, best\_y  
  
  
*def* draw():  
 y = x = np.linspace(-15, 15, 1000)  
 x, y = np.meshgrid(x, y)  
 z = function(x, y)  
 ax = plt.axes(projection='3d')  
 ax.contour3D(x, y, z, 100)  
 ax.set\_xlabel('x')  
 ax.set\_ylabel('y')  
 ax.set\_zlabel('z')  
 plt.show()

function.py

*import* numpy *as* np  
  
G = 3  
K = 2  
  
x0 = (K \* G + 2)  
y0 = (G - 2)  
error = 10 \*\* -6  
  
  
*def* function(x, y):  
 *return* x \*\* 2 + (y - G) \*\* 2 - 2 \* K \* G \* x + K  
  
  
*def* function\_x\_der(x):  
 *return* 2 \* x - 2 \* K \* G  
  
  
*def* function\_y\_der(y):  
 *return* 2 \* y - 2 \* G  
  
  
*def* function\_der(input):  
 *return* np.array([function\_x\_der(input[0]), function\_y\_der(input[1])])

gradient\_descent.py

*from* function *import* \*  
*import* numpy *as* np  
  
  
*def* start\_gradient(input: np, h):  
 *return* iteration(input, 0, h)  
  
  
*def* iteration(input, prev, h):  
 grad = function\_der(input)  
 lenth = np.sum(grad \*\* 2) \*\* 0.5  
 *if* lenth == 0: *return* input  
  
 cur = function(input[0], input[1])  
 *# print("current\_z=" + str(cur) + "; previous\_z=" + str(prev) + "; h=" + str(h))*  
  
 *if* abs(cur - prev) < error: *return* input  
 *if* cur > prev: h /= 2  
 *return* iteration(input - h \* grad / lenth, cur, h)

adaptive.py

*import* random  
  
*from* function *import* \*  
  
h = 1  
tries = 20  
  
  
*def* random\_single\_vector():  
 e = np.array([random.random()-0.5, random.random()-0.5])  
 *return* e / (np.sum(e \*\* 2) \*\* 0.5)  
  
  
*def* start\_adaptive(input: np):  
 *return* iteration(input, h)  
  
  
*def* iteration(input, h):  
 *if* h < error: *return* input  
 *for* i *in* range(tries):  
 attempt = np.copy(input)  
 attempt = attempt + h \* random\_single\_vector()  
 *if* function(attempt[0], attempt[1]) <= function(input[0], input[1]):  
 *return* iteration(attempt, h \* 2)  
  
 *return* iteration(input, h / 2)

**Висновки**

Під час цієї лабораторної роботи були розроблені два алгоритми за градієнтним методом та адаптивним методом для пошуку мінімуму функції багатьох зміних. Згідно спостереженням, градієнтний спуск є стільки ж ефективним як адаптивний метод, проте потребує значно менше ітерацій для знаходження точного значення. В ході обчилень отримані результати задовльняють абсолютну задану похибку в порівняні зі значенням отриманим з бібліотеки.