Theorems and Conjectures with Python

ALESSIO DRIVET
GeoGebra Institute of Turin
ITALY

Abstract: - This study investigates the integration of computational methods—specifically the Python programming language—into the exploration and empirical testing of mathematical theorems and conjectures. The distinction between formally proven theorems and conjectures, which remain unproven but are widely believed to be true, serves as the conceptual framework. Three case studies are presented: a numerical verification of Pick's Theorem; an algorithmic test of Goldbach's Conjecture up to a user-specified bound; and a novel, AI-generated conjecture concerning "prime jump permutations," examined through exhaustive enumeration. While emphasizing the inherent limitations of computational experimentation in place of formal proof, this work also highlights the value of such approaches in supporting mathematical intuition, facilitating pattern recognition, and stimulating conjecture formulation. The results underscore the growing role of programming and artificial intelligence in contemporary mathematical inquiry.

Key-Words: - Theorems, Conjectures, Python, Computational Mathematics

Received: May 8, 2024. Revised: January 19, 2025. Accepted: March 21, 2025. Published: May 13, 2025.

1 Introduction

Two fundamental concepts that characterize mathematics are theorems and conjectures. Although they are often associated concepts, they differ in the degree of certainty and how they are validated.

A theorem is a mathematical statement confirmed through rigorous logical reasoning. In other words, a theorem is a proposition that starting from already known principles (axioms, definitions, previous theorems), is demonstrated to be true. Some famous examples of theorems discussed in school are the Pythagorean theorem on right-angled triangles and Euclid's theorem on the infinity of prime numbers. However, some theorems are rarely stated but extremely important, such as Jordan's topological theorem, Pick's geometric theorem, or Bayes' probabilistic theorem.

Conversely, a conjecture is a statement believed to be true but not yet formally proven. Conjectures are generally advanced as hypotheses based on patterns or trends in data but whose truth has not been demonstrated rigorously. Some well-known examples of conjectures to propose in class are the Goldbach conjecture, the twin primes conjecture or the Collatz conjecture.

The use of computer tools increasingly influences mathematics. In particular, Python has become one of the most used tools for exploring and verifying, especially conjectures. While, on the one hand, the formal proof of a theorem requires rigorous logical deduction, on the other hand, Python allows to perform numerical and empirical simulations that can help to verify (not prove) theorems and conjectures, at least in a limited context and under certain conditions. By exploiting the enormous potential of Large Language Models (LLMs), with little effort, it is possible to have very efficient codes that can be refined by operating interactively. Not only that, but an interesting path is to ask for the formulation of a conjecture that has not yet been formulated.

The integration of computational tools like Python into the study of theorems and conjectures offers significant educational opportunities. Using programming to explore and test mathematical concepts, students can gain a deeper, more interactive understanding of mathematics. This approach moves beyond abstract theory and engages students in the practical process of mathematical discovery, where they can test hypotheses, observe patterns, and refine their reasoning.

In the text, we propose three examples that can constitute a first basis for discussion; of course, it would be possible to broaden the discussion, but a much larger space would be needed.

2 Examples

Let us first examine Pick's theorem. It is an interesting and valuable result in geometry, which concerns the area of a polygon with vertices placed on a grid of points with *integer coordinates*. This theorem offers a formula that calculates the area of a

polygon using the number of points I within the polygon and the number of points B on its edges. The formula of Pick's theorem is as follows:

$$A = I + B/2 - 1.$$

The following program allows us to test the theorem by constructing polygons with the desired number of vertices on a grid in a desired number of trials. We emphasize that it is a verification, a numerical testbed; a proof is a separate matter.

Listing 1 Pick's Theorem Verification

```
import matplotlib.pyplot as plt
from shapely.geometry import Polygon
import math

def calculate_edge_points(vertices):
    """Calculate the number of integer points on the edge of the polygon using the GCD."""
    border = 0
    number_vertices = len(vertices)
    for i in range(number_vertices):
        x1, y1 = vertices[i]
        x2, y2 = vertices[(i + 1) % number_vertices]
        border += math.gcd(abs(x2 - x1), abs(y2 - y1))
    return border

def calculate_interior_points(vertices):
    """Calculate the interior points using the formula of Pick's Theorem."""
    n = len(vertices)
    area = 0
    for i in range(n):
        x1, y1 = vertices[i]
        x2, y2 = vertices[(i + 1) % n]
        area += x1 * y2 - y1 * x2
    area = abs(area) / 2
    edge = calculate_edge_points(vertices)
    interior = area - edge / 2 + 1
    return int(round(interior))

def calculate_area_with_pick(vertices):
    """Calculate the area using Pick's Theorem."""
    interior = calculate_interior_points(vertices)
    edge = calculate_edge_points(vertices)
    area = interior + edge / 2 - 1
    return area

def click_management(event):
    """Manages the acquisition of points clicked by the user."""
    if event.xdata is not None and event.ydata is not None:
        x, y = round(event.xdata), round(event.ydata)
        vertices.append((x, y))
        plt.scatter(x, y, color='red')
        plt.draw()
        if len(vertices) == number_vertices:
            plt.close()

def draw_grid(max_size):
    """Create an interactive grid for drawing polygons."""
    fig, ax = plt.subplots(figsize=(6, 6))
    ax.set_xlim(0, max_size)
    ax.set_ylim(0, max_size)
    ax.set_xticks(range(0, max_size + 1))
    ax.set_yticks(range(0, max_size + 1))
    ax.grid(True)
    ax.set_aspect('equal', adjustable='box')
    fig.canvas.mpl_connect('button_press_event', click_management)
    plt.show()

# Input from user
vertices = []
number_vertices = int(input("Enter the number of vertices of the polygon: "))
max_size = int(input("Enter the maximum size of the grid: "))
number_of_experiments = int(input("Enter the number of experiments to run: "))

# Running the experiments
for i in range(number_of_experiments):
    vertices.clear()
    print(f"\nRunning experiment {i + 1}:")
    draw_grid(max_size)

    if len(vertices) == number_vertices:
        polygon = Polygon(vertices)
        calculated_area = polygon.area
        area_pick = calculate_area_with_pick(vertices)

        print(f"Calculated area (Shapely): {calculated_area}")
        print(f"Area according to Pick's Theorem: {area_pick}")

        # Draw the polygon with the results
        x, y = polygon.exterior.xy
        plt.figure(figsize=(6, 6))
        plt.plot(x, y, marker='o', linestyle='-')
        plt.fill(x, y, alpha=0.5, fc='r', ec='black')
        plt.title(f"Calculated Area: {calculated_area}\nArea according to Pick: {area_pick}")
        plt.grid(True)
        plt.gca().set_aspect('equal', adjustable='box')
```

```
    plt.xticks(range(0, max_size + 1))
    plt.yticks(range(0, max_size + 1))
    plt.show()
  else:
    print("The specified number of vertices has not been reached.")
```

Let's take Goldbach's conjecture as a second example. It states that every even number greater than 2 can be written as the sum of two prime numbers.

A program can check whether there are counterexamples up to a given input value. The conjecture has not yet been formally proven, but it has been verified numerically up to extremely large numbers. Therefore, even if the program does not find any counterexamples, it does not guarantee that there are none. Up to 1,000,000 the program should run in a few seconds or minutes. Up to 10,000,000 the program may take minutes or even hours.

Listing 2Goldbach's Conjecture

```
import sympy

# Function to verify Goldbach's conjecture for a single number
def verify_goldbach(num):
  if num <= 2 or num % 2 != 0:
    return None  # Skip checking for numbers that are not even or are less than 2

  # Find two prime numbers that sum to num
  for i in range(2, num // 2 + 1):
    if sympy.isprime(i) and sympy.isprime(num - i):
      return (i, num - i)  # Return the pair of prime numbers
  return None    # No pair found, possible counterexample

# Main function to test all even numbers up to n
def test_goldbach_up_to(n):
  counterexample_found = False
  for i in range(4, n + 1, 2):  # Check all even numbers from 4 to n
    result = verify_goldbach(i)
    if result is None:  # If no valid result is found, it's a counterexample
      counterexample_found = True
      break   # Stop the loop as soon as a counterexample is found

  # Print a final message
  if counterexample_found:
    print("Counterexample found! At least one even number cannot be expressed as the sum of two prime numbers.")
  else:
    print(f"No counterexamples found for even numbers from 2 to {n}.")

# User input
n = int(input("Enter a number n to test all even numbers up to n: "))

# Test all even numbers up to n
test_goldbach_up_to(n)
```

For each even number n , the algorithm checks all prime numbers less than n/2 to see if the sum forms n . For very large values the number of checks for each even number increases significantly and the complexity is about $O(n \cdot \sqrt{n})$.

To conclude this brief review, we can ask the AI to create a new conjecture and implement its verification. First, we define a "prime jump permutation" as a permutation in which each element moves from its original position by a number of positions that is a prime number (positive or negative), with the movements calculated modulo n. The conjecture is:

For any $n \geq 3$, the number of "prime jump permutations" of n elements is NEVER divisible by n.

This conjecture has important implications in group theory and combinatorics, since it indicates a certain regularity in permutation structures.

Listing 3First jump conjecture

```
from itertools import permutations
from math import sqrt

def is_prime(number):
  """
  Check if a number is prime.
  """
  if number < 2:
    return False
  if number in (2, 3):
    return True
  if number % 2 == 0 or number % 3 == 0:
    return False
  i = 5
  while i * i <= number:
    if number % i == 0 or number % (i + 2) == 0:
      return False
    i += 6
```

```
    return True

def is_prime_jump_permutation(permutation):
    """
    Check if a permutation satisfies the prime jump condition.
    """
    n = len(permutation)
    for i, element in enumerate(permutation):
        jump = (element - i) % n
        if jump == 0 or not is_prime(jump):
            return False
    return True

def count_prime_jump_permutations(n):
    """
    Count how many permutations of length n satisfy the prime jump condition.
    """
    return sum(1 for perm in permutations(range(n)) if is_prime_jump_permutation(perm))

def verify_conjecture(max_limit):
    """
    Verify the conjecture for values of n from 3 to max_limit.
    """
    print("{:<5} {:<10} {:<15} {:<20}".format("n", "P(n)", "P(n) mod n", "Conjecture valid?"))
    print("-" * 60)
    for n in range(3, max_limit + 1):
        num_permutations = count_prime_jump_permutations(n)
        remainder = num_permutations % n
        conjecture_valid = remainder == 0
        print("{:<5} {:<10} {:<15} {:<20}".format(n, num_permutations, remainder, 'Yes' if conjecture_valid else 'No'))

# Verify the conjecture for n from 3 to 10
max_limit = 10
verify_conjecture(max_limit)
```

As can be seen from the output, the conjecture is disproved for n = 10.

n	P(n)	P(n) mod n	Conjecture valid?
3	1	1	No
4	2	2	No
5	2	2	No
6	17	5	No
7	24	3	No
8	258	2	No
9	448	7	No
10	770	0	Yes

Computational complexity can result in high processing times. P(n) represents the number of permutations of length n that satisfy the prime jump condition. The most interesting aspect is that it is possible to modify the program by adapting it to variants of the conjecture, for example, by setting the condition that the jump must be an odd number, belong to a particular sequence (e.g., Fibonacci, Catalan or prime numbers) or must be a composite number.

3 Conclusions

This text highlights the role of computational methods, particularly Python, in exploring and verifying mathematical theorems and conjectures. While formal proofs remain the cornerstone of mathematical rigour, computational tools provide valuable insights by enabling numerical experiments and simulations that can support or challenge existing conjectures. The examples presented demonstrate the potential of computational verification but also underscore the limitations of such approaches instead of formal proof.

A simple program can test the twin prime conjecture. The numerical verification of the twin prime conjecture allows us to reason that, although we do not have a formal proof, it is possible to collect a large amount of empirical evidence that suggests the conjecture's validity. This stimulates critical thinking, as students must distinguish between formal evidence and empirical verification.

Looking forward, there are several promising directions for future work. First, extending the current algorithms to handle more complex conjectures or optimise computational efficiency could yield more profound insights into unresolved problems. Additionally, the integration of machine learning techniques could further automate the process of conjecture generation and hypothesis testing, uncovering new mathematical patterns.

Moreover, as computational power continues to grow, the boundaries between empirical testing and formal proof may blur, with increasingly sophisticated algorithms contributing to the discovery of novel proofs or the disproof of long-standing conjectures.

References:

[1] Apostol, T. M. (2013). *Introduction to analytic number theory*. Springer Science & Business Media.
[2] Courant, R., & Robbins, H. (1941). What Is Mathematics? Oxford University Press. *New York*, 3540360.
[3] Cousin, A. A., Andrade, D., & Zenon, W. T. (2010). Pick's theorem in the classroom. *BOLETIM SOCIEDADE PARANAENSE DE MATEMATICA*, *28*(1), 79-83.
[4] Graham, R. L., Knuth, D. E., & Patashnik, O. (1994). Concrete Mathematics, AddisonWesley. *Reading, MA*.
[5] Hardy, G. H., & Wright, E. M. (1979). *An introduction to the theory of numbers*. Oxford university press.
[6] Izadkhah, H., & Behzadidoost, R. (2024). *Challenging Programming in Python: A Problem Solving Perspective*. Springer.
[7] Kosova, R., Kapçiu, R., Hajrulla, S., & Kosova, A. M. (2023). A Revıew of Mathematical Conjectures: Exploring Engaging Topics for University Mathematics Students. *International Journal of Advanced Natural Sciences and Engineering Researches (IJANSER)*, *7*(11), 180-186.
[8] Montgomery, H. L., & Vaughan, R. C. (2007). *Multiplicative number theory I: Classical theory* (No. 97). Cambridge university press.
[9] Polis, C. (1991). Pick's Theorem extended and generalized. *Mathematics Teacher*, *84*(5), 399-401.
[10] Richstein, J. (2001). Verifying the Goldbach conjecture up to $4 \cdot 10^{14}$. *Mathematics of computation*, *70*(236), 1745-1749.
[11] Wang, Y. (2022). A proof of goldbach conjecture by mirror-prime decomposition. *WSEAS Transactions on Mathematics*, *21*, 563-571.
[12] Wang, Y. (2022). A Proof of the Twin Prime Conjecture in the Ƥ x Ƥ Space. *WSEAS Transactions on Mathematics*, *21*, 585-593.

The author have no conflicts of interest to declare that are relevant to the content of this article.