**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
імені ІГОРЯ СІКОРСЬКОГО»**

**Навчально-науковий інститут прикладного системного аналізу  
Кафедра штучного інтелекту**

**Звіт**

**про виконання лабораторної роботи №1  
з дисципліни «Інтелектуальний аналіз даних»**

Виконали:  
студент III курсу, груп КІ-33 та КІ-31  
Тал Майк, Піддубна Марія, Дмитренко Владислав

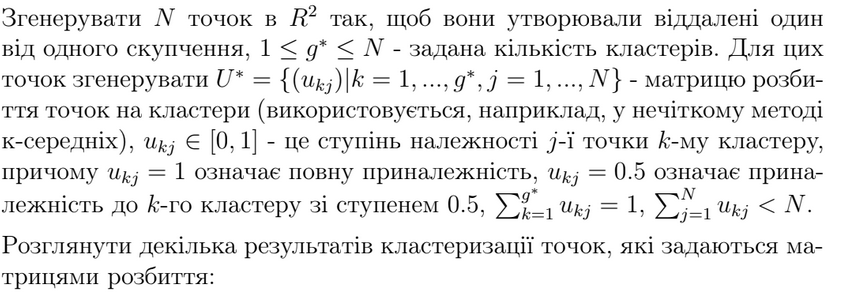
Прийняв:

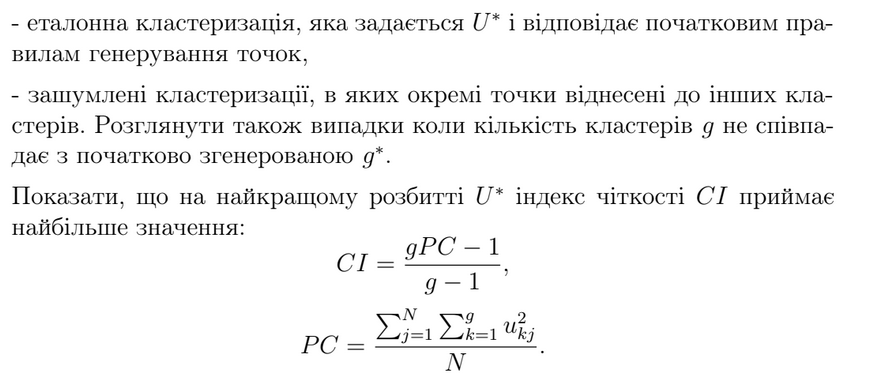
Викладач: Андросов Д.В.

Київ – 2025

**Варіант 14**

**Завдання:**





**Текст програми:**

# Автор: Тал Майк, Піддубна Марія, Дмитренко Владислав, студенти груп КІ-33 та КІ-31.

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.datasets import make\_blobs

# Параметри

N = 100 # кількість точок

g\_star = 3 # задана кількість кластерів

np.random.seed(42)

# Генеруємо N точок в R2 так, щоб вони утворювали віддалені скупчення

X, true\_labels = make\_blobs(

n\_samples=N,

centers=g\_star,

cluster\_std=1.5,

center\_box=(-10, 10),

random\_state=42,

)

print(f"Згенеровано {N} точок з {g\_star} кластерами")

print(f"Координати перших 5 точок:")

for i in range(5):

print(f"Точка {i+1}: ({X[i,0]:.2f}, {X[i,1]:.2f})")

def generate\_membership\_matrix(points, num\_clusters):

"""Генерує матрицю розбиття U"""

kmeans = KMeans(n\_clusters=num\_clusters, random\_state=42)

kmeans.fit(points)

centers = kmeans.cluster\_centers\_

# Обчислюємо відстані від точок до центрів

U = np.zeros((num\_clusters, len(points)))

for j in range(len(points)):

distances = []

for k in range(num\_clusters):

dist = np.sqrt(np.sum((points[j] - centers[k]) \*\* 2))

distances.append(dist)

# Конвертуємо відстані в коефіцієнти належності

for k in range(num\_clusters):

if distances[k] == 0:

U[k, j] = 1.0

else:

sum\_part = 0

for i in range(num\_clusters):

if distances[i] > 0:

sum\_part += (distances[k] / distances[i]) \*\* 2

U[k, j] = (

1.0 / sum\_part if sum\_part > 0 else 1.0 / num\_clusters

)

# Нормалізуємо щоб сума = 1

for j in range(len(points)):

col\_sum = np.sum(U[:, j])

if col\_sum > 0:

U[:, j] = U[:, j] / col\_sum

return U

def add\_noise\_to\_matrix(U, noise\_level):

"""Додає шум до матриці"""

U\_noisy = U.copy()

noise = np.random.uniform(-noise\_level, noise\_level, U.shape)

U\_noisy += noise

U\_noisy = np.clip(U\_noisy, 0, 1) # обмежуємо [0,1]

# Перенормалізуємо

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

col\_sum = np.sum(U\_noisy[:, j])

if col\_sum > 0:

U\_noisy[:, j] = U\_noisy[:, j] / col\_sum

return U\_noisy

def calculate\_PC(U):

"""PC = сума всіх u^2 / N"""

return np.sum(U\*\*2) / U.shape[1]

def calculate\_CI(U):

"""CI = (g\*PC - 1) / (g - 1)"""

g = U.shape[0]

if g == 1:

return 1.0

PC = calculate\_PC(U)

CI = (g \* PC - 1) / (g - 1)

return CI

def print\_matrix(U, title, max\_show=8):

"""Друкує матрицю"""

print(f"\n{title}")

print("=" \* 50)

g, N = U.shape

print(f"Розмір: {g} кластерів x {N} точок")

show\_points = min(max\_show, N)

print(f"\nПерші {show\_points} точок:")

# Заголовок

header = "Кластер "

for j in range(show\_points):

header += f" Т{j+1:02d} "

print(header)

print("-" \* len(header))

# Рядки матриці

for k in range(g):

row = f" {k+1} "

for j in range(show\_points):

row += f" {U[k,j]:5.3f}"

print(row)

print("-" \* len(header))

# Сума по стовпцях

sum\_row = "Сума "

for j in range(show\_points):

col\_sum = np.sum(U[:, j])

sum\_row += f" {col\_sum:5.3f}"

print(sum\_row)

# Статистика

print(f"\nСтатистика матриці:")

print(f"Середнє значення: {np.mean(U):.4f}")

print(f"Мін: {np.min(U):.4f}, Макс: {np.max(U):.4f}")

# 1. Еталонна кластеризація U\*

print("\n" + "=" \* 60)

print("ГЕНЕРАЦІЯ МАТРИЦЬ РОЗБИТТЯ")

print("=" \* 60)

U\_star = generate\_membership\_matrix(X, g\_star)

print\_matrix(U\_star, "ЕТАЛОННА МАТРИЦЯ U\*")

PC\_star = calculate\_PC(U\_star)

CI\_star = calculate\_CI(U\_star)

print(f"PC = {PC\_star:.4f}")

print(f"CI = {CI\_star:.4f}")

# 2. Зашумлені кластеризації

print("\n" + "=" \* 60)

print("ЗАШУМЛЕНІ КЛАСТЕРИЗАЦІЇ")

print("=" \* 60)

results = []

results.append(("Еталонна U\*", U\_star, CI\_star, PC\_star, g\_star))

noise\_levels = [0.05, 0.1, 0.2]

for noise in noise\_levels:

U\_noisy = add\_noise\_to\_matrix(U\_star, noise)

PC\_noisy = calculate\_PC(U\_noisy)

CI\_noisy = calculate\_CI(U\_noisy)

results.append((f"Шум {noise}", U\_noisy, CI\_noisy, PC\_noisy, g\_star))

if noise == 0.1: # показуємо одну зашумлену матрицю

print\_matrix(U\_noisy, f"ЗАШУМЛЕНА МАТРИЦЯ (шум = {noise})")

print(f"PC = {PC\_noisy:.4f}")

print(f"CI = {CI\_noisy:.4f}")

# 3. Різна кількість кластерів

print("\n" + "=" \* 60)

print("РІЗНА КІЛЬКІСТЬ КЛАСТЕРІВ")

print("=" \* 60)

for g in [2, 4, 5]:

if g != g\_star:

U\_diff = generate\_membership\_matrix(X, g)

PC\_diff = calculate\_PC(U\_diff)

CI\_diff = calculate\_CI(U\_diff)

results.append((f"g = {g}", U\_diff, CI\_diff, PC\_diff, g))

if g == 4: # показуємо одну матрицю з іншим g

print\_matrix(U\_diff, f"МАТРИЦЯ З g = {g} (замість g\* = {g\_star})")

print(f"PC = {PC\_diff:.4f}")

print(f"CI = {CI\_diff:.4f}")

# Таблиця результатів

print("\n" + "=" \* 60)

print("РЕЗУЛЬТАТИ КЛАСТЕРИЗАЦІЇ")

print("=" \* 60)

print(f"{'Тип':<20} {'CI':<8} {'PC':<8} {'g':<3}")

print("-" \* 40)

for name, U, CI, PC, g in results:

print(f"{name:<20} {CI:<8.4f} {PC:<8.4f} {g:<3}")

# Знаходимо найкращий CI

best\_CI = max(results, key=lambda x: x[2])

print("-" \* 40)

print(f"Найкращий CI: {best\_CI[2]:.4f} ({best\_CI[0]})")

# Показати що на U\* індекс CI найбільший

print(f"\nПеревірка: еталонна U\* дає найвищий CI = {CI\_star:.4f}")

all\_CI = [r[2] for r in results]

is\_best = CI\_star == max(all\_CI)

print(f"Це підтверджує теорію: {is\_best}")

# Візуалізація

print("\n" + "=" \* 60)

print("ВІЗУАЛІЗАЦІЯ")

print("=" \* 60)

fig, axes = plt.subplots(2, 3, figsize=(15, 8))

axes = axes.flatten()

for i, (name, U, CI, PC, g) in enumerate(results[:6]):

if i < len(axes):

# Жорстке розбиття для візуалізації

hard\_labels = np.argmax(U, axis=0)

axes[i].scatter(

X[:, 0], X[:, 1], c=hard\_labels, cmap="viridis", alpha=0.7

)

axes[i].set\_title(f"{name}\nCI = {CI:.3f}")

axes[i].set\_xlabel("X1")

axes[i].set\_ylabel("X2")

axes[i].grid(True, alpha=0.3)

for i in range(len(results), len(axes)):

axes[i].set\_visible(False)

plt.tight\_layout()

plt.show()

# Графік порівняння CI

plt.figure(figsize=(10, 6))

names = [r[0] for r in results]

ci\_values = [r[2] for r in results]

bars = plt.bar(range(len(names)), ci\_values, alpha=0.7)

# Виділяємо еталонну червоним

bars[0].set\_color("red")

bars[0].set\_alpha(0.9)

plt.title("Порівняння індексу чіткості CI")

plt.ylabel("CI")

plt.xticks(range(len(names)), names, rotation=45)

plt.grid(True, alpha=0.3)

plt.tight\_layout()

plt.show()

print(f"\nВисновок: найкращий розбиття U\* має CI = {CI\_star:.4f}")

**Результати роботи програми:**

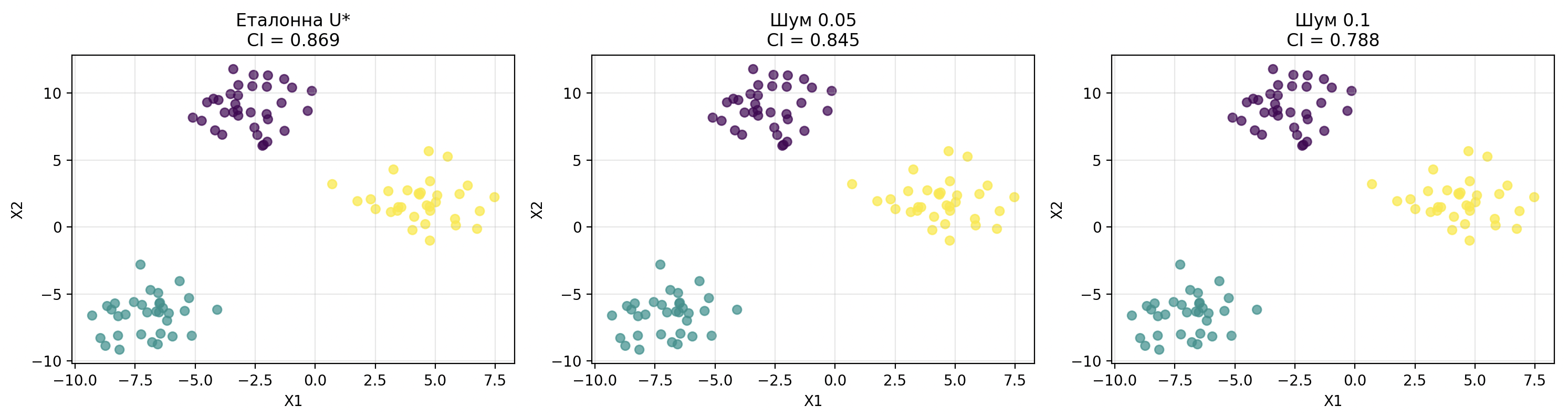
****

Рисунок 1— Кластеризація з різними параметрами

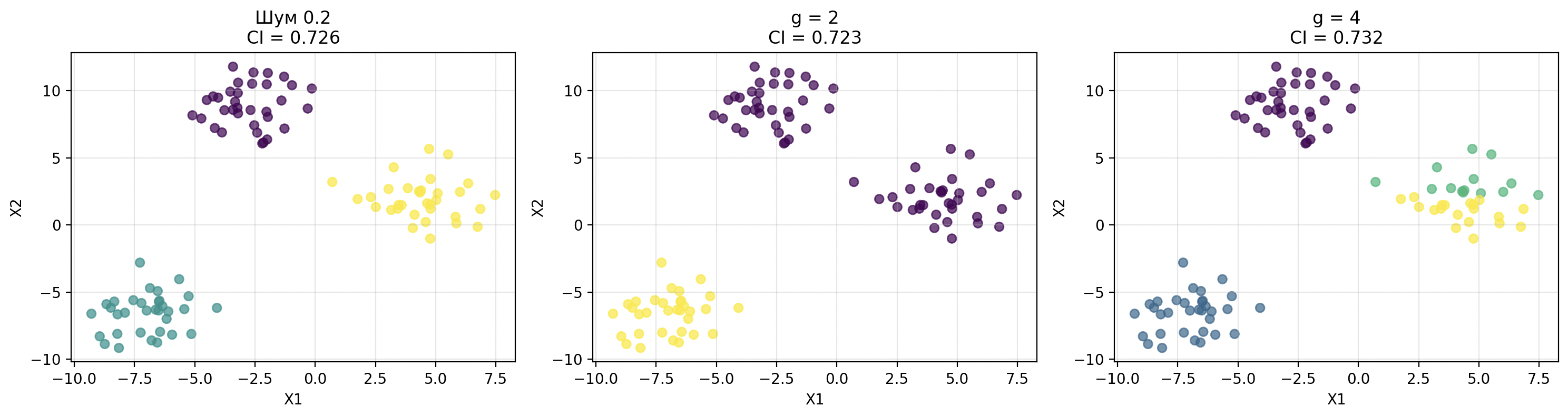
****

Рисунок 2— Кластеризація з різними параметрами

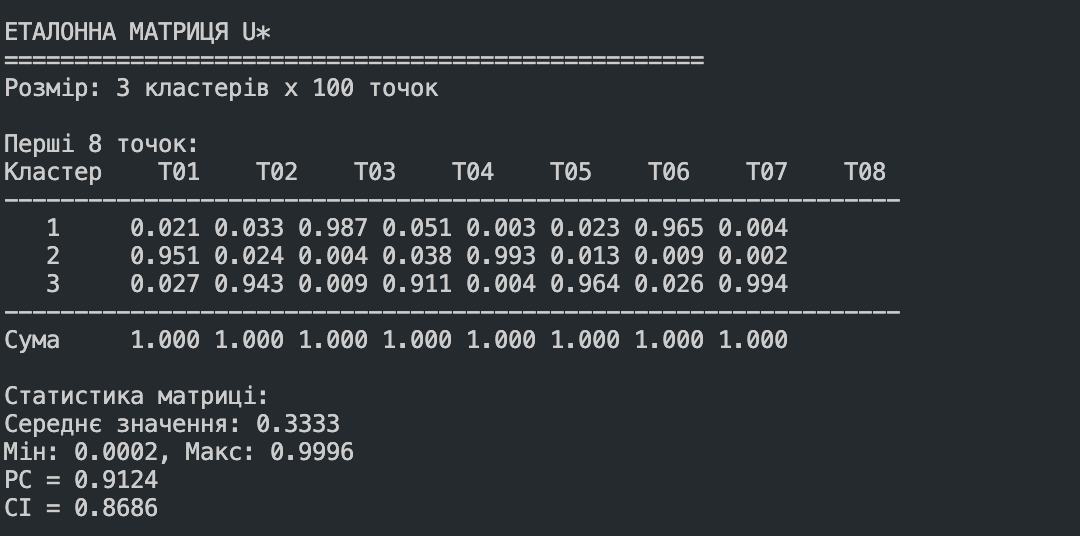


Рисунок 4 — Розподіл належності точок по кластерам, еталонна матриця

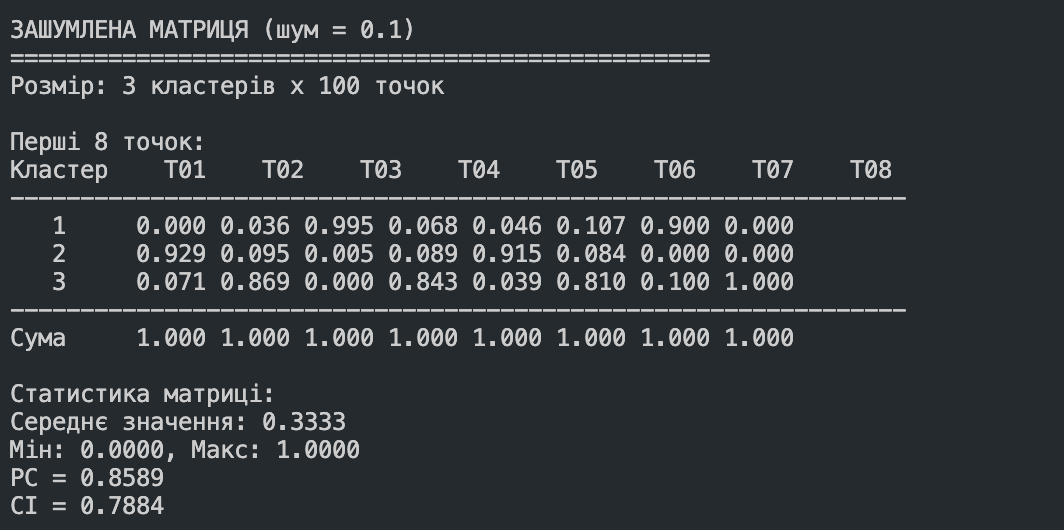


Рисунок 4 — Розподіл належності точок по кластерам, зашумлена матриця

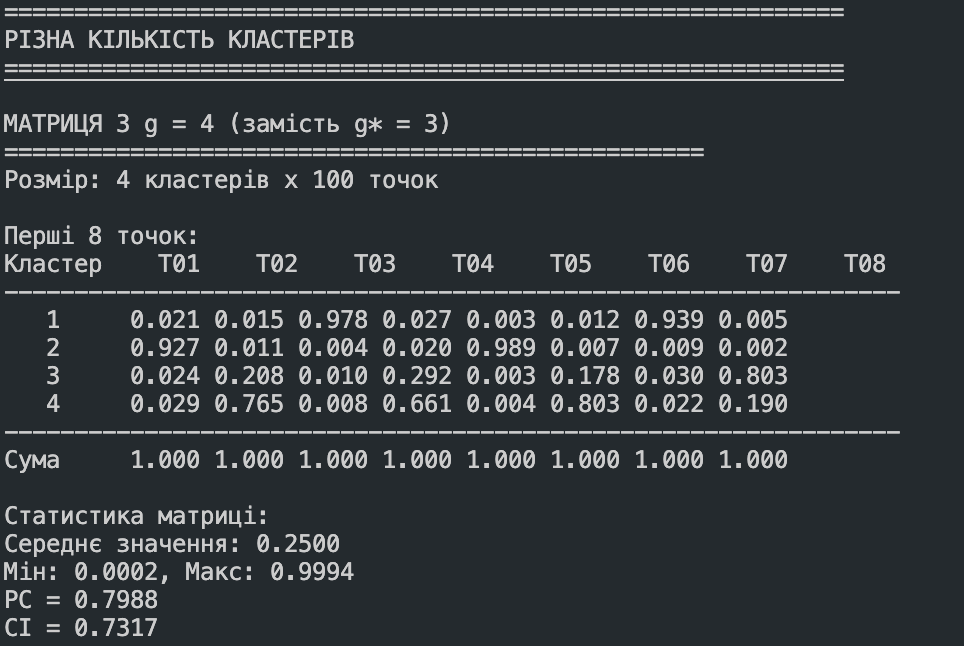


Рисунок 5 — Розподіл належності точок по кластерам, різна кількість кластерів

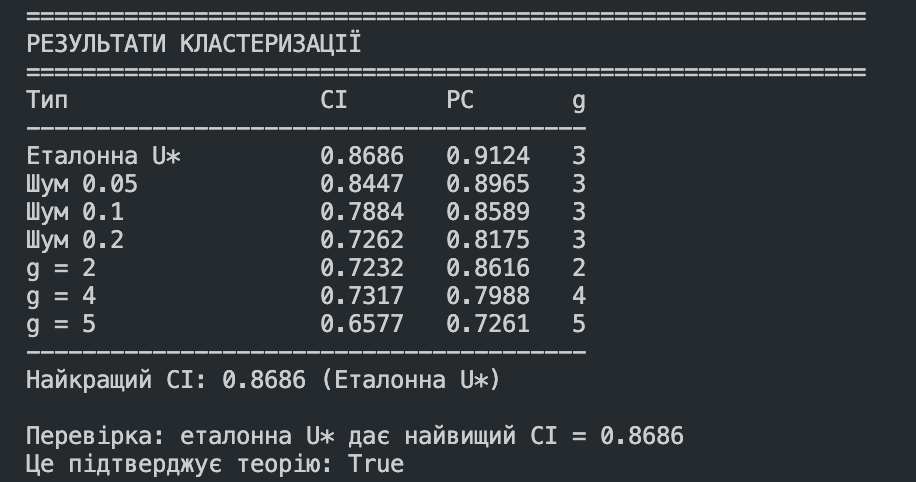


Рисунок 6 — Результати кластеризації

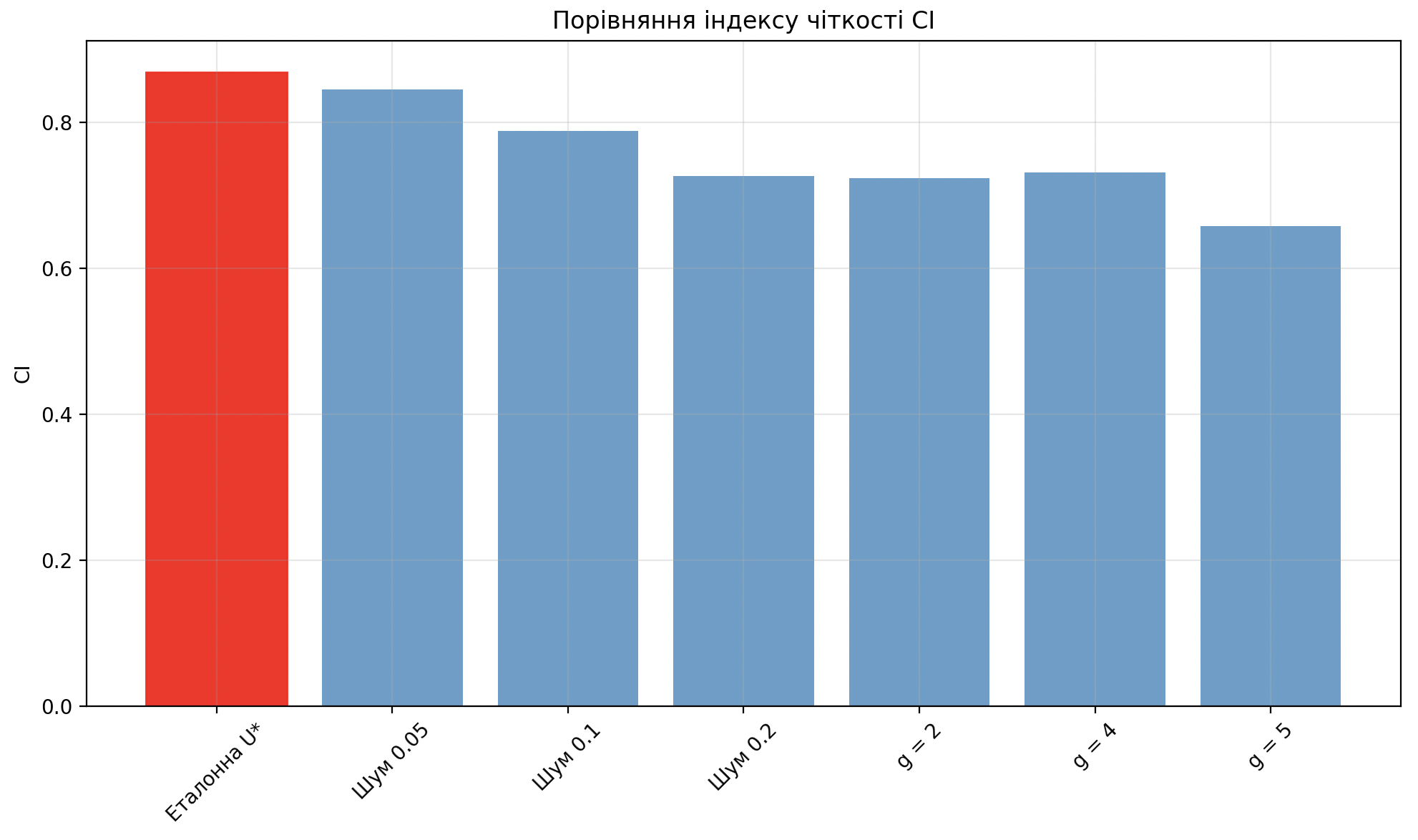


Рисунок 7 — Порівняння індексу чіткості CI

**Висновки:**

У результаті проведеного дослідження було здійснено аналіз матриць розбиття точок на кластери з використанням еталонного розподілу та його модифікацій із додаванням шуму та зміною кількості кластерів. Зокрема, еталонна матриця U∗, отримана за допомогою алгоритму K-середніх, продемонструвала найвищий індекс чіткості (CI) порівняно із зашумленими варіантами та альтернативними числами кластерів. Додавання шуму до матриці призводило до зменшення CI, що свідчить про втрату чіткості розбиття та зниження однозначності належності точок до кластерів. Аналогічно, зміна кількості кластерів від базового значення g∗=3gтакож впливала на показники чіткості, знижуючи CI при меншій або більшій кількості кластерів, ніж еталонна.

Візуалізація результатів підтвердила кількісні висновки: точки, розподілені за еталонною матрицею U∗, утворюють віддалені та компактні скупчення, що відображено у виразній диференціації кольорів при жорсткому розбитті. Порівняльний аналіз CI для всіх розглянутих варіантів дозволив виділити оптимальне розбиття, яке забезпечує максимальну чіткість кластеризації. Отримані результати підкреслюють ефективність використання CI як метрики для оцінки якості розбиття та демонструють важливість вибору адекватної кількості кластерів і врахування впливу шуму при практичній кластеризації даних.