Прикладний статистичний аналіз

Посібник для студентів

Ігор Мірошниченко

Юлія Хлевна

2025-04-10

Зміст

# Передмова

# Вступ

# 1. Біноміальний критерій

## 1.1 Генеральна сукупність та вибірка

Ви вирішили створити платформу онлайн-курсів з програмування. Ви записали навчальні відео та запропонували користувачам доступ за передплатою. Вартість курсу для студента становить 1000 гривень, а витрати на підтримку платформи та індивідуальні консультації коштують вам 500 гривень з кожного студента.

Проте ви помічаєте, що деякі люди відмовляються від курсу після першого заняття, якщо матеріал їм здається складним або нецікавим. Інвестори готові підтримати ваш проєкт, якщо рівень відмов буде нижче 50%.

Щоб це перевірити, ви проводите експеримент: залучаєте 30 нових студентів. 20 із них проходять курс й оплачують доступ, а 11 відмовляються. 20 — це більше половини, але чи достатньо цього, щоб довести перспективність проєкту?

Розв’язуючи таку задачу, ми припускаємо, що існує певна аудиторія, яка користуватиметься нашим сервісом. Цю групу називають **генеральною сукупністю**. Якщо запустити сервіс для всіх потенційних користувачів, у ньому буде певна частка успішних випадків, позначимо її як . Це невідомий параметр, який ми не можемо визначити безпосередньо. Натомість ми можемо проводити експерименти та **досліджувати** результати. Оскільки протестувати продукт на всій аудиторії неможливо, ми беремо **вибірку** з генеральної сукупності та аналізуємо частку успішних випадків.

Згідно з результатами нашого експерименту, спостережувана ймовірність оплати становить [[1]](#footnote-22). Це означає, що 67% студентів оплатили доступ. Чи можемо ми зробити висновок, що справжня частка успішних випадків перевищує 50%?

Розгляньмо, чому отримане значення *може не бути* переконливим доказом. Припустимо, що ймовірність успішної оплати дорівнює , і змоделюємо можливі результати для 30 студентів.

Давайте спростимо цю задачу до прикладу з підкиданням монетки та змоделюємо результати для 30 спроб:

* Якщо монетка випаде орлом, студент оплачує доступ.
* Якщо монетка випаде решкою, студент відмовляється від курсу.
* Використаємо метод integers()[[2]](#footnote-23) до класу Generator, яка генерує випадкові цілі числа в заданому діапазоні.
* Підкинемо монетку 30 разів та порахуємо кількість успішних випадків.

|  |
| --- |
| Лістинг 1.1: Підкидання монетки  rng = np.random.default\_rng(seed=18)  n = 30 results = rng.integers(0, 1, size = 30, endpoint = True) success = np.sum(results) / n  print(f"Кількість успішних випадків: {round(success, 3) \* 100}%")  Рядок 1  Ініціалізуємо генератор випадкових чисел з фіксованим seed.  Рядок 3  Кількість студентів.  Рядок 4  Генеруємо випадкові числа для кожного студента.  Рядок 5  Обчислюємо частку успішних випадків.  Рядок 7  Виводимо результат.  Кількість успішних випадків: 70.0% |

Ми бачимо, що в експерименті частка успішних випадків навіть перевищила 63%, тоді як у симуляції була закладена ймовірність 50%.

Тому, на жаль, ми не можемо з абсолютною точністю визначити, яким є справжнє значення у генеральній сукупності та чи перевищує воно 50%, незалежно від того, скільки спостережень ми проводимо. Однак, застосовуючи методи прикладної статистики, ми зможемо використати інструменти, які допоможуть ухвалити правильне рішення, зокрема й у цьому випадку.

## 1.2 Статистичні гіпотези

### 1.2.1 Постановка задачі

Ми з’ясували, що навіть за ймовірності можна отримати значну кількість успішних випадків. Насправді ми спеціально підбирали seed для отримання такого результату. Якщо повторити цей експеримент з іншим значенням seed або збільшити кількість спостережень, результат може виявитися іншим.

|  |
| --- |
| Порада |
| Спробуйте змінити seed (наприклад 22) або кількість спостережень та перевірте, як змінюється результат. |

Тож велика кількість успішних випадків може бути результатом випадковості. Щоб вирішити, чи можна вважати результати експерименту **статистично значущими** необхідно отримати відповідь на питання:

Чи можна вважати, що спостережуване значення є більшим від ?

Звернімося до теорії ймовірностей. Факт підписки на наш сервіс для кожного окремого студента можна розглядати як випадкову величину , яка підпорядковується розподілу Бернуллі[[3]](#footnote-30). Параметр цього розподілу, а саме ймовірність успіху, нам невідомий.

де — ймовірність успіху.

Нас цікавить підтвердження того, що . У статистиці для перевірки гіпотез розглядають дві можливості:

* **Нульова гіпотеза** () формулюється як твердження, яке ми прагнемо спростувати.
* **Альтернативна гіпотеза** () висловлює припущення, яке ми хочемо довести.

Скорочено це записують як:

Зауважимо, що якщо в нашому експерименті з 30 студентами можна дивитися не на частку успіхів, а на їх **кількість**.

Тоді питання можна переформулювати так:

За умови вірності наскільки ймовірно отримати 20 або більше успішних випадків з 30?

Якщо ми проводимо незалежних спостережень, то сума цих випадкових величин також підпорядковується біноміальному розподілу[[4]](#footnote-31).

де — випадкова величина, яка показує успіх у -му спостереженні, — кількість успішних випадків у спостереженнях, — кількість спостережень, — ймовірність успіху.

Давайте подивимось, як це виглядає графічно. Для цього побудуємо графік функції щільності ймовірностей для біноміального розподілу з параметрами та .

|  |
| --- |
| Лістинг 1.2: Функція щільності ймовірностей для біноміального розподілу  n = 30 mu = 0.5  x = np.arange(0, n + 1) y = binom.pmf(x, n, mu)  plt.bar(x, y, color=turquoise) plt.bar(x[x >= 20], y[x >= 20], color=red\_pink) plt.xlabel("Кількість успішних випадків") plt.ylabel("Ймовірність") plt.show()  Рядок 1  Кількість студентів.  Рядок 2  Ймовірність успіху.  Рядок 4  Створюємо масив з усіма можливими значеннями кількості успішних випадків.  Рядок 5  Обчислюємо ймовірності для кожної кількості успішних випадків.  Рядок 7  Створюємо гістограму з ймовірностями.  Рядок 8  Виділяємо ймовірності для кількості успішних випадків, які є більшими або рівними 20. |

|  |
| --- |
| Рисунок 1.1: Візуалізація функції щільності ймовірностей для біноміального розподілу |

Ціановим[[5]](#footnote-38) кольором позначено ймовірності для кожної кількості успішних випадків. Рожевими виділено ймовірності для кількості успішних випадків, яка перевищує або дорівнює 20.

### 1.2.2 Критерій

Щойно ми розробили алгоритм, який на основі вибірки або визнає наявність доказів на користь , або повідомляє, що таких доказів немає. Відповідно, він або відхиляє , або не відхиляє її.

Такий алгоритм називається **критерієм**. Його можна подати у вигляді функції , яка приймає реалізацію вибірки та повертає , якщо слід відхилити , та в іншому випадку.

Давайте припустимо, що ми вирішили відхилити , якщо кількість успішних випадків перевищує або дорівнює 21. Тоді критерій набуде вигляду:

Зазвичай скорочують запис і пишуть просто правило, за яким відхиляємо

Позначимо , , тоді критерій набуває вигляду:

Так влаштована більшість класичних критеріїв у прикладній статистиці, тому величинам у ньому дано спеціальні назви. називається **статистикою критерію**, — **критичним значенням**.

може бути будь-якою функцією від вибірки, яку ви вважаєте логічною для перевірки гіпотези. У нашому випадку це кількість успіхів, або сума всіх . Але ви можете вибрати й інші: максимальне значення, суму перших 5 значень або навіть просто перший елемент.

### 1.2.3 Критична область

Знову перепишемо наше основне запитання, тільки тепер з використанням нашого критерію :

Наскільки часто може бути таке, що за справедливості критерій відхиляє гіпотезу?

Відповідь на це запитання залежить від критичного значення. Зараз ми взяли його рівним 21, побачивши на картинці, що великі відхилення відбуваються при рідко. Але що означає рідко й наскільки рідко, не сказали. Тепер наша мета зрозуміти, як вибрати критичне значення , виходячи з **частоти помилок** нашого критерію.

Вибираючи , ми можемо або часто відхиляти нульову гіпотезу, коли мале, або можемо робити це рідше, коли велике. Щоб вибрати правильне значення, потрібно визначитися, коли наш критерій помиляється.

* . Якщо відхиляти гіпотезу при отриманні хоча б 16 успішних підписок із 30, то це навряд чи влаштує інвесторів. Так, успіхів більше половини. Але якщо в генеральній сукупності ймовірність 0.5, то майже в половині випадків ми будемо відхиляти гіпотезу. Критерій помилково повертає , тобто це помилка **хибно позитивна** (false positive, **FP**).
* . У такому разі будемо відхиляти гіпотезу тільки за 29 або 30 успіхів. Ці значення, звісно, говорять про те, що відхилення від 50% успіхів сильне. Але якщо в генеральній сукупності ймовірність, наприклад, 60%, то такі значення будуть виходити рідко. Але ж такі ймовірності теж влаштували б інвесторів, й ми б змогли відкрити стартап! А з таким критерієм ми навряд чи доб’ємося цього. Не відхилити гіпотезу , коли вона неправильна — це теж помилка. Вона називається **хибно негативна** (false negative, **FN**), оскільки критерій повернув 0 помилково.

У нашому завданні інвесторам важливіше хибно позитивна помилка. Їм дуже не хочеться потрапити в ситуацію, коли їм показали доказ успішності бізнесу, а виявилося, більшість користувачів відмовляється оформлювати підписку й компанія не отримує прибуток. Це призведе до збитків. Хибно негативна помилка призведе до того, що ви втратите успішний бізнес, але інвестори грошей не втратять.

Тому виберемо поріг, щоб ймовірність хибно позитивної помилки була задовільною, або ж **частота хибнопозитивних спрацьовувань** (False Positive Rate, FPR). Для цього треба зрозуміти, як часто ми будемо відхиляти гіпотезу, за умови вірності .

Тепер знову переформулюємо основне питання, повністю з використанням нових термінів, й врешті-решт відповімо на нього.

Який FPR у критерію для перевірки гіпотези проти ?

Коли є вірною, щоб порахувати кількість успіхів ми проводили 30 разів підкидання монетки з ймовірністю орла . Кількість орлів (тобто успіхів) у такому експерименті має розподіл, який називається біноміальним, тобто при наша статистика має біноміальний розподіл .

Обчислимо FPR для

Це вже ймовірність події за конкретного розподілу випадкової величини. Його можна подивитися за таблицею або, що зручніше, обчислити з використанням мов програмування.

### 1.2.4 Обчислення FPR

Давайте порахуємо суму ймовірностей для кількостей успіхів від 21 до 30 включно. Покажемо графічно, як це виглядає на [Рисунку 1.2](#fig-binom-pmf-fpr).

x = np.arange(0, n + 1)  
y = binom.pmf(x, n, 0.5)  
  
plt.bar(x, y, color=turquoise)  
plt.bar(x[x >= crit\_subs], y[x >= crit\_subs], color=red\_pink)  
for i in range(crit\_subs - 2, crit\_subs + 4):  
 plt.text(i + 0.5, y[i] + 0.001, f"{round(y[i] \* 100, 1)}%",  
 ha='center', va='bottom', size=8, rotation = 30)  
plt.xlabel("Кількість успішних випадків")  
plt.ylabel("Ймовірність")  
plt.show()

|  |
| --- |
| Рисунок 1.2: Ймовірність хибно відхилити за умови її вірності |

Залишається лише обчислити суму ймовірностей для кількостей успіхів від 21 до 30 включно. Це і буде нашим FPR.

У нашому випадку це буде 2.1%. Якщо FPR не перевищує деякої константи , то критерій називається критерієм **рівня значущості** . Статистичний критерій з = 100% створити тривіально — достатньо завжди відхиляти — тому така постановка не має сенсу.

Рівень значущості зазвичай обирають на основі бізнес-міркувань. Він позначає те, який ризик неправильного прийняття позитивного рішення ми вважаємо прийнятним. Зазвичай беруть , але якщо потрібне більш точне ухвалення рішення, можуть вибрати , , . Якщо ж рішення не таке критичне, можуть вибрати .

Припустимо, вибрали значення , скористаємося критерієм : тобто якщо кількість успішних випадків перевищує або дорівнює 21, то відхиляємо .

Якщо уважно подивитись на [Рисунок 1.2](#fig-binom-pmf-fpr), то можна помітити, що ми можемо відхиляти при кількості успіхів від 20, а не 21, оскільки такий все ще буде відповідати :

Якщо ж обрати 19, то FPR буде більше :

## 1.3 Статистичні функції в Python

У цій частині подивимося, як вивести те, що ми отримали в частині 2, за допомогою Python. А також зрозуміємо, як знайти відповідне за допомогою Python.

### 1.3.1 Біноміальний розподіл

Ми з’ясували, що статистика має біноміальний розподіл.

Біноміальний розподіл — розподіл кількості успіхів у послідовності з незалежних випадкових експериментів, ймовірність успіху в кожному з яких дорівнює .

Щоб працювати з розподілом, можна створити об’єкт-розподіл за допомогою бібліотеки scipy.stats.

|  |
| --- |
| Лістинг 1.3: Створення біноміального розподілу  from scipy.stats import binom  n = 30 mu = 0.5  binom\_dist = binom(n, mu) |

1. Кількість спостережень.
2. Ймовірність успіху.

### 1.3.2 Функція ймовірностей

Функція ймовірності дискретного розподілу — ймовірність, з якою приймає значення .

У Python це функція pmf (probability mass function).

|  |
| --- |
| Лістинг 1.4: Обчислення ймовірностей біноміального розподілу для кількості успіхів  binom\_dist.pmf(20) |

0.027981600724160654

Зобразимо розподіл статистики за справедливості на графіку. Для цього можна передати відразу масив точок, для яких треба розрахувати ймовірність.

x = np.arange(0, n + 1)  
y = binom\_dist.pmf(x)  
  
crit\_subs = 21  
  
plt.bar(x, y, color=turquoise, label="Ймовірність успіхів")  
plt.bar(x[x >= crit\_subs], y[x >= crit\_subs], color=red\_pink, label="Критичне значення")  
plt.xlabel("Кількість успішних випадків")  
plt.ylabel("Ймовірність")  
plt.show()

Рядок 1

Масив точок.

Рядок 2

Розрахунок ймовірностей.

Рядок 4

Критичне значення.

Рядок 6

Ймовірність успіхів.

Рядок 7

Критичне значення.

|  |
| --- |
| Рисунок 1.3: Функція щільності ймовірностей біноміального розподілу |

Насправді вже зараз ми можемо порахувати ймовірність потрапляння в критичну область. Потрібно просто підсумувати ймовірності для кількостей успіхів від 21 до 30.

|  |
| --- |
| Лістинг 1.5: Обчислення ймовірностей для критичної області  np.round(np.sum(y[crit\_subs:]), 4) |

0.0214

Отже, ми дійсно побудували критерій рівня значущості . Ба більше, це критерій рівня значущості 0.021.

А що якби ми взяли ?

|  |
| --- |
| Лістинг 1.6: Обчислення ймовірностей для критичної області при  crit\_subs = 19 np.round(np.sum(y[crit\_subs:]), 4) |

0.1002

Тоді ймовірність помилки вже навіть більше , що зовсім нам не підходить.

А якщо ?

|  |
| --- |
| Лістинг 1.7: Обчислення ймовірностей для критичної області при  crit\_subs = 20 np.round(np.sum(y[crit\_subs:]), 4) |

0.0494

Видно, що немає такого , щоб FPR був рівно .

### 1.3.3 Кумулятивна функція розподілу

Кумулятивна функція розподілу

У Python це функція cdf (Cumulative Distribution Function).

|  |
| --- |
| Лістинг 1.8: Обчислення кумулятивної функції розподілу біноміального розподілу  binom\_dist.cdf(19) |

0.9506314266473055

Ймовірність отримати або менше успіхів у нашому завданні . А оскільки , можемо обчислити рівень значущості нашого критерію.

|  |
| --- |
| Лістинг 1.9: Обчислення рівня значущості критерію  1 - binom\_dist.cdf(19) |

0.04936857335269451

### 1.3.4 Квантиль

Щоб вибрати критичну область для критерію, ми хотіли б знайти точку, площа стовпців праворуч від якої була б . Тобто площа стовпців зліва — . Така точка називається *квантилью*.

Але при й нашому біноміальному розподілі, такої точки немає. Ми з’ясували, що є точка, праворуч від якої площа , а в наступної вже . Щоб визначити квантиль у цьому випадку, модифікуємо визначення. Квантиль — величина, яку не перевищує з імовірністю хоча б . Тобто .

**Приклад 1.1** Для величини порахуємо 0.95-квантиль. Вирішимо задачу просто підбором.

Бачимо, що 18 нам ще не підходить, а 19 й більші значення вже підійдуть. У них функція розподілу буде більшою за . Відповідь — найменше відповідне значення, тобто 19. При цьому немає точки, де функція розподілу дорівнювала б в точності.

Якби розподіл був неперервним, можна було б сказати, що квантиль — це таке , на якому функція розподілу дорівнює . Але для дискретного розподілу такого може не бути.

У Python квантиль можна порахувати через функцію ppf (Percent Point Function).

|  |
| --- |
| Лістинг 1.10: Обчислення квантилю біноміального розподілу  binom\_dist.ppf(0.95) |

19.0

Як тепер підібрати для будь-яких й для будь-якого рівня значущості ?

1. Потрібно знайти , таке що
2. Тобто потрібно
3. приймає тільки цілі значення: , або
4. Отже, з визначення квантилі,
5. Значить

|  |
| --- |
| Лістинг 1.11: Знаходження критичного значення для критерію  def find\_crit\_subs(n, mu, alpha):  binom\_dist = binom(n, mu)  return binom\_dist.ppf(1 - alpha) + 1   find\_crit\_subs(30, 0.5, 0.05) |

20.0

Критичне значення , отже підсумковий критерій має такий вигляд

, значить гіпотезу ми не відкидаємо.

При цьому нам вдалося побудувати процес, за яким ми ухвалюємо рішення для будь-якого рівня значущості та значення статистики критерію.

## 1.4 -значення

Зауважимо, що зараз, якщо нам зададуть іншу , нам доведеться перебудовувати критерій заново. Це не зовсім зручно. У статистиці є механізм *-значення*, який дає змогу прийняти рішення для всіх відразу.

### 1.4.1 Більш екстремальні значення

Припустимо, ми провели експеримент й порахували для критерію його статистику . Позначимо отримане значення , у поточній задачі це . Якби кількість успішних підписок була більшою, це б сильніше свідчило на користь альтернативної гіпотези . Тобто в разі значення ми були б ще сильніше впевнені в тому, що наш бізнес буде окупатися. Тоді значення називається *більш екстремальним*, ніж значення . У нашій задачі більш екстремальним із двох значень є те, яке більше.

Визначимо поняття екстремальності формально:

Найчастіше критерії інших видів можна привести до цього, тоді для них теж визначено поняття екстремальності.

### 1.4.2 -значення

**p-value** — це ймовірність отримати таке або більш екстремальне значення статистики за умови вірності .

|  |
| --- |
| Рисунок 1.4: -значення для критерію |

Тепер виведемо формулу через функції Python:

Зобразимо на графіку область більш екстремальних значень й p-value для різних значень статистики.

|  |
| --- |
| Рисунок 1.5: -значення для критерію |

Можна побачити, що в критичній області -значення , а поза нею -значення . Саме таке правило й використовується для прийняття рішення.

Причому за -значення одразу видно, що якби в нашу критичну область включили значення , наш критерій допускав би FPR у випадків, що вже неприпустимо. Тому й гіпотезу ми не відкидаємо.

Зауважимо, що для обчислення -значення не знадобилося знання , а потрібна була тільки статистика й форма критерію.

## 1.5 Двосторонні критерії

До цього моменту нас цікавили відхилення від ймовірності в тільки в один бік. І логічно, адже це продиктовано бізнесом. Тільки велика частка успішних підписок призведе до успіху. І зазвичай при прийнятті рішень так й буває. **При тестуванні нового рішення або продукту розглядають альтернативну гіпотезу тільки в бік поліпшення**, тому що в іншому разі немає сенсу впроваджувати рішення на всіх користувачів.

Однак **іноді** може знадобитися доводити відхилення в обидва боки, якщо ви перевіряєте якесь припущення. Нехай вам дали монетку й просять перевірити, чесна вона чи ні. Монетка чесна, якщо під час підкидання ймовірність випадання орла дорівнює . Ви підкидаєте монетку разів, кожен кидок — бернуллівська величина, аналогічно завданню з сервісом освітніх послуг. Нульова гіпотеза та ж сама: . Але тепер ми хочемо відкидати цю гіпотезу як у разі великої ймовірності орла, так і в разі маленької, відповідно перевіряємо *двосторонню гіпотезу*.

Виберемо критичну область для критерію за такої альтернативи. Скористаємося тією ж статистикою . Тільки тепер відхилення в кожну сторону однаково важливі. Відкидати гіпотезу будемо не тільки на досить великих значеннях, а й на досить маленьких. Наприклад, якщо у нас було всього орла з — це свідчення на користь того, що , але не на користь .

Оскільки відхилення в різні боки однаково важливі, а розподіл симетричний, шукати критерій можна в такому вигляді:

### 1.5.1 Як вибрати критичну область

Подивимося, який вигляд матиме критична область у такому разі.

|  |
| --- |
| Рисунок 1.6: Двостороння критична область для критерію |

З картинки видно, що якщо тепер відкидати відхилення за , то необхідно відкидати й , а отже, загальна площа стовпців буде вже приблизно . Тому за рівня значущості й успіхів гіпотеза вже не відкинеться.

Якщо ж виставити , то така область уже підходить, площа стовпців .

Щоб вибрати порогову константу за формулою, можна помітити, що критична область симетрична, а значить праворуч площа не повинна бути більшою, ніж . А таку задачу ми вже вміємо розв’язувати.

Реалізуємо функцію на Python.

|  |
| --- |
| Лістинг 1.12: Знаходження критичного значення для двостороннього критерію  def find\_crit\_subs\_two\_sided(n, mu, alpha):  binom\_dist = binom(n, mu)  return n / 2 - binom\_dist.ppf(alpha / 2) + 1  find\_crit\_subs\_two\_sided(30, 0.5, 0.05) |

6.0

### 1.5.2 Як знайти -значення

Критерій має вигляд

Позначимо відхилення суми від 15 як , тоді ми маємо критерій

Тобто більш екстремальними вважатимуться ті значення суми, що знаходяться далі від 15. Щоб обчислити -значення, доведеться порахувати суму площ із двох сторін окремо.

|  |
| --- |
| Лістинг 1.13: Обчислення -значення для двостороннього критерію  def pvalue\_two\_sided\_sym(n, q):  binom\_h0 = binom(n=n, p=0.5)  diff = np.abs(q - 15)  right\_sq = 1 - binom\_h0.cdf(15 + diff - 1)  left\_sq = binom\_h0.cdf(15 - diff)  return left\_sq + right\_sq  pvalue\_two\_sided\_sym(30, 21) |

0.04277394525706769

Насправді через симетричність розподілу ліва і права площа виходять однаковими, тому можна порахувати площу з одного боку і помножити на 2.

|  |
| --- |
| Лістинг 1.14: Обчислення -значення для двостороннього критерію (спрощено)  def pvalue\_two\_sided\_sym\_simple(n, q):  binom\_h0 = binom(n=n, p=0.5)  diff = np.abs(q - 15)  right\_sq = 1 - binom\_h0.cdf(15 + diff - 1)  return 2 \* right\_sq  pvalue\_two\_sided\_sym\_simple(30, 21) |

0.04277394525706768

Тепер навіть у разі орлів -значення , тому відкидати будемо значення, починаючи з й менші або такі, що дорівнюють .

### 1.5.3 Випадок із несиметричним розподілом

Коли розподіл за справедливості несиметричний, відхилення від очікуваного значення в різні боки можуть бути по-різному критичними. Як приклад розглянемо також біноміальний розподіл, але з імовірністю успіху .

Тоді можна ліву і праву критичні області побудувати окремо, виділивши на них по площі. Праву область ми вже вміємо шукати, знайдемо ліву.

binom\_h0\_nonsym = binom(n=30, p=0.8)  
  
probs = binom\_h0\_nonsym.pmf(np.arange(31))  
  
plt.bar(np.arange(31), probs, color=turquoise, label="Binom(30, 0.8)")  
plt.legend(fontsize=8)  
plt.show()

|  |
| --- |
| Рисунок 1.7: Біноміальний розподіл з імовірністю успіху |

Для того, щоб побудувати двосторонній критерій, потрібно знайти ліворуч і праворуч області, площа яких становить не більше, ніж . Для правого боку ми вже розв’язували таку задачу, розв’яжемо для лівого.

Шукаємо , таке що

Спочатку знайдемо перше число, де ймовірність . А це за визначенням -квантиль. Достатньо взяти попереднє число, і воно буде задовольняти нашій умові.

|  |
| --- |
| Лістинг 1.15: Знаходження критичного значення для двостороннього критерію  def two\_sided\_criterion\_nonsym(n, mu, alpha):  binom\_h0 = binom(n=n, p=mu)  c2 = binom\_h0.ppf(1 - alpha/2) + 1  c1 = binom\_h0.ppf(alpha/2) - 1  return c1, c2  two\_sided\_criterion\_nonsym(30, 0.8, 0.05) |

(18.0, 29.0)

Отже, наш критерій для перевірки гіпотези

має вигляд

Тут межа уже має логічний вигляд, бо треба спростувати 80% орлів/успіхів, а для цього потрібна велика їхня кількість.

Зобразимо критичну область на графіку.

C1, C2 = two\_sided\_criterion\_nonsym(30, 0.8, 0.05)  
  
plt.figure(figsize=(6, 4))  
plt.bar(np.arange(31), probs, color=turquoise, label="Binom(30, 0.8)")  
plt.bar(np.arange(31)[np.arange(31) <= C1], probs[np.arange(31) <= C1], color=red\_pink, label="Критичне значення")  
plt.bar(np.arange(31)[np.arange(31) >= C2], probs[np.arange(31) >= C2], color=red\_pink)  
plt.xlabel("Кількість успішних випадків")  
plt.ylabel("Ймовірність")  
plt.legend(fontsize = '8', loc = 'upper left')  
plt.show()

|  |
| --- |
| Рисунок 1.8: Двостороння критична область для критерію |

### 1.5.4 -значення для несиметричного розподілу

Цей критерій — об’єднання двох критеріїв рівня значущості , для кожного з яких можна порахувати -значення. Позначимо їх як . Перший критерій відкидається при , другий при . А наш об’єднаний, коли виконано одну з цих умов, тобто

Отже, можна рахувати -значення як й порівнювати з .

Проведемо аналогію із симетричним випадком: якщо сума опинилася в лівій частині, то потрібно порахувати -значення лівого критерію і помножити на 2. Якщо сума опинилася в правій частині, то потрібно порахувати -значення правого критерію і помножити на 2.

|  |
| --- |
| Лістинг 1.16: Обчислення -значення для двостороннього критерію з несиметричним розподілом  def pvalue\_two\_sided(n, q, mu=0.5):  binom\_h0 = binom(n=n, p=mu)  pvalue\_left = binom\_h0.cdf(q)  pvalue\_right = 1 - binom\_h0.cdf(q - 1)  return 2 \* min(pvalue\_left, pvalue\_right)  pvalue\_two\_sided(30, 28, 0.8) |

0.08835797030399428

Видно, що -значення , отже, на рівні значущості навіть успіхів недостатньо, щоб відкинути ймовірність успіху в .

Зауважимо, що ця ж функція працює і для симетричного випадку, повертаючи той самий результат.

|  |
| --- |
| Лістинг 1.17: Обчислення -значення для двостороннього критерію з симетричним розподілом  pvalue\_two\_sided(n=30, q=20, mu=0.5) |

0.09873714670538902

|  |
| --- |
| Лістинг 1.18: Знаходження -значення для двостороннього критерію  pvalue\_two\_sided\_sym(n=30, q=20) |

0.09873714670538904

## 1.6 Готові функції

Звісно, можна використати готові функції з бібліотеки scipy. Для цього використаємо функцію binomtest, котра має параметри:

* k — кількість успіхів
* n — кількість спостережень
* p — ймовірність успіху
* alternative — тип гіпотези:
  + two-sided: двостороння
  + greater: правостороння
  + less: лівостороння

|  |
| --- |
| Лістинг 1.19  from scipy.stats import binomtest  result = binomtest(19, 30, 0.5, alternative='two-sided')  print(f"Статистика: {result.statistic:.2f}") print(f"p-значення: {result.pvalue:.4f}")  Статистика: 0.63 p-значення: 0.2005 |

## 1.7 Висновки

Ми розглянули, як можна використовувати біноміальний розподіл для перевірки гіпотези про ймовірність успіху. Для цього ми визначили критерій, критичну область, -значення. Показали, як можна використовувати ці поняття для різних видів гіпотез: односторонніх, двосторонніх, симетричних та несиметричних.

## 1.8 Питання для самоперевірки

1. Які гіпотези можна перевірити за допомогою біноміального розподілу?
2. Як визначити критичну область для критерію?
3. Як визначити -значення для критерію?
4. Як визначити критичну область для двостороннього критерію?
5. Як визначити -значення для двостороннього критерію?
6. Як визначити критичну область для несиметричного розподілу?
7. Як визначити -значення для несиметричного розподілу?

# 2. Статистична потужність, ефект та довірчі інтервали

## 2.1 Статистична потужність

### 2.1.1 Хибно негативні помилки

Раніше під час побудови критеріїв ми звертали увагу тільки на , рівень значущості критерію. Але цей параметр контролює лише хибнопозитивну помилку (False Positive), а саме ймовірність, що критерій прийме за умови вірності .

Але є ще один вид помилок, які може допустити критерій — хибно негативні помилки (False Negative). Це випадки, коли критерій приймає за умови вірності . Це важливо, оскільки вони можуть вказувати на те, що критерій не чутливий до змін, які відбуваються в даних.

Випадок, коли ймовірність FPR , але при цьому ймовірність хибно негативні помилки (False Negative Rate, FNR) величезна, можна навести легко. Для цього достатньо **ніколи** не відкидати гіпотезу, взявши критерій .

Наведемо приклад, коли помилки False Negative відбуваються не завжди, але критерії є все одно нечутливими.

### 2.1.2 Критерій пори року

Поставимо гіпотезу про те, що зараз на вулиці літо. Для перевірки можна було б, звісно, подивитися в календар, але ми зробимо інакше.

Подивимося у вікно і визначимо, чи йде там сніг. Якщо він йде, то це непоганий доказ того, що зараз не літо, а отже можна відкинути .

Порахуємо FPR та FNR для цього критерію. Ми знаємо, що влітку сніг іде дуже рідко (ймовірність помилки нижча за ), тож це точно критерій рівня значущості , чого зазвичай достатньо для критеріїв.

Але що з FNR? Розглянемо конкретний випадок: зараз вересень. Оскільки у вересні майже завжди немає снігу, можна сказати, що FNR більша за , отже, цей критерій насправді мало дієвий.

Сформулюємо інший критерій рівня значущості , причому в цьому разі рівень значущості можна вибрати довільним.

Виходить, цей критерій випадковий, і він не використовує взагалі жодної інформації про погоду. Однак вимогам до рівня значущості він задовольняє.

Обчислимо FNR.

За , як у першому випадку, отримуємо ймовірність FNR , тобто за однакового рівня значущості з першим критерієм, другий критерій частіше припускається хибно негативної помилки.

### 2.1.3 Потужність

У статистиці заведено позитивним результатом вважати відкидання нульової гіпотези, бо зазвичай підтвердження альтернативи означає наявність бізнес-результату. Тому вважається хорошим критерій, який частіше дає змогу виявити бізнес-результат. І рахують тоді не ймовірність хибно негативної помилки, а *потужність*, що дорівнює ймовірності відкинути нульову гіпотезу за вірності , тобто ймовірність **істинно позитивного результату** (True Positive Rate, TPR).

Коли альтернатива складається з множини результатів, потужність розглядають як функцію від результату. Наприклад, можна порахувати потужність першого та другого критеріїв взимку й восени.

**Перший критерій**

**Другий критерій**

Зазвичай завдання пошуку найкращого критерію формулюється як пошук якомога потужнішого критерію за заданого рівня значущості . Але ми сказали, що потужність — функція від параметра, у нашому випадку від місяця.

Якщо ми застосовуватимемо критерій у січні, то потужнішим буде перший критерій, а якщо в травні, то потужнішим буде другий критерій. Тому потрібно розуміти, коли буде застосовуватися критерій, а отже, ми шукаємо найпотужніший критерій у галузі, яка нас цікавить.

Хоча в реальності в травні потужність обох критеріїв настільки низька, що вони просто не приносять користі, й використовувати їх не має сенсу.

## 2.2 Потужність для біноміального розподілу

Застосуємо нові знання про потужність для нашої задачі з освітнім сервісом. З бізнес-міркувань ми вже вибрали , а отже, знаємо, що ми неправильно відкидаємо гіпотезу з ймовірністю не більше, ніж . Тобто цим обмежена ймовірність хибно позитивної помилки.

А з якою ймовірністю ми будемо *правильно* відкидати гіпотезу? І яка в нас буде ймовірність хибно негативної помилки? На це запитання якраз відповість формула потужності.

Згадаймо критерій, за яким ми приймаємо рішення:

Тобто якщо отримуємо хоча б успішних підписок, то відкидаємо .

Зауважимо, що потужність залежить від того, яке значення у нашій генеральній сукупності. Зафіксуємо спочатку параметр й порахуємо потужність для нього. Якщо істинний параметр такий, то статистика має розподіл .

binom\_h0 = binom(n=30, p=0.5)  
binom\_alternative = binom(n=30, p=0.6)  
  
x\_grid = np.arange(1, 31)  
crit\_reg = x\_grid >= 20  
  
probs\_h0 = binom\_h0.pmf(x\_grid)  
plt.bar(x\_grid, probs\_h0, color=turquoise, label='PMF, $Binom(0.5, 30)$')  
  
probs\_alternative = binom\_alternative.pmf(x\_grid)  
plt.bar(x\_grid, probs\_alternative, color=slate, label='PMF, $Binom(0.6, 30)$')  
plt.bar(x\_grid[crit\_reg], probs\_alternative[crit\_reg], color=red\_pink, label='Критична область')  
  
plt.legend(fontsize=8)  
plt.show()

|  |
| --- |
| Рисунок 2.1: Потужність критерію для |

Як і раніше, нас цікавить імовірність отримати або більше успіхів. Але якщо раніше ми дивилися на неї для розподілу з й хотіли, щоб вона була меншою за , то тепер ми дивимося за та прагнемо зробити цю величину якомога більшою. Порівняно з обчисленням FPR формула не зміниться, змінюється тільки

|  |
| --- |
| Лістинг 2.1: Обчислення потужності критерію  critical\_value = 20 power = 1 - binom(n=30, p=0.6).cdf(critical\_value - 1) fpr = 1 - binom(n=30, p=0.5).cdf(critical\_value - 1)  print(f"Хибно позитивна помилка: {fpr:.1%}") print(f"Потужність: {power:.1%}") |

Хибно позитивна помилка: 4.9%  
Потужність: 29.1%

Видно, що потужність близько . Це досить маленьке значення, адже якщо наш продукт прибутковий, то ми побачимо це за допомогою нашого тесту тільки з імовірністю в відсотків. Ми легко можемо пропустити ефект.

Що ж можна зробити, щоб зробити потужність вищою? Щоб розібратися, реалізуємо функцію потужності в загальному вигляді.

|  |
| --- |
| Лістинг 2.2: Обчислення потужності критерію для загального випадку  def get\_stat\_power(N, mu\_h0, mu\_alternative, alpha):  '''Обчислює статистичну потужність критерію для біноміального розподілу    Параметри:  N - кількість бернуллієвських експериментів (розмір вибірки)  mu\_h0 - імовірність успіху в нульовій гіпотезі  mu\_alternative - передбачувана ймовірність успіху в експерименті  alpha - рівень значущості критерію  '''  binom\_h0 = binom(n=N, p=mu\_h0)  binom\_alternative = binom(n=N, p=mu\_alternative)  critical\_value = binom\_h0.ppf(1 - alpha) + 1  return 1 - binom\_alternative.cdf(critical\_value - 1)   get\_stat\_power(30, 0.5, 0.6, alpha=0.05) |

0.2914718612234968

Коли в житті ми спостерігаємо якесь явище і бачимо його лише кілька разів, ми не впевнені в тому, що воно не випадкове. Якщо ж бачимо його досить часто, то вже складаємо закономірності. Так і в статистиці. Коли ми подивилися на 30 потенційних підписок, ми помічаємо, що частка доставок більше половини. Але ми все ще не впевнені. Щоб отримати більше впевненості, потрібно провести більше спостережень, тобто знайти більше пробних клієнтів.

Подивимося, що буде, якщо ми проведемо експеримент на 300 клієнтах.

|  |
| --- |
| Лістинг 2.3: Обчислення потужності критерію для 300 клієнтів  get\_stat\_power(300, 0.5, 0.6, alpha=0.05) |

0.9655326717180749

Бачимо, що потужність уже дуже близька до . Але провести 300 пробних занять набагато затратніше, ніж 30. І за ресурсами, і за часом. Тому зазвичай балансують між потужністю і тривалістю/витратами експерименту.

Прийнято вважати, що прийнятною для роботи потужністю вважається . Подивимося, як змінюється потужність при зростанні розміру вибірки, і скільки потрібно провести експериментів, щоб детектувати ефект при у випадків.

n\_grid = np.arange(10, 600, 10)  
power = get\_stat\_power(n\_grid, 0.5, 0.6, alpha=0.05)  
  
plt.xlabel('Кількість пробних занять')  
plt.ylabel('Потжність')  
  
plt.plot(n\_grid, power, color=turquoise)  
plt.axhline(0.8, ls='--', color=red\_pink, label='Потужність = 80%')  
  
min\_n = n\_grid[power >= 0.8].min()  
plt.axvline(min\_n, ls='--', color=slate, label=f'N = {min\_n}')  
  
plt.legend()  
plt.show()

|  |
| --- |
| Рисунок 2.2: Залежність потужності від розміру вибірки для |

Бачимо, що для потужності в достатньо набрати 160 пробних занять.

А що, якщо ми хочемо детектувати ще менший ефект? Наприклад, якщо хочемо відкидати гіпотезу за . Часто поліпшення ймовірності успіху на може бути значущим для продукту, тому це питання не позбавлене сенсу.

n\_grid = np.arange(10, 30000, 59)  
power = get\_stat\_power(n\_grid, 0.5, 0.51, alpha=0.05)  
  
plt.xlabel('Кількість пробних занять', fontsize=8)  
plt.ylabel('Потжність', fontsize=8)  
  
plt.plot(n\_grid, power, color=turquoise)  
plt.axhline(0.8, ls='--', color=red\_pink, label='Потужність = 80%')  
  
min\_n = n\_grid[power >= 0.8].min()  
plt.axvline(min\_n, ls='--', color=slate, label=f'N = {min\_n}')  
  
plt.legend()  
plt.show()

|  |
| --- |
| Рисунок 2.3: Залежність потужності від розміру вибірки для |

Бачимо, що потрібно понад 15 тисяч клієнтів, щоб детектувати такий ефект! Дуже складно знайти стільки пробних клієнтів. Але потрібно замислитися над питанням, а чи варто це робити? У нашому випадку, якщо ймовірність успіху , то прибуток із замовлень буде невеликий, і вкладення інвесторів, звісно, окупатимуться, але дуже довго. Тому збільшення на для нашого завдання не значуще *практично*, а отже, не потрібно намагатися набирати 15 тисяч людей, а можна зупинитися і на 160.

Перед кожним експериментом аналітику варто замислюватися над питанням тривалості тесту і кількості учасників. Для цього потрібно зрозуміти:

* Який ефект є для завдання практично значущим?
* Скільки знадобиться випробовуваних, щоб детектувати цей ефект частіше, ніж у випадків?

З графіків видно, що для детектування меншого ефекту потрібен більший розмір вибірки. Подивимося, як для фіксованого змінюється потужність для різних параметрів .

mu\_grid = np.linspace(0.5, 0.9, 100)  
power = get\_stat\_power(30, 0.5, mu\_grid, alpha=0.05)  
  
plt.xlabel('Ймовірність успіху')  
plt.ylabel('Потужність')  
  
plt.plot(mu\_grid, power, color=turquoise)  
plt.axhline(0.8, ls='--', color=red\_pink, label='Потужність = 80%')  
  
min\_mu = mu\_grid[power >= 0.8].min()  
plt.axvline(min\_mu, ls='--', color=slate, label=f'$\mu = {min\_mu:.2f}$')  
  
plt.legend()  
plt.show()

|  |
| --- |
| Рисунок 2.4: Залежність потужності від параметра |

У нашому експерименті ми добре детектуємо ефект, тільки якщо ймовірність успіху в генеральній сукупності хоча б .

## 2.3 Мінімальна величина ефекту

Вище на [Рисунок 2.4](#fig-binom-power-mu) ми побачили, що з хорошою потужністю понад ми можемо помітити ефект у процентних пункти. Причому це можна порахувати навіть до проведення експерименту. У нашому випадку таке збільшення успішності щодо цілком можливо, і з ним можна працювати. Але коли аналітики перевіряють зміни, найчастіше очікуваний ефект коливається в районі одного, максимум двох відсотків! Для подібних змін не підійде обрана постановка експерименту, а значить і проводити його не має сенсу.

Тому перед запуском експериментів аналітики повідомляють **мінімальну величину ефекту, яку можна задетектувати** (Minimal Detectable Effect, MDE). У нашому випадку процентних пункти.

Більш формально, MDE для гіпотези — це мінімальний ефект , за якого критерій рівня значущості для перевірки цієї гіпотези за істинного параметра та розміру вибірки відкидатиме з потужністю більшою, ніж .

Найчастіше беруть . Напишемо функцію, яка обчислюватиме MDE підбором.

|  |
| --- |
| Лістинг 2.4: Обчислення MDE  def binom\_test\_mde\_one\_sided(N, mu0, alpha=0.05, min\_power=0.8):  delta\_grid = np.linspace(0, 1 - mu0, 500)   power = get\_stat\_power(N, mu0, mu0 + delta\_grid, alpha=alpha)  fit\_delta = delta\_grid[power >= min\_power]  return fit\_delta[0]  binom\_test\_mde\_one\_sided(30, 0.5) |

0.21843687374749496

Результат збігається з обчисленнями за графіком [Рисунок 2.4](#fig-binom-power-mu). Тобто ми можемо детектувати ефект у процентних пункти.

Зазвичай MDE розраховують не просто так, а нерозривно з ним іде питання про визначення **розміру вибірки**.

У нашому завданні ми знайшли клієнтів, не обчислюючи спочатку, скільки їх знадобиться. Але що якщо отриманий MDE занадто великий й потрібно зробити його меншим, оскільки очікувані зміни набагато менші? Тоді вирішується зворотне завдання, За необхідним MDE визначити обсяг вибірки. Якщо ми говоримо, що хочемо детектувати +10 в.п., тобто 60% успішних підписок, то потрібно знайти 160 тестових клієнтів, це видно з попередніх графіків. Якщо 30 осіб нам, наприклад, шукати місяць, такий тест може затягнутися майже на півроку. Тому варто подумати про те, щоб виділити додаткові ресурси на пошук клієнтів, наприклад, залучити маркетологів.

## 2.4 Довірчі інтервали

Раніше ми навчилися перевіряти гіпотезу . Як відповідь ми отримуємо лише вердикт “відкидаємо ” або “не відкидаємо ”. Однак у вибірці міститься набагато більше інформації, й ми можемо більше зрозуміти про параметр, ніж порівняння з числом .

Якщо гіпотеза не відкидається, це означає, що значення припустиме для нашої вибірки. Отримані значення можна пояснити значенням . Але якщо у нас є механізм перевірки для будь-якого , ми можемо для всіх значень дізнатися, які з них допустимі, і отримати множину можливих значень . Така множина називається *довірчим інтервалом*.

**Довірчий інтервал** рівня — множина значень параметра , для яких гіпотеза не відкидається критерієм рівня значущості .

З визначення випливає, що різні критерії можуть породжувати різні довірчі інтервали. У цій частині розглянемо, які інтервали породжуються двостороннім критерієм. Для цього з кроком переберемо значення і перевіримо гіпотези.

|  |
| --- |
| Лістинг 2.5: Довірчі інтервали для біноміального розподілу  def two\_sided\_criterion\_nonsym(n, mu, alpha):  binom\_h0 = binom(n=n, p=mu)  c2 = binom\_h0.ppf(1 - alpha/2) + 1  c1 = binom\_h0.ppf(alpha/2) - 1  return c1, c2   success\_cnt = 19 mu\_grid = np.arange(0, 1, 0.001) mu\_no\_rejection = []  for mu\_h0 in mu\_grid:  c1, c2 = two\_sided\_criterion\_nonsym(30, mu\_h0, alpha=0.05)  if success\_cnt > c1 and success\_cnt < c2:  mu\_no\_rejection.append(mu\_h0)  print(f'95% довірчий інтервал: [{min(mu\_no\_rejection)} - {max(mu\_no\_rejection)}]') |

95% довірчий інтервал: [0.439 - 0.8]

1. Функція, що обчислює критичні значення для двостороннього критерію.
2. Кількість успішних підписок.
3. Сітка значень .
4. Список значень , для яких гіпотеза не відкидається.
5. Перебір значень .

Отримавши такий інтервал, ми відразу можемо зробити висновок, що гіпотеза не відкидається, оскільки лежить у довірчому інтервалі. Але при цьому відразу зрозуміло, що на рівні значущості .

Звичайно ж, у довірчому інтервалі лежить значення , для якого успіхів — це найбільш правдоподібний результат. При цьому інтервал несиметричний щодо точки .

Подивимося, як можна візуально знайти межу інтервалу. Ми отримали успіхів. Для кожного статистика має розподіл . Будемо малювати цей розподіл і дивитися, чи потрапляє у критичну область.

|  |
| --- |
| Лістинг 2.6: Довірчі інтервали для біноміального розподілу  mus\_h0 = [0.2, 0.438, 0.439, 0.8, 0.81, 0.9]  fig, axes = plt.subplots(3, 2, figsize=(8, 10))  for mu\_h0, ax in zip(mus\_h0, axes.flatten()):  binom\_h0 = binom(n=30, p=mu\_h0)  probs = binom\_h0.pmf(x\_grid)   ax.bar(x\_grid, probs, color=turquoise, label=f'PMF, $Binom({mu\_h0}, 30)$')  c1, c2 = two\_sided\_criterion\_nonsym(30, mu\_h0, alpha=0.05)  crit\_reg = (x\_grid <= c1) | (x\_grid >= c2)  ax.bar(x\_grid[crit\_reg], probs[crit\_reg], color=red\_pink, label='Критична область')   is\_rejection = success\_cnt <= c1 or success\_cnt >= c2  ax.axvline(success\_cnt, ls='--', label=f'Q = {success\_cnt} ' + ('відхилено' if is\_rejection else 'не видхилено'), color='gray', alpha=0.4)   rejection\_prob = probs[crit\_reg].sum()  ax.set\_title(f'$\mu = {mu\_h0}$', fontsize=8)  ax.legend() |

|  |
| --- |
| Рисунок 2.5: Довірчі інтервали для біноміального розподілу |

Видно, що зі зростанням гістограма зсувається вправо. І спочатку потрапляє в праву критичну область. Потім, починаючи з точки , значення вже опиняється поза критичною областю, і тільки з починає потрапляти в ліву критичну область.

Таким чином, ліва межа довірчого інтервалу — це перша точка, коли значення статистики перестало потрапляти до критичної області, а права межа - остання точка, коли значення не потрапляє до правої критичної області.

## 2.5 Односторонні довірчі інтервали

Насправді, двосторонній критерій потрібен вкрай рідко. Контролювати хибно похитивну помилку нам потрібно тільки для відхилень у бік, корисний для бізнесу. У випадку завдання з освітнім сервісом це отримання *більшої* конверсії в успіх.

Спробуємо скористатися одностороннім критерієм для побудови довірчого інтервалу.

|  |
| --- |
| Лістинг 2.7: Односторонні довірчі інтервали для біноміального розподілу  def make\_binom\_criterion(n, mu=0.5, alpha=0.05):  binom\_h0 = binom(n=n, p=mu)  q = binom\_h0.ppf(1 - alpha)  return q + 1  success\_cnt = 19 mu\_grid = np.arange(0, 1.001, 0.001) mu\_no\_rejection = []  for mu\_h0 in mu\_grid:  crit\_val = make\_binom\_criterion(n=30, mu=mu\_h0, alpha=0.05)  if success\_cnt < crit\_val:  mu\_no\_rejection.append(mu\_h0)  print(f'95% довірчий інтервал: [{min(mu\_no\_rejection)} - {max(mu\_no\_rejection)}]') |

95% довірчий інтервал: [0.467 - 1.0]

Коли ми використовували двосторонній інтервал, ми отримали ліву межу . Виходить, що односторонній інтервал з точки зору лівої межі дає нам більше інформації. При цьому з точки зору правої межі ми втрачаємо інформацію зовсім. Вона дорівнює 1 просто тому, що ймовірність не може бути більшою.

Насправді зазвичай на праву межу не дивляться під час аналізу, коли ми шукаємо позитивний ефект.

Припустимо, ми отримали не успіхів, а . Побудуємо 2 види інтервалів.

|  |
| --- |
| Лістинг 2.8: Двосторонній довірчий інтервал для успіхів  success\_cnt = 22 mu\_grid = np.arange(0, 1, 0.001) mu\_no\_rejection = []  for mu\_h0 in mu\_grid:  c1, c2 = two\_sided\_criterion\_nonsym(30, mu\_h0, alpha=0.05)  if success\_cnt > c1 and success\_cnt < c2:  mu\_no\_rejection.append(mu\_h0)  print(f'Двосторонній 95% довірчий інтервал: [{min(mu\_no\_rejection):.3f} - {max(mu\_no\_rejection):.3f}]') |

Двосторонній 95% довірчий інтервал: [0.542 - 0.877]

|  |
| --- |
| Лістинг 2.9: Односторонній довірчий інтервал для успіхів  success\_cnt = 22 mu\_grid = np.arange(0, 1.001, 0.001) mu\_no\_rejection = []  for mu\_h0 in mu\_grid:  crit\_val = make\_binom\_criterion(n=30, mu=mu\_h0, alpha=0.05)  if success\_cnt < crit\_val:  mu\_no\_rejection.append(mu\_h0)  print(f'Односторонній 95% довірчий інтервал: [{min(mu\_no\_rejection):.3f} - {max(mu\_no\_rejection):.3f}]') |

Односторонній 95% довірчий інтервал: [0.571 - 1.000]

За обома довірчими інтервалами ми робимо висновок, що конверсія значимо відрізняється від . Але односторонній інтервал дає кращу нижню оцінку на ймовірність успіху. Ми можемо зрозуміти, що наша конверсія більша за . А інформація з двостороннього інтервалу про те, що ймовірність менша за не додає нам користі.

Навіщо ж ми тоді взагалі використовуємо двосторонній інтервал? Щоб це зрозуміти, подивимося, як виглядають візуально межі для одностороннього інтервалу.

|  |
| --- |
| Лістинг 2.10: Односторонній довірчий інтервал для успіхів  fig, axes = plt.subplots(3, 2, figsize=(8, 10))  for mu\_h0, ax in zip(mus\_h0, axes.flatten()):  binom\_h0 = binom(n=30, p=mu\_h0)  probs = binom\_h0.pmf(x\_grid)   ax.bar(x\_grid, probs, color=turquoise, label=f'PMF, $Binom({mu\_h0}, 30)$')  c = make\_binom\_criterion(30, mu\_h0, alpha=0.05)  crit\_reg = (x\_grid >= c)  ax.bar(x\_grid[crit\_reg], probs[crit\_reg], color=red\_pink, label='Критична область')   is\_rejection = success\_cnt >= c  ax.axvline(success\_cnt, ls='--', label=f'Q = {success\_cnt} ' + ('відхилено' if is\_rejection else 'не відхилено'), color='gray', alpha=0.4)   rejection\_prob = probs[crit\_reg].sum()  ax.set\_title(f'$\mu = {mu\_h0}$', fontsize=8)  ax.legend() |

|  |
| --- |
| Рисунок 2.6: Односторонній довірчий інтервал для успіхів |

Порівняно з [Рисунок 2.5](#fig-binom-ci) ми бачимо, що права критична область стала більшою через те, що там тепер знаходиться не , а від усіх значень. При цьому лівої критичної області просто не існує, тому за великих не відбувається потрапляння до неї, а значить ми не відкидаємо гіпотезу.

Зауважимо, що якби ми будували двосторонній інтервал, але з удвічі більшою , потрапляння в праву критичну область траплялися б за тих самих , що й в односторонньому критерії. Тому часто для пошуку односторонньої межі будують двосторонній довірчий інтервал із більшою , ігноруючи при цьому праву межу. Це зручно, оскільки можна користуватися тільки однією функцією для критерію.

Перевіримо, що вийде за .

|  |
| --- |
| Лістинг 2.11: Двосторонній довірчий інтервал для успіхів з  success\_cnt = 19 mu\_grid = np.arange(0, 1, 0.001) mu\_no\_rejection = []  for mu\_h0 in mu\_grid:  c1, c2 = two\_sided\_criterion\_nonsym(30, mu\_h0, alpha=0.1)  if success\_cnt > c1 and success\_cnt < c2:  mu\_no\_rejection.append(mu\_h0)  print(f'95% довірчий інтервал: [{min(mu\_no\_rejection):.3f} - {max(mu\_no\_rejection):.3f}]') |

95% довірчий інтервал: [0.467 - 0.778]

Бачимо, що отримали таку саму ліву межу, як і в односторонньому інтервалі.

## 2.6 Властивості довірчих інтервалів

Згадаймо визначення довірчого інтервалу.

Нехай є критерій рівня значущості для перевірки гіпотези , — статистика критерію, а — її реалізація на конкретній вибірці . Тоді **довірчим інтервалом** називається множина таких , на яких критерій не відкидає гіпотезу .

Процедура підрахунку інтервалу — це довгий перебір значень із деяким кроком. Але це все ще залишається деякою функцією від вибірки, тобто *статистикою* й випадковою величиною, причому її розподіл залежить від статистики , а отже, і від початкової вибірки, та від параметра у генеральній сукупності.

Позначимо межі інтервалу за — статистики критерію, які відповідають лівій та правій межі інтервалу.

### 2.6.1 Ймовірність попадання в інтервал

Яким би не було істинне значення , ймовірність того, що воно перебуває між та , не нижча, ніж . Значення називається **рівнем довіри** довірчого інтервалу.

Важливо, що випадковість тут прихована саме в і , а не в . Параметр невідомий, але ми припускаємо його константним і не випадковим.

Перевіримо справедливість цієї властивості. Для цього зафіксуємо й проведемо множину експериментів:

* Генеруємо вибірку з розподілу з параметром .
* Обчислюємо статистику .
* Рахуємо довірчий інтервал для .

Перевіряємо, що частка випадків, коли параметр опинився всередині інтервалу, хоча б

|  |
| --- |
| Лістинг 2.12: Перевірка властивості довірчого інтервалу.  import time  start\_time = time.time()  def my\_binomial\_confint(n, alpha, q):  mu\_grid = np.arange(0, 1.1, 0.1) # np.arange(0, 1.001, 0.001)  mu\_no\_rejection = []   for mu\_h0 in mu\_grid:  c1, c2 = two\_sided\_criterion\_nonsym(30, mu\_h0, alpha=0.05)  if q > c1 and q < c2:  mu\_no\_rejection.append(mu\_h0)   return min(mu\_no\_rejection), max(mu\_no\_rejection)  N\_EXPERIMENTS = 1000 SAMPLE\_SIZE = 30 latent\_mu = 0.5 binom\_true = binom(n=SAMPLE\_SIZE, p=latent\_mu)  confint\_fail\_cases = 0  for i in range(N\_EXPERIMENTS):  q = binom\_true.rvs()  L, R = my\_binomial\_confint(n=SAMPLE\_SIZE, alpha=0.05, q=q)  if L < latent\_mu < R:  pass  else:  confint\_fail\_cases += 1  success\_cases = round(100 \* (N\_EXPERIMENTS - confint\_fail\_cases) / N\_EXPERIMENTS, 2)  print(f"Відсоток успішних випадків: {success\_cases}%")  end\_time = time.time() print(f"Час виконання: {end\_time - start\_time:.4f} секунди")  Відсоток успішних випадків: 61.4% Час виконання: 3.7903 секунди |

Зазначимо, що цей код працював понад 5 хвилин. Це через те, що під час кожного експерименту потрібно побудувати довірчий інтервал, а значить перевірити 1000 можливих параметрів .

Бачимо, що властивість виконалася. Ми очікували хоча б влучень, отримали навіть . Насправді це значно більше, ніж ми очікували. Це відбувається через дискретність розподілу. З тієї ж причини під час пошуку критичної області ми не могли вибрати стовпці із сумарною висотою рівно .

#### 2.6.1.1 Доведення

Під час формулювання властивості ми припускаємо, що є деяка — ймовірність успіху в генеральній сукупності. Коли ми проводимо штучний експеримент, ми фіксуємо її й можемо вважати істинною .

Щоразу ми генеруємо й перевіряємо, чи потрапила у довірчий інтервал. Намалюємо розподіл статистики , який уже нам знайомий. Намалюємо й область ймовірності , як ми робили це раніше.

|  |
| --- |
| Лістинг 2.13: Розподіл статистики при істинній ймовірності успіху  mu0 = 0.5 binom\_mu0 = binom(n=30, p=mu0) probs = binom\_mu0.pmf(x\_grid)  plt.bar(x\_grid, probs, color=turquoise, label=f'PMF, $Binom({mu0}, 30)$') c1, c2 = two\_sided\_criterion\_nonsym(30, mu0, alpha=0.05) crit\_reg = (x\_grid >= c2) | (x\_grid <= c1) plt.bar(x\_grid[crit\_reg], probs[crit\_reg], color=red\_pink, label='Критична область') plt.legend() plt.show() |

|  |
| --- |
| Рисунок 2.7: Розподіл статистики при істинній ймовірності успіху |

Нехай реалізувалося значення статистики . За такою вибіркою можна побудувати довірчий інтервал на . Він буде якось розташований, але зараз нас цікавить, чи потрапить у нього . За визначенням потрапляння в інтервал відбудеться, якщо не відкидається гіпотеза . Але тоді за справедливості статистика має той розподіл, що і на малюнку. І гіпотеза відкидається тільки в разі потрапляння в критичну область, а це трапляється з ймовірністю .

Отже, з ймовірністю хоча б перебуватиме в довірчому інтервалі.

Часто так й вводять визначення довірчого інтервалу. Для вибірки — це така пара статистик і , що хоч яким би не було ,

де і — статистики, що залежать від вибірки. Знову звертаємо увагу, що випадковість тут прихована не в параметрі , а в статистиках від вибірки.

### 2.6.2 Довірчий інтервал Вілсона

Розглянутий зараз алгоритм побудови довірчого інтервалу працює занадто довго. У Python є функції, які дозволяють швидше розрахувати інтервал. Наприклад, можна скористатися методом Вілсона і функцією proportion\_confint.

Повторимо експерименти з новим типом довірчого інтервалу, тут можемо дозволити більше реалізацій вибірки, оскільки інтервал рахується недовго.

|  |
| --- |
| Лістинг 2.14: Довірчий інтервал Вілсона  from statsmodels.stats.proportion import proportion\_confint  start\_time = time.time()  N\_EXPERIMENTS = 1000 SAMPLE\_SIZE = 30 latent\_mu = 0.5 binom\_true = binom(n=SAMPLE\_SIZE, p=latent\_mu)  confint\_fail\_cases = 0  for i in range(N\_EXPERIMENTS):  q = binom\_true.rvs()   L, R = proportion\_confint(  count=q,  nobs=SAMPLE\_SIZE,  alpha=0.05,  method='wilson'  )  if L < latent\_mu < R:  pass  else:  confint\_fail\_cases += 1  success\_cases = round(100 \* (N\_EXPERIMENTS - confint\_fail\_cases) / N\_EXPERIMENTS, 2) print(f"Відсоток успішних випадків: {success\_cases}%") end\_time = time.time() print(f"Час виконання: {end\_time - start\_time:.4f} секунди") |

Відсоток успішних випадків: 96.6%  
Час виконання: 0.0714 секунди

Зауважимо, що наше може знаходитись в довірчому інтервалі менше, ніж у випадків. Це відбувається через те, що швидкі методи працюють наближено, оцінюючи розподіл статистики при збільшенні розміру вибірки. Чим розмір вибірки більший, тим ближчим буде інтервал до -ного.

Залежність частки успішних влучень у довірчий інтервал від розміру вибірки зобразимо на [Рисунок 2.8](#fig-binom-ci-2).

|  |
| --- |
| Лістинг 2.15: Залежність частки успішних влучень у довірчий інтервал від розміру вибірки  n\_grid = np.arange(1, 1000, 25).tolist() interval\_success\_rate = []  for n in n\_grid:  confint\_fail\_cases = 0  for i in range(N\_EXPERIMENTS):  binom\_true = binom(n=n, p=latent\_mu)  q = binom\_true.rvs()   L, R = proportion\_confint(  count=q,  nobs=n,  alpha=0.05,  method='wilson'  )  if L < latent\_mu < R:  pass  else:  confint\_fail\_cases += 1  interval\_success\_rate.append(1 - confint\_fail\_cases / N\_EXPERIMENTS)  plt.xlabel('Розмір вибірки $n$') plt.ylabel('Частка успішних влучень')  plt.plot(n\_grid, interval\_success\_rate, label='Частка успішних влучень', color=turquoise) plt.axhline(0.95, ls='--', label='Желаемая успешность', color=red\_pink)  plt.legend() plt.show() |

|  |
| --- |
| Рисунок 2.8: Залежність частки успішних влучень у довірчий інтервал від розміру вибірки |

Видно, що на будь-якому розмірі вибірки під час використання інтервалу Вілсона можна отримати менше влучень, але що більший розмір вибірки, то менше графік відхиляється від .

# 3. -критерій Фішера

У цьому розділі ми розглянемо -критерій Фішера, який використовується для перевірки гіпотез про середнє значення генеральної сукупності з відомою дисперсією.

Далі, для виведення критеріїв нам потрібен нормальний розподіл. *Потому що саме цьому розподілу підпорядковується середнє вибірок*. Тож давайте подивимося, що це взагалі таке, як з ним працювати в Python й які в нього є властивості.

## 3.1 Нормальний розподіл

Нормальний розподіл — неперервний розподіл, у якому щільність спадає зі збільшенням відстані від математичного сподівання за швидкістю, пропорційною квадрату відстані (див. [формулу 3.1](#eq-normal-density)).

де — випадкова величина, — математичне сподівання, — дисперсія.

На графіку нижче показано, як виглядає нормальний розподіл з різними параметрами та .

|  |
| --- |
| Лістинг 3.1: Візуалізація нормального розподілу з різними параметрами та .  x = np.linspace(-5, 5, 1000) params = [(0, 1), (0, 2), (1, 1), (1, 2), (2, 1), (2, 2)]  for mu, sigma in params:  plt.plot(x, norm.pdf(x, mu, sigma), label=f'μ={mu}, σ={sigma}')  plt.xlabel('x') plt.ylabel('f(x)') plt.legend() plt.show() |

|  |
| --- |
| Рисунок 3.1: Нормальний розподіл з різними параметрами |

## 3.2 Нормальний розподіл у Python

Нехай ми хочемо задати розподіл . Для цього є клас norm[[6]](#footnote-181).

Параметри класу:

* loc — це
* scale — це , або **стандартне відхилення**. Не дисперсія!

Методи класу:

* rvs() — згенерувати випадкові числа з розподілу
* cdf(x) — кумулятивна функція розподілу (cumulative distribution function, CDF) в точці , ймовірність того, що випадкова величина менша або дорівнює .
* ppf(q) — квантиль функції розподілу (percent-point function, PPF) для ймовірності , ймовірність того, що випадкова величина менша або дорівнює .
* pdf(x) — щільність ймовірності (probability density function, PDF) в точці , ймовірність того, що випадкова величина дорівнює .

CDF та PPF — це функції, які пов’язані між собою. CDF визначає ймовірність того, що випадкова величина менша або дорівнює , а PPF визначає значення , для якого ймовірність менша або дорівнює .

Ініціалізуємо клас norm з параметрами та (стандартний нормальний розподіл). Далі, згенеруємо випадкову вибірку з 50 спостережень, а також обчислимо PDF, CDF та PPF для .

|  |
| --- |
| Лістинг 3.2: Нормальний розподіл у Python.  std\_norm = norm(loc=0, scale=1)  rnorm = std\_norm.rvs(size=50, random\_state=42)  CDF = std\_norm.cdf(1.5) PDF = std\_norm.pdf(1.5) PPF = std\_norm.ppf(0.933)  display(  Markdown(f"$P(X \\leq 1.5) = {CDF:.3f}$"),  Markdown(f"$f(1.5) = {PDF:.3f}$"),  Markdown(f"$z\_{{0.933}} = \Phi^{{-1}}(0.933) = {PPF:.3f}$") )  Рядок 1  Ініціалізація класу norm з параметрами та .  Рядок 3  Генерація випадкової вибірки з 50 спостережень.  Рядок 5  Обчислення PDF для .  Рядок 6  Обчислення CDF для .  Рядок 7  Обчислення PPF для .  Рядок 10  Ймовірність того, що випадкова величина менша або дорівнює .  Рядок 11  Ймовірність того, що випадкова величина дорівнює .  Рядок 12  Значення , для якого ймовірність менша або дорівнює . |

Візуалізація методів класу norm показана на [рисунку 3.2](#fig-normal-distribution-2).

|  |
| --- |
| Рисунок 3.2: Демонстрація методів класу norm |

## 3.3 Властивості нормального розподілу

Нормальний розподіл має кілька важливих властивостей[[7]](#footnote-190):

1. **Сума двох незалежних нормально розподілених випадкових величин також має нормальний розподіл**:

де та — незалежні нормально розподілені випадкові величини з параметрами , та , відповідно.

1. **Множення нормально розподіленої випадкової величини на константу також дає нормально розподілену величину**:

де — константа, — нормально розподілена випадкова величина з параметрами , .

### 3.3.1 Перевірка властивостей в Python

За допомогою мови Python ми можемо перевірити ці властивості. Почнемо з [Рівняння 3.2](#eq-normal-sum). Для цього ми згенеруємо дві нормально розподілені випадкові величини та з параметрами , та , . Потім, ми обчислимо їхню суму та перевіримо, чи має вона нормальний розподіл з параметрами та .

|  |
| --- |
| Лістинг 3.3: Візуалізація нормального розподілу суми двох нормально розподілених випадкових величин.  mean\_one, mean\_two = 3, -1 var\_one, var\_two = 4, 2  n = 10000  x1 = norm.rvs(loc=mean\_one, scale=np.sqrt(var\_one), size=n) x2 = norm.rvs(loc=mean\_two, scale=np.sqrt(var\_two), size=n)  x\_sum = x1 + x2 check\_sum = norm(loc=mean\_one + mean\_two, scale=np.sqrt(var\_one + var\_two))  x\_grid = np.linspace(-8, 12, 1000)  fig, ax = plt.subplots() sns.histplot(x\_sum, kde=True, stat='density', color=turquoise, label='Емпіричний розподіл', ax=ax) plt.plot(x\_grid, check\_sum.pdf(x\_grid), color=red\_pink, label='Теоретичний розподіл', alpha=0.8) plt.xlabel('x') plt.ylabel('Щільність') plt.legend() plt.show()  Рядок 1  Параметри та .  Рядок 2  Параметри та .  Рядок 4  Кількість спостережень.  Рядки 6-7  Генерація нормально розподілених випадкових величин та .  Рядок 9  Сума двох нормально розподілених випадкових величин.  Рядок 10  Параметри суми .  Рядок 12  Стандартне відхилення суми .  Рядок 15  Емпіричний розподіл суми .  Рядок 16  Теоретичний розподіл суми . |

Видно, що розподіли приблизно збіглися! А значить ми переконалися, що формула правильна.

Другу властивість [Рівняння 3.3](#eq-normal-multiply) можна перевірити аналогічно. Для цього ми згенеруємо нормально розподілену випадкову величину з параметрами , та помножимо її на константу . Потім, ми перевіримо, чи має вона нормальний розподіл з параметрами та .

|  |
| --- |
| Лістинг 3.4: Візуалізація нормального розподілу множення нормально розподіленої випадкової величини на константу.  mean\_one = 0 var\_one = 1 a = 2 n = 10000 x1 = norm.rvs(loc=mean\_one, scale=np.sqrt(var\_one), size=n) x\_mult = a \* x1 check\_mult = norm(loc=a \* mean\_one, scale=np.sqrt(a\*\*2 \* var\_one)) x\_grid = np.linspace(-8, 8, 1000)  fig, ax = plt.subplots() sns.histplot(x\_mult, kde=True, stat='density', color=turquoise, label='Емпіричний розподіл', ax=ax) plt.plot(x\_grid, check\_mult.pdf(x\_grid), color=red\_pink, label='Теоретичний розподіл', alpha=0.8) plt.xlabel('x') plt.ylabel('Щільність') plt.legend() plt.show()  Рядок 1  Параметри та .  Рядок 2  Параметри .  Рядок 3  Константа .  Рядок 4  Кількість спостережень.  Рядок 5  Генерація нормально розподіленої випадкової величини .  Рядок 6  Множення нормально розподіленої випадкової величини на константу .  Рядок 7  Параметри множення на константу .  Рядки 8,11  Стандартне відхилення множення на константу .  Рядки 12-16  Емпіричний та теоретичний розподіл множення на константу . |

Цього разу розподіли також збіглися. А значить ми переконалися, що формула правильна.

## 3.4 Центральна гранична теорема

Для початку пригадаємо теорему, яка є основоположною теоремою для всіх критеріїв, які ми розглянемо найближчим часом.

**Теорема 3.1 (Центральна гранична теорема, ЦГТ)** Нехай — **незалежно** однаково розподілені випадкові величини, в яких існують математичне сподівання та дисперсія: і , тоді збігається за розподілом[[8]](#footnote-205) до .

Це означає, що якщо випадкові величини в експерименті **незалежні й однаково розподілені** й ваша вибірка **досить велика**, то можна вважати, що

де — середнє арифметичне вибірки, — кількість спостережень, — математичне сподівання генеральної сукупності, — дисперсія генеральної сукупності.

|  |
| --- |
| Примітка |
| Випадкові величини можуть бути слабко залежні одна від одної й злегка по-різному розподілені. Центральна гранична теорема все ще буде правильною, Gnedenko and Kolmogorov (2021). |

### 3.4.1 Візуалізація ЦГТ

Щоб краще розуміти, як працює ЦГТ, я пропоную візуалізувати теорему: подивимося на розподіл середніх значень у різних вибірках. Як ми це зробимо?

* Щоб подивитися, що деяка випадкова величина з нормального розподілу, нам потрібна вибірка цих випадкових величин.
* У цьому випадку нам потрібна вибірка статистик із ЦГТ. Тому нам потрібно згенерувати вибірок по елементів у кожній.
  + По кожній вибірці треба порахувати середнє за елементами.
  + У підсумку ми отримаємо вибірку з елементів.
  + Вона і має бути з нормального розподілу.

|  |
| --- |
| Лістинг 3.5: Візуалізація ЦГТ при великій вибірці з біноміального розподілу.  def visualize\_CLT(sample\_generator, expected\_value, variance):  np.random.seed(42)  N = 5000  clt\_sample = []  for \_ in range(N):  sample = sample\_generator()  sample\_size = len(sample)  statistic = np.sqrt(sample\_size) \* (np.mean(sample) - expected\_value) / np.sqrt(variance)  clt\_sample.append(statistic)   x = np.linspace(-4, 4, 1000)  fig, ax = plt.subplots()  sns.histplot(clt\_sample, kde=True, stat='density', color=turquoise, label='Емпіричний розподіл', ax=ax)  ax.plot(x, norm().pdf(x), color=red\_pink, label='$\mathcal{N}(0, 1)$', alpha=0.8)  plt.legend()  plt.xlabel('X')  plt.ylabel('Щільність')  plt.show()   p = 0.01 n = 20 size = 5000  visualize\_CLT(lambda: np.random.binomial(n, p, size),  expected\_value = p \* n,  variance = n \* p \* (1 - p) )  Рядок 3  Кількість вибірок.  Рядок 4  Пустий масив для зберігання статистик.  Рядок 6  Генерація вибірки з елементами.  Рядок 7  Кількість елементів у вибірці.  Рядок 8  Обчислення статистики.  Рядок 9  Додавання статистики до масиву.  Рядки 11,13-18  Візуалізація емпіричного розподілу та теоретичного розподілу.  Рядок 25  Генерація вибірки з біноміального розподілу.  Рядок 26  Математичне сподівання біноміального розподілу.  Рядок 27  Дисперсія біноміального розподілу. |

Емпірична щільність достатньо близько збігається з теоретичним розподілом. А що якщо зменшити вибірку, за якою рахується середнє?

|  |
| --- |
| Лістинг 3.6: Візуалізація ЦГТ при малій вибірці з біноміального розподілу.  p = 0.05 n = 20 size = 10  visualize\_CLT(lambda: np.random.binomial(n, p, size),  expected\_value = p \* n,  variance = n \* p \* (1 - p) ) |

Стало значно гірше: з’явилися прогалини в розподілі, та й сама емпірична функція розподілу зміщена. Тож наш експеримент підтвердив важливість розміру вибірки для коректної роботи ЦГТ.

Тепер подивимось на експоненціальний розподіл.

|  |
| --- |
| Лістинг 3.7: Візуалізація ЦГТ при великій вибірці з експоненціального розподілу.  p = 5 size = 400 visualize\_CLT(lambda: np.random.exponential(scale=1/p, size=size),  expected\_value = 1/p,  variance = 1/(p\*\*2) )  Рядок 1  Параметр експоненціального розподілу.  Рядок 2  Розмір вибірки.  Рядок 3  Генерація вибірки з експоненціального розподілу.  Рядок 4  Математичне сподівання експоненціального розподілу задається як .  Рядок 5  Дисперсія експоненціального розподілу задається як . |

Бачимо, що і тут усе добре працює!

### 3.4.2 Інші формулювання ЦГТ

Наступні формулювання є еквівалентними, тому що ми можемо перетворити одне в інше за допомогою простих алгебраїчних перетворень. Вони можуть бути корисними в різних ситуаціях, залежно від того, що ми хочемо перевірити.

## 3.5 Нормальна апроксимація й застосування -критерію

### 3.5.1 Апроксимація нормальним розподілом

Згадайте задачу на самому початку [розділу 1.1](#sec-population). У нас є вибірка користувачів з параметром , й ми хочемо перевірити гіпотезу:

де — гіпотетичне значення параметра .

Раніше, ми вирішували цю задачу через біноміальний розподіл:

* Нехай реалізація . Тоді

Згадаємо, як вирішити цю задачу за допомогою Python ([1.19](#lst-python-binomtest)).

А тепер подивимося, що нам говорить ЦГТ:

* За досить великого розміру вибірки ,
* .

При цьому цього разу ми дивимося статистику не в точці , як робили раніше, а в точці , **оскільки у нас неперервний розподіл, то нам не потрібно віднімати 1**:

* у разі нормального розподілу: ;
* у разі біноміального розподілу: .

Подивимось, як це виглядає в Python. Для цього створимо функцію get\_pvalue\_by\_normal\_approx, яка буде приймати на вхід параметри , , та повертати -значення. Порівняємо результати за точним біноміальним тестом та нашим наближенням.

|  |
| --- |
| Лістинг 3.8: Порівняння точного біноміального тесту та нормальної апроксимації при малій кількості спостережень.  def get\_pvalue\_by\_normal\_approx(t, n, mu\_0):  mu = n \* mu\_0  sigma = np.sqrt(n \* mu\_0 \* (1 - mu\_0))  return 1 - norm(loc=mu, scale=sigma).cdf(t)  n = 30 mu\_0 = 0.5 t = 19  p\_value = get\_pvalue\_by\_normal\_approx(t, n, mu\_0)  print(f"p-значення за нормальною апроксимацією = {p\_value:.4f}") print(f"p-значення за точним біноміальним тестом = {binomtest(t, n, mu\_0, alternative='greater').pvalue:.4f}")  Рядок 2  Математичне сподівання біноміального розподілу.  Рядок 3  Стандартне відхилення.  Рядок 4  Обчислення -значення.  Рядок 6  Кількість спостережень.  Рядок 7  Гіпотетичне значення параметра .  Рядок 8  Реалізація статистики.  Рядок 10  Обчислення -значення.  Рядок 12  Виведення -значення.  Рядок 13  Виведення точного -значення.  p-значення за нормальною апроксимацією = 0.0721 p-значення за точним біноміальним тестом = 0.1002 |

Ми бачимо, що значення не дуже-то й збіглися. Але, як ми пам’ятаємо, нормальна апроксимація працює тільки з деякого великого . Тому давайте спробуємо повторити експеримент із більшаимо кількість спостережень.

|  |
| --- |
| Лістинг 3.9: Порівняння точного біноміального тесту та нормальної апроксимації при великій кількості спостережень.  n = 3000 mu\_0 = 0.5 t = 1544  p\_value = get\_pvalue\_by\_normal\_approx(t, n, mu\_0)  print(f"p-значення за нормальною апроксимацією = {p\_value:.4f}") print(f"p-значення за точним біноміальним тестом = {binomtest(t, n, mu\_0, alternative='greater').pvalue:.4f}")  p-значення за нормальною апроксимацією = 0.0541 p-значення за точним біноміальним тестом = 0.0561 |

Ми бачимо, що відмінність тепер тільки в 3 знаку після коми, а не в другому, як раніше. Що більше ми братимемо вибірку, то меншою буде помилка про що говорить ЦГТ.

### 3.5.2 -критерій Фішера

-критерій Фішера використовується для перевірки гіпотез про математичне сподівання випадкової величини з відомою дисперсією. Він є одним із найпоширеніших критеріїв у статистиці, оскільки дозволяє оцінити, чи є різниця між середніми значеннями двох груп статистично значущою.

Для **двостороннього** критерію ми можемо використовувати -критерій Фішера, але з урахуванням того, що ми перевіряємо гіпотезу про те, що не дорівнює . Тобто, ми хочемо перевірити, чи є різниця між середніми значеннями двох груп статистично значущою в обидва боки.

Нульова та альтернативна гіпотези для двостороннього -критерію Фішера мають вигляд:

де — гіпотетичне значення параметра .

Статистика -критерію Фішера має вигляд:

де — середнє арифметичне вибірки, — відома дисперсія генеральної сукупності, — кількість спостережень.

При достатньо великій вибірці згідно ЦГТ -критерій Фішера має нормальний розподіл:

Двосторонній критерій набуває вигляду:

де — рівень значущості.

А -значення для двостороннього критерію розраховується за формулою:

**Односторонній** критерій перевіряє гіпотезу про те, що більше або менше . Нульова та альтернативна гіпотези для одностороннього -критерію Фішера мають вигляд:

Тоді односторонній -критерій Фішера має вигляд:

де — функція розподілу стандартного нормального розподілу, — рівень значущості, — реалізація статистики -критерію Фішера.

## 3.6 -критерій Фішера в Python

Напишемо функцію z\_test\_pvalue, яка буде приймати на вхід параметри sample\_mean (середнє арифметичне вибірки), sample\_size (кількість спостережень), population\_mean (гіпотетичне значення параметра ), population\_variance (дисперсія генеральної сукупності) та alternative (альтернативна гіпотеза). Функція буде повертати -значення для двостороннього або одностороннього -критерію Фішера.

|  |
| --- |
| Лістинг 3.10: Реалізація -критерію Фішера в Python.  def z\_test\_pvalue(sample\_mean, sample\_size, population\_mean, population\_variance, alternative='two-sided'):  z = (sample\_mean - population\_mean) / (np.sqrt(population\_variance) / np.sqrt(sample\_size))  if alternative == 'two-sided':  p\_value = 2 \* min(norm.cdf(z), 1 - norm.cdf(z))  elif alternative == 'greater':  p\_value = 1 - norm.cdf(z)  elif alternative == 'less':  p\_value = norm.cdf(z)  else:  raise ValueError("Оберіть одну з альтернатив: ['two-sided', 'greater', 'less']")  return p\_value  Рядок 2  Обчислення статистики -критерію Фішера.  Рядок 3  Перевірка двосторонньої гіпотези.  Рядок 4  Обчислення -значення для двостороннього -критерію Фішера.  Рядок 5  Перевірка правосторонньої гіпотези.  Рядок 6  Обчислення -значення для правостороннього -критерію Фішера.  Рядок 7  Перевірка лівосторонньої гіпотези.  Рядок 8  Обчислення -значення для лівостороннього -критерію Фішера.  Рядок 10  Виклик помилки, якщо альтернативна гіпотеза не відповідає жодній з можливих.  Рядок 11  Повернення -значення. |

Тепер ми можемо перевірити гіпотезу про те, що не дорівнює , за допомогою -критерію Фішера. Для цього ми можемо використати функцію z\_test\_pvalue та порівняємо з результатами, які ми отримали раніше за допомогою біноміального тесту та нормальної апроксимації.

|  |
| --- |
| Лістинг 3.11: Порівняння -значення за -критерієм Фішера, нормальною апроксимацією та точним біноміальним тестом.  n = 30 mu\_0 = 0.5 t = 19 sample\_mean = t / n population\_variance = mu\_0 \* (1 - mu\_0)  p\_value = z\_test\_pvalue(sample\_mean, n, mu\_0, population\_variance, alternative='greater') print(f"p-значення за Z-критерієм Фішера = {p\_value:.4f}") print(f"p-значення за нормальною апроксимацією = {get\_pvalue\_by\_normal\_approx(t, n, mu\_0):.4f}") print(f"p-значення за точним біноміальним тестом = {binomtest(t, n, mu\_0, alternative='greater').pvalue:.4f}")  Рядок 4  Обчислення математичного сподівання вибірки.  Рядок 5  Дисперсія генеральної сукупності.  p-значення за Z-критерієм Фішера = 0.0721 p-значення за нормальною апроксимацією = 0.0721 p-значення за точним біноміальним тестом = 0.1002 |

Ми бачимо, що -значення за -критерієм Фішера та нормальною апроксимацією збігаються, а точний біноміальний тест дає трохи інше значення. Залишається питання: чи можна уточнити результати -тесту при малих вибірках? Відповідь: так, можна. Для цього існує поправка на неперервність, яка дозволяє покращити точність апроксимації і її ми розглянемо далі.

## 3.7 Поправка на неперервність

Задля кращого розуміння, давайте спочатку візуалізуємо -значення в залежності від величини успіхів експерименту для біноміального тесту та -критерію Фішера. Для цього побудуємо три варіанти:

* -значення *за нормальною апроксимацією*.
  + Розрахунок в Python: 1 - norm.cdf(t).
* -значення *біноміального тесту за умови, що — неціле число*.
  + Розглянемо на прикладі , тоді -значення буде дорівнювати
  + Розрахунок в Python: 1 - binom.cdf(t, n, mu\_0).
* -значення *біноміального тесту за умови, що — ціле число*.
  + Розглянемо на прикладі , тоді -значення буде дорівнювати
  + Розрахунок в Python: 1 - binom.cdf(t - 1, n, mu\_0).

|  |
| --- |
| Лістинг 3.12: Порівняння -значення біноміального і нормального розподілів.  def cmp\_pvalue\_binom\_and\_norm(n, mu0, t, add\_to\_x=0):  x\_axis = np.linspace(0, n, 1000)  dots\_to\_show = np.arange(0, n + 1, 1)   add\_str = "" if add\_to\_x == 0 else f"{add\_to\_x}"   sum\_mu = n \* mu0  sum\_variance = n \* mu0 \* (1 - mu0)  sum\_std = np.sqrt(sum\_variance) # 2>   binom\_dist = binom(n=n, p=mu0)  norm\_dist = norm(loc=sum\_mu, scale=sum\_std)   plt.hlines(1 - binom\_dist.cdf(x\_axis[:-1]), x\_axis[:-1], x\_axis[1:], color=turquoise, linestyle='-',linewidth=1)  plt.vlines(x\_axis[:-1], 1 - binom\_dist.cdf(x\_axis[:-1]), 1 - binom\_dist.cdf(x\_axis[1:]), color=turquoise, linestyle=':', linewidth=1)    plt.scatter(dots\_to\_show, 1 - binom\_dist.cdf(dots\_to\_show-1), color=turquoise,  alpha=1, linewidths=0.5, s=25,  label=f'Binom pvalue = 1-binom.cdf(x-1)')   plt.scatter(t, 1 - norm\_dist.cdf(t + add\_to\_x), color=red\_pink,  alpha=1, marker='o', s=50, label=f'norm p-value({t})')   plt.scatter(t, 1 - binom\_dist.cdf(t - 1), color=turquoise, marker='o',  alpha=1, s=50, label=f'binom p-value({t})')   plt.plot(x\_axis, 1 - norm\_dist.cdf(x\_axis + add\_to\_x), color=red\_pink, alpha=0.5,  label=f'Normal pvalue = 1-norm.cdf(x{add\_str})')      plt.legend()  plt.xlabel('t')  plt.ylabel('$p$-значення')  plt.show()  n = 30 mu\_0 = 0.5 t = 15  cmp\_pvalue\_binom\_and\_norm(n, mu\_0, t)  Рядок 5  Додатковий доданок до -координати (про нього ми поговоримо пізніше).  Рядки 7-8  Параметри нормального розподілу.  Рядки 11-12  Створення біноміального та нормального розподілів.  Рядки 17-19  -значення біноміального розподілу.  Рядки 21-22  -значення нормального розподілу.  Рядки 24-25  -значення біноміального розподілу у точці .  Рядки 27-28  -значення нормального розподілу у точці . |

Якщо порівняти різницю між -значеннями біноміального та нормального розподілів, то ми отримаємо, що -значення біноміального розподілу завжди більше за -значення нормального розподілу. При цьому із збільшенням вибірки ця різниця зменшується. Давайте подивимось на ці різниці для різних значень .

Для початку візьмемо та ([Лістинг 3.13](#lst-continuity-correction-2)).

|  |
| --- |
| Лістинг 3.13: Порівняння -значення біноміального та нормального розподілів при малому та .  n = 20 t = 10 mu\_0 = 0.5  binom\_pvalue = 1 - binom(n, mu\_0).cdf(t - 1) norm\_pvalue = 1 - norm(loc=n \* mu\_0, scale=np.sqrt(n \* mu\_0 \* (1 - mu\_0))).cdf(t) diff = binom\_pvalue - norm\_pvalue  print(f"p-значення біноміального розподілу = {binom\_pvalue:.4f}") print(f"p-значення нормального розподілу = {norm\_pvalue:.4f}") print(f"Різниця між p-значеннями = {diff:.4f}")  Рядок 5  -значення біноміального розподілу.  Рядок 6  -значення нормального розподілу.  Рядок 7  Різниця між -значеннями.  p-значення біноміального розподілу = 0.5881 p-значення нормального розподілу = 0.5000 Різниця між p-значеннями = 0.0881 |

Тепер візьмемо та ([Лістинг 3.14](#lst-continuity-correction-3)).

|  |
| --- |
| Лістинг 3.14: Порівняння -значення біноміального та нормального розподілів при малому та .  n = 20 t = 16 mu\_0 = 0.5  binom\_pvalue = 1 - binom(n, mu\_0).cdf(t - 1) norm\_pvalue = 1 - norm(loc=n \* mu\_0, scale=np.sqrt(n \* mu\_0 \* (1 - mu\_0))).cdf(t) diff = binom\_pvalue - norm\_pvalue  print(f"p-значення біноміального розподілу = {binom\_pvalue:.4f}") print(f"p-значення нормального розподілу = {norm\_pvalue:.4f}") print(f"Різниця між p-значеннями = {diff:.4f}")  p-значення біноміального розподілу = 0.0059 p-значення нормального розподілу = 0.0036 Різниця між p-значеннями = 0.0023 |

І накінці візьмемо та ([Лістинг 3.15](#lst-continuity-correction-4)).

|  |
| --- |
| Лістинг 3.15: Порівняння -значення біноміального та нормального розподілів при великому та .  n = 200 t = 100 mu\_0 = 0.5  binom\_pvalue = 1 - binom(n, mu\_0).cdf(t - 1) norm\_pvalue = 1 - norm(loc=n \* mu\_0, scale=np.sqrt(n \* mu\_0 \* (1 - mu\_0))).cdf(t) diff = binom\_pvalue - norm\_pvalue  print(f"p-значення біноміального розподілу = {binom\_pvalue:.4f}") print(f"p-значення нормального розподілу = {norm\_pvalue:.4f}") print(f"Різниця між p-значеннями = {diff:.4f}")  p-значення біноміального розподілу = 0.5282 p-значення нормального розподілу = 0.5000 Різниця між p-значеннями = 0.0282 |

Ми бачимо, що з ростом вибірки різниця між -значеннями біноміального та нормального розподілів зменшується. Але як зробити так, щоб два -значення збіглися? Для цього слід звернути увагу на точки перетину двох ліній: біноміального та нормального розподілів. Зауважимо, що вони перетинаються приблизно на середині відрізка: між та . Тому спробуємо “змістити” графік нормального розподілу на праворуч.

Це означає, що ми повинні відняти від -координати точки перетину. Тобто, ми можемо використовувати поправку на неперервність, яка дозволяє покращити точність апроксимації. Тоді -значення для біноміального розподілу буде дорівнювати:

де — функція розподілу стандартного нормального розподілу.

Подивимось на графік з поправкою на неперервність ([Лістинг 3.16](#lst-continuity-correction-5)).

|  |
| --- |
| Лістинг 3.16: Порівняння -значення біноміального та нормального розподілів з поправкою на неперервність.  cmp\_pvalue\_binom\_and\_norm(30, 0.5, 15, add\_to\_x=-0.5) |

Ми бачимо, що -значення біноміального та нормального розподілів тепер збігаються.

Порівняємо -значення біноміального та нормального розподілів з поправкою на неперервність ([Лістинг 3.17](#lst-continuity-correction-6)).

|  |
| --- |
| Лістинг 3.17: Порівняння -значення біноміального та нормального розподілів з поправкою на неперервність при та .  n = 30 t = 19 mu\_0 = 0.5  binom\_pvalue = 1 - binom(n, mu\_0).cdf(t - 1) norm\_pvalue = 1 - norm(loc=n \* mu\_0, scale=np.sqrt(n \* mu\_0 \* (1 - mu\_0))).cdf(t) norm\_pvalue\_correct = 1 - norm(loc=n \* mu\_0, scale=np.sqrt(n \* mu\_0 \* (1 - mu\_0))).cdf(t - 0.5)  print(f"p-значення біноміального розподілу = {binom\_pvalue:.4f}") print(f"p-значення нормального розподілу = {norm\_pvalue:.4f}") print(f"p-значення нормального розподілу з поправкою на неперервність = {norm\_pvalue\_correct:.4f}")  Рядок 5  -значення біноміального розподілу.  Рядок 6  -значення нормального розподілу.  Рядок 7  -значення нормального розподілу з поправкою на неперервність.  p-значення біноміального розподілу = 0.1002 p-значення нормального розподілу = 0.0721 p-значення нормального розподілу з поправкою на неперервність = 0.1006 |

Ми бачимо, що -значення біноміального та нормального розподілів з поправкою на неперервність тепер збігаються.

Додамо поправку на неперервність до нашої функції z\_test\_pvalue ([Лістинг 3.18](#lst-z-test-2)).

|  |
| --- |
| Лістинг 3.18: Функція -критерію Фішера з поправкою на неперервність.  def z\_test\_pvalue(sample\_mean, sample\_size, population\_mean, population\_variance, alternative='two-sided', continuity\_correction=False):  if continuity\_correction:  sample\_mean = (sample\_mean \* sample\_size - 1/2) / sample\_size  z = (sample\_mean - population\_mean) / (np.sqrt(population\_variance) / np.sqrt(sample\_size))  if alternative == 'two-sided':  p\_value = 2 \* min(norm.cdf(z), 1 - norm.cdf(z))  elif alternative == 'greater':  p\_value = 1 - norm.cdf(z)  elif alternative == 'less':  p\_value = norm.cdf(z)  else:  raise ValueError("Оберіть одну з альтернатив: ['two-sided', 'greater', 'less']")  return p\_value  n = 30 t = 19 mu0 = 0.5 variance = mu0 \* (1 - mu0)  p\_value = z\_test\_pvalue(t / n, n, mu0, variance, alternative='greater', continuity\_correction=True) print(f"p-значення за Z-критерієм Фішера з поправкою на неперервність = {p\_value:.4f}")  Рядки 2-3  Перевірка наявності поправки на неперервність.  p-значення за Z-критерієм Фішера з поправкою на неперервність = 0.1006 |

Чудово, тепер ми можемо використовувати -критерій Фішера з поправкою на неперервність для перевірки гіпотез про математичне сподівання випадкової величини *з відомою дисперсією*. Але що робити, якщо дисперсія невідома? Для цього існує -критерій Стьюдента, який ми розглянемо далі.

# 4. -критерій Стьюдента

Спробуємо розв’язати таке завдання.

**Приклад 4.1** Менеджмент компанії розглядає новий підхід до планування щотижневих нарад, щоб зменшити втрати часу співробітників. Раніше середня тривалість таких нарад складала 70 хвилин. Ідея полягає в тому, щоб перейти до нової структури нарад, яка, за задумом, дозволить зменшити тривалість нарад до 60 хвилин.

Протягом одного тижня провели 7 нарад у новому форматі й зафіксували їх тривалість. Якщо з’ясується, що нові наради тривають довше, ніж у середньому 70 хвилин, новий формат вважатимуть неефективним.

Ваше завдання — перевірити, чи новий формат нарад дійсно ефективніший.

Вийшла вибірка середньої тривалості нарад (в хвилинах): [50, 55, 70, 45, 40, 70, 80].

Для початку переформулюємо умову мовою математики. Є вибірка:

* — значення середньої тривалості нарад у новому форматі;
* Будемо вважати, що з нормального розподілу, тобто .

meeting\_time = np.array([50, 55, 70, 45, 40, 70, 80])  
print(f"Середнє значення: {np.mean(meeting\_time):.2f} хвилин")

Середнє значення: 58.57 хвилин

Наша гіпотеза звучить так:

Здається, що ми таке вже вміємо вирішувати: згадаємо про -критерій:

* Статистика
* За досить великого розміру вибірки (за ЦГТ)
* Односторонній критерій:
  + -значення = , де — реалізація статистики , — функція розподілу
* Двосторонній критерій:
  + -значення = , де — реалізація статистики

Тоді треба лише порахувати таку статистику: .

Але є суттєва проблема: **ми не знаємо** ! Тому ми не можемо використовувати -критерій.

Давайте спробуємо оцінити за допомогою вибірки. Для цього скористаємося формулою:

Вона називається **вибірковою дисперсією**. Вибіркова дисперсія є *незміщеною* та *консистентною* оцінкою дисперсії генеральної сукупності.

Вибіркова дисперсія є **незміщеною**, оскільки ми ділимо на , а не на . Це робиться для того, щоб уникнути систематичної помилки в оцінці дисперсії. **Консистентність** пояснюється тим, що з ростом вибірки ми все ближче підходимо до істинної дисперсії генеральної сукупності.

Для розрахунку вибіркової дисперсії в Python можна скористатися функцією np.var з параметром ddof=1, що означає, що ми ділимо на .

meeting\_time\_var = np.var(meeting\_time, ddof=1)  
print(f"Вибіркова дисперсія: {meeting\_time\_var:.2f} хвилин")

Вибіркова дисперсія: 222.62 хвилин

Давайте введемо новий критерій -тест, у якому ми підставимо:

Залишилося перевірити: **Чи правда, що при розподіл -статистики — стандартний нормальний?**

Для цього пропонується подивитися, як насправді буде розподілена статистика у завданні, яке було поставлено від початку.

Для цього будемо вважати, що вибірка складається з 7 елементів й .

* Ми раз згенеруємо вибірку та порахуємо щоразу статистику .
* У підсумку ми отримаємо вибірку розміру для й зможемо побудувати гістограму розподілу. Окремо побудуємо розподіл . Якщо емпіричний розподіл візуально збіжиться з теоретичним нормальним, значить, усе добре. А якщо ні, то так просто ми не можемо замінити на .
  + Додатково подивимося, що буде, якщо замінити на . Добре, що на штучному прикладі ми знаємо дисперсію.

Для цього ми напишемо функцію sample\_statistics, яка зможе побудувати розподіл для будь-якої статистики, а не тільки для . Вона приймає на вхід:

* number\_of\_experiments — кількість експериментів, які ми хочемо провести;
* statistic\_function — функція, яка обчислює статистику;
* sample\_size — розмір вибірки;
* sample\_distr — розподіл, з якого ми генеруємо вибірку.

def sample\_statistics(number\_of\_experiments, statistic\_function, sample\_size, sample\_distr):  
 statistic\_sample = []  
 for \_ in range(number\_of\_experiments):  
 sample = sample\_distr.rvs(sample\_size)  
 statistic = statistic\_function(sample)  
 statistic\_sample.append(statistic)  
 return statistic\_sample

Тепер перевіримо, чи дійсно розподілена нормально. Для цього скористаємося функцією sample\_statistics та побудуємо гістограму для . Генерувати вибірку будемо з нормального розподілу .

sample\_size = 7  
M = 100000  
sample\_distr = norm(loc=5, scale=3)  
  
T\_X = lambda sample: np.sqrt(sample\_size) \* (np.mean(sample) - sample\_distr.mean()) / np.sqrt(np.var(sample, ddof=1))  
Z\_X = lambda sample: np.sqrt(sample\_size) \* (np.mean(sample) - sample\_distr.mean()) / sample\_distr.std()  
  
samples = {  
 "T(X)": sample\_statistics(  
 number\_of\_experiments=M, statistic\_function=T\_X,  
 sample\_size=sample\_size, sample\_distr=sample\_distr),  
  
 "Z(X)": sample\_statistics(  
 number\_of\_experiments=M, statistic\_function=Z\_X,  
 sample\_size=sample\_size, sample\_distr=sample\_distr)  
}  
  
for i, name in enumerate(["T(X)", "Z(X)"]):  
 plt.subplot(1, 2, i + 1)  
 current\_sample = samples[name]  
 l\_bound, r\_bound = np.quantile(current\_sample, [0.001, 0.999])  
  
 x = np.linspace(l\_bound, r\_bound, 1000)  
 sns.distplot(current\_sample, label='Емпіричний розподіл', color=turquoise)  
 plt.plot(x, norm(0, 1).pdf(x), label='$\mathcal{N}(0, 1)$', color=red\_pink)  
 plt.legend(loc='upper left')  
 plt.xlabel(f'{name}')  
 plt.xlim((l\_bound, r\_bound))  
 plt.ylabel('Щільність')  
 plt.grid(linewidth=0.2)  
  
plt.show()

|  |
| --- |
| Рисунок 4.1: Симуляція розподілу та |

Ми бачимо, що:

* -тест тут працює: .
* Але ось для це не так! \*\*Вони відрізняються! А значить -критерій не підходить для початкової задачі!

Для того щоб стало зрозуміло, чому так сталося, розглянемо у деталях. При створенні критерію є два кроки:

1. Придумати статистику для критерію
   * Із цим ми успішно впоралися, придумавши .
2. Зрозуміти розподіл статистики.
   * І ось це найскладніший крок, який не дозволяє використовувати будь-яку придуману статистику. Потрібно також розуміти її розподіл.
   * І з цим, як ми побачили, ми провалилися для . Нормальний розподіл не підійшов.

Але чому не розподілена нормально, хоча $? Чому при заміні на усе зіпсувалося?

**Справа в тому, що — це випадкова величина!** Згадаймо, як ми виводили -критерій:

1. Ми порахували, що . З ЦГТ або, у випадку вище, з властивостей нормального розподілу.
2. Далі, все також із властивостей цього розподілу випливає, що якщо ми віднімемо константу або поділимо на константу, то нормальний розподіл не перетвориться на інший: тому .

**Але ми нічого не знаємо про** , де , де невідомо. Ми не знаємо поки що жодних теорем, які б хоч якось доводили, що тут також залишиться нормальний розподіл.

Давайте подивимося на розподіл на все тому ж нормальному розподілі.

S2 = lambda sample: np.std(sample, ddof=1)  
S2\_sample = sample\_statistics(  
 number\_of\_experiments=M, statistic\_function=S2,  
 sample\_size=sample\_size, sample\_distr=sample\_distr  
)  
  
sns.distplot(S2\_sample, label='Емпіричний розподіл', color=turquoise)  
plt.legend()  
plt.xlabel('$\sqrt{S^2}$')  
plt.ylabel('Щільність')  
plt.grid(linewidth=0.2)  
plt.show()

|  |
| --- |
| Рисунок 4.2: Розподіл |

Розподіл несиметричний й незрозуміло як розподілений. Тому, коли ми якусь величину з нормального розподілу ділимо на несиметричний незрозумілий розподіл, ми й отримуємо, що наша статистика не з нормального розподілу.

Тож давайте виведемо критерій, який допоможе розв’язати початкову задачу!

## 4.1 -тест Стьюдента

Для того щоб вивести -тест, нам потрібно зрозуміти, як розподіляється статистика . Для того, щоб це дізнатися, нам знадобиться кілька фактів:

1. Нехай
2. Нехай . Тоді [[9]](#footnote-278).
   * Тоді [[10]](#footnote-279).
   * . Тоді
   * А значить
3. Нехай і з незалежні. Тоді статистика — з розподілу Стьюдента[[11]](#footnote-280) з ступенями свободи.
   * и незалежні[[12]](#footnote-281).
   * Тоді

У підсумку, статистика — взята з розподілу Стьюдента з ступенем свободи. **Але тільки в разі, якщо початкова вибірка з нормального розподілу!**

Тепер нам достатньо даних, щоб побудувати -тест:

Статистика буде виглядати так:

Тоді односторонній критерій набуває вигляду:

А -значення для одностороннього критерію можна обчислити так:

де z — реалізація статистики , — функція розподілу

Двосторонній критерій буде виглядати так:

При цьому -значення для двостороннього критерію можна обчислити так:

де — реалізація статистики , — функція розподілу .

## 4.2 -тест у Python

Давайте тепер протестуємо всі наші теоретичні дослідження на практиці. Для цього нам знадобляться наступні бібліотеки функції:

* scipy.stats.chii2 — для розподілу ;
* scipy.stats.t — для розподілу Стюдента;
* scipy.stats.ttest\_1samp — для -тесту.

Подивимось на розподіл та розподіл .

sample\_size = 7  
sample\_distr = norm(loc=5, scale=3)  
sample = sample\_distr.rvs(sample\_size)  
M = 10000  
  
eta\_statistic = lambda sample: np.var(sample, ddof=1) \* (sample\_size - 1) / sample\_distr.var()  
eta\_sample = sample\_statistics(  
 number\_of\_experiments=M, statistic\_function=eta\_statistic,  
 sample\_size=sample\_size, sample\_distr=sample\_distr  
)  
  
chi2\_dist = chi2(df=sample\_size-1)  
  
l\_bound, r\_bound = np.quantile(eta\_sample, [0.001, 0.999])  
x = np.linspace(l\_bound, r\_bound, 1000)  
  
sns.distplot(eta\_sample, label='Емпіричний розподіл', color=turquoise)  
plt.plot(x, chi2\_dist.pdf(x), label='$\chi^2$', color=red\_pink)  
plt.legend()  
plt.xlabel('$\eta$')  
plt.ylabel('Щільність')  
plt.grid(linewidth=0.2)  
plt.show()

|  |
| --- |
| Рисунок 4.3: Розподіл емпіричного та теоретичного . |

Ми бачимо, що емпіричний розподіл та теоретичний збігаються. Це означає, що ми можемо використовувати -тест для перевірки гіпотези.

Тепер перевіримо, чи дійсно описується розподілом Стьюдента. Для цього скористаємося функцією sample\_statistics та побудуємо гістограму для . Генерувати вибірку будемо з нормального розподілу .

sample\_size = 7  
sample\_distr = norm(loc=5, scale=3)  
sample = sample\_distr.rvs(sample\_size)  
M = 10000  
  
T\_X = lambda sample: np.sqrt(sample\_size) \* (np.mean(sample) - sample\_distr.mean()) / np.std(sample, ddof=1)  
T\_sample = sample\_statistics(  
 number\_of\_experiments=M, statistic\_function=T\_X,  
 sample\_size=sample\_size, sample\_distr=sample\_distr  
)  
  
  
T\_dist = t(df=sample\_size-1)  
  
l\_bound, r\_bound = np.quantile(T\_sample, [0.001, 0.999])  
x = np.linspace(l\_bound, r\_bound, 1000)  
  
sns.distplot(T\_sample, color=turquoise, label='Емпіричний розподіл')  
plt.plot(x, T\_dist.pdf(x), c=red\_pink, label='$t\_{n-1}$')  
plt.plot(x, norm(0, 1).pdf(x), c=slate, linestyle='--', label='$\mathcal{N}(0, 1)$')  
plt.legend()  
plt.xlabel('$T(X)$')  
plt.ylabel('Щільність')  
plt.xlim((l\_bound, r\_bound))  
plt.grid(linewidth=0.2)  
plt.show()

|  |
| --- |
| Рисунок 4.4: Розподіл |

Розподіл Стьюдента практично ідеально описує дані, тоді як нормальний розподіл більш “центрований”.

Тепер, як викликати вбудований -тест у Python? Для цього скористаємося функцією scipy.stats.ttest\_1samp. Вона приймає на вхід:

* a — вибірка;
* popmean — середнє значення генеральної сукупності, яке ми хочемо перевірити;
* axis — вздовж якої осі обчислювати тест. За замовчуванням 0;
* nan\_policy — як обробляти NaN. Може приймати значення propagate, raise, omit. За замовчуванням propagate;
* alternative — альтернативна гіпотеза. Може приймати значення two-sided, less, greater. За замовчуванням two-sided.

meeting\_time = np.array([50, 55, 70, 45, 40, 70, 80])  
  
ttest\_result = ttest\_1samp(meeting\_time, 70, alternative='less')  
print(f"Статистика: {ttest\_result.statistic:.2f}")  
print(f"p-значення: {ttest\_result.pvalue:.2f}")

Статистика: -2.03  
p-значення: 0.04

Оскільки -значення менше 0.05, то ми відхиляємо нульову гіпотезу. Це означає, що середня тривалість нарад у новому форматі триває менше 70 хвилин. Відповідно до -тесту, ми можемо стверджувати, що новий формат нарад дійсно скорочує їх тривалість.

## 4.3 Довірчі інтервали

Давайте тепер розглянемо, як можна оцінити параметри генеральної сукупності за допомогою -тесту. Розглянемо два виведення довірчого інтервалу.

### 4.3.1 Перший метод

Нехай є статистика та критерій для перевірки гіпотези рівня значущості .

Тоді довірчий інтервал для рівня довіри : множина таких , що критерій не відкидає для них .

Нехай — істинне середнє вибірки. Ми також знаємо, що за .

Нас цікавлять такі , що: , у цьому разі критерій не відкинеться.

Розпишемо, щоб у центрі залишилося тільки : . А отже, наш довірчий інтервал:

де

### 4.3.2 Другий метод

Довірчим інтервалом для параметра рівня довіри є пара статистик , таких, що .

Це класичне визначення довірчого інтервалу. Тобто, ми повинні знайти такі та , що .

Тоді

Цей довірчий інтервал збігається з попереднім. Тобто, ми можемо використовувати обидва методи для побудови довірчого інтервалу.

## 4.4 Довірчі інтервали у Python

Давайте тепер побудуємо довірчий інтервал для середнього значення тривалості нарад у новому форматі. Для цього скористаємося функцією scipy.stats.t.interval. Вона приймає на вхід:

* confidence — рівень значущості;
* df — кількість ступенів свободи;
* loc — середнє значення, за замовчуванням 0;
* scale — стандартна девіація, за замовчуванням 1.

Для побудови лівостороннього довірчого інтервалу візьмемо confidence на рівні 90%, оскільки ми хочемо перевірити, чи тривалість нарад у новому форматі *менша* 70 хвилин.

meeting\_time = np.array([50, 55, 70, 45, 40, 70, 80])  
  
confidence = 0.90  
df = len(meeting\_time) - 1  
loc = np.mean(meeting\_time)  
scale = np.std(meeting\_time, ddof=1) / np.sqrt(len(meeting\_time))  
  
interval = t.interval(confidence, df, loc, scale)  
print(f"Довірчий інтервал: {np.round(interval, 2)}")

Довірчий інтервал: [47.61 69.53]

## 4.5 -тест та вимога нормальності

# 5. Монте-Карло в задачах статистики

# Підсумки

# Список літератури

Basu, D. 1955. *On Statistics Independent of a Complete Sufficient Statistic*. *Sankhya*. Vol. 15.

Cochran, William G. 1934. *The Distribution of Quadratic Forms in a Normal System, with Applications to the Analysis of Covariance*. *Mathematical Proceedings of the Cambridge Philosophical Society*. Vol. 30. 3. Cambridge University Press. <https://doi.org/10.1017/S0305004100016595>.

Gnedenko, Boris V., and Alexander N. Kolmogorov. 2021. *Limit Distributions for Sums of Independent Random Variables*. Martino Fine Books.

Lemons, Don S. 2002. *An Introduction to Stochastic Processes in Physics*. The Johns Hopkins University Press.

1. У статистиці позначається як оцінка параметра . [↑](#footnote-ref-22)
2. Метод integers() генерує випадкові цілі числа в заданому діапазоні. Аргумент endpoint вказує, що верхня межа включається у діапазон. [↑](#footnote-ref-23)
3. Розподіл Бернуллі — це дискретний розподіл ймовірностей, який моделює випадковий експеримент з двома можливими результатами: успіхом або невдачею. [↑](#footnote-ref-30)
4. Біноміальний розподіл моделює кількість успішних випадків у послідовності незалежних випробувань. Сума незалежних випадкових величин з розподілу Бернуллі підпорядковується біноміальному розподілу. [↑](#footnote-ref-31)
5. Англ. *cyan*, від грец. κυανouς — “блакитний”, “лазуровий”. [↑](#footnote-ref-38)
6. Документація доступна за посиланням <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html>. [↑](#footnote-ref-181)
7. Доведення цих властивостей можна знайти в роботі Lemons (2002). [↑](#footnote-ref-190)
8. Послідовність випадкових величин збігається за розподілом до , позначаємо , якщо для всіх , в яких неперервна. [↑](#footnote-ref-205)
9. Розподіл — це розподіл суми квадратів незалежних нормальних випадкових величин з нульовим математичним сподіванням. Тобто, якщо , то . [↑](#footnote-ref-278)
10. Доведення Cochran (1934). [↑](#footnote-ref-279)
11. Розподіл Стьюдента — це розподіл, який виникає при нормальному розподілі з невідомою дисперсією. Якщо , то , де — вибіркове середнє, — вибіркова стандартна девіація. [↑](#footnote-ref-280)
12. Доведення Basu (1955). [↑](#footnote-ref-281)