Empirical Investigation of an Open Conjecture: Every odd integer greater than 5 can be expressed as the sum of a prime and twic

Agentic NL→Lean 4 Pipeline
Job #43

April 26, 2026

Abstract

This report documents the empirical investigation of an open mathematical conjecture that could not be formally proved or disproved in Lean 4 with Mathlib. Numerical experiments were conducted to gather evidence for or against the conjecture. The empirical verdict is: **Empirically Supported**. The conjecture remains formally open.

1 Conjecture Statement

Conjecture 1.

Every odd integer greater than 5 can be expressed as the sum of a prime and twice another prime.

2 Status

Formal Status: OPEN — no Lean 4 proof or disproof was found.

Empirical Verdict: Empirically Supported

The pipeline attempted formal verification in Lean 4 with Mathlib but was unable to produce a compiling proof or disproof. Empirical testing was then conducted to gather numerical evidence.

3 Basic Empirical Testing

The following output was produced by the basic numerical experiment:

```
=== EXPERIMENT PLAN ===

Conjecture (Levy's conjecture, 1963):
    Every odd integer n > 5 can be expressed as n = p + 2q,
    where p and q are (not necessarily distinct) primes.

This is a famous open problem closely related to Goldbach's conjecture.
We will collect strong empirical evidence via several complementary tests:

```

```
  TEST 1 (Exhaustive sweep, small n):
      Verify the conjecture for ALL odd n in [7, N_small].
      A single failure here would refute the conjecture.

  TEST 2 (Counterexample search at large random n):
      Sample many large odd integers (up to ~10^8) and check the
      conjecture. This stresses the conjecture in a regime where
      structure is less obvious.

  TEST 3 (Representation count r(n)):
      Compute r(n) = #{(p,q) prime : n = p + 2q}.
      Plot r(n) vs n. The -HardyLittlewood heuristic predicts
          r(n) ~ C * n / (log n)^2
      for an explicit constant C depending on n mod small primes.
      Verifying this asymptotic growth is strong evidence that
      r(n) -> infinity, so r(n) >= 1 a fortiori.

  TEST 4 (Minimum representation count over a window):
      Track min_{n<=N} r(n). If this minimum grows, counterexamples
      become extraordinarily unlikely.

  TEST 5 (Edge cases):
      Test the smallest odd integers (7, 9, 11, ...) explicitly
      and exhibit a witness (p, q) for each.

  TEST 6 (Statistical fit):
      Linear regression of log(r(n)) against log(n / log(n)^2)
      to check the predicted slope ~ 1.

Building prime sieve up to 2,000,000 ...
  148,933 primes found.

--- TEST 1: Exhaustive verification for all odd n in [7, N_MAX] ---
Computing r(n) for all odd n up to 200,000 via prime-shift convolution...
  Odd integers tested in [7, 200000]: 99,997
  Failures (counterexamples): 0
  min r(n)  in range = 1
  mean r(n) in range = 948.77
  max r(n)  in range = 4304

Existence-only sweep for all odd n up to 2,000,000 ...
  Odd integers tested in [7, 2,000,000]: 999,997
  Counterexamples found: 0

--- TEST 2: Random large odd integers up to 10^8 ---
  Random trials: 12,000, range [7, 100,000,000]
  Failures: 0
  Every random odd integer admitted a witness (p,q).

--- TEST 5: Edge cases (smallest odd integers, with explicit witnesses) ---
  n=    7 = 3 + 2*2   verified=True
  n=    9 = 5 + 2*2   verified=True
  n=   11 = 7 + 2*2   verified=True
  n=   13 = 7 + 2*3   verified=True
```

```
  n=   15 = 11 + 2*2   verified=True
  n=   17 = 13 + 2*2   verified=True
  n=   19 = 13 + 2*3   verified=True
  n=   21 = 17 + 2*2   verified=True
  n=   23 = 19 + 2*2   verified=True
  n=   25 = 19 + 2*3   verified=True
  n=   27 = 23 + 2*2   verified=True
  n=   29 = 23 + 2*3   verified=True
  n=   31 = 17 + 2*7   verified=True
  n=   51 = 47 + 2*2   verified=True
  n=   99 = 89 + 2*5   verified=True
  n=  101 = 97 + 2*2   verified=True
  n=  999 = 977 + 2*11   verified=True
  n= 1001 = 997 + 2*2   verified=True

--- TESTS 3,4,6: Growth of r(n) and -HardyLittlewood comparison ---
           N      min r(n) for odd n in [7,N]
         100                                1
         500                                1
        1000                                1
        5000                                1
       10000                                1
       50000                                1
      100000                                1
      200000                                1

  HL heuristic ratio r_actual / r_pred over 206 samples:
    mean   = 0.7255
    median = 0.6981
    std    = 0.1233

  Linear fit log r(n) = a*log(n/log(n)^2) + b
    slope     = 0.9900  (predicted ~ 1)
    intercept = 0.3707

Generating publication-quality plots ...
```

4 Advanced Empirical Testing

A research-grade experiment was designed with nonlinear analysis, parameter sweeps, and convergence testing. Output:

```
=== ADVANCED EXPERIMENT PLAN ===

Conjecture (Lemoine / Levy, 1963): every odd n > 5 admits primes p,q with
n = p + 2q.  This is a number-theoretic conjecture, so the analog of a
"PDE solver" is the *exact, vectorized arithmetic operator* that counts
representations:

    r(n) := #{(p,q) prime : n = p + 2q}

WHAT WE SIMULATE (going far beyond the basic experiment):
```

```
(A) Exact r(n) via FFT-based prime-shift convolution at MULTIPLE
    resolutions N   {2.5e5, 5e5, 1e6, 2e6}.  This is the "grid
    refinement / convergence study": values must agree exactly on the
    overlap region (machine-integer convergence).

(B) Hardy-Littlewood / circle-method asymptotic test against the
    *full nonlinear* singular-series prediction
        r(n) ~ 2 C · n / log²n · S(n),
        S(n) =  _{p|n,  p3} (p-1)/(p-2),
        C    =  _{ p3}(1 - 1/(p-1)²)  (twin-prime constant).
    Computing S(n) for every odd n up to 2·10 via the actual prime
    factorization sieve is the analog of evaluating a closed-form
    reference solution at every grid point.

(C) Truncated Euler-product convergence of  C(P) → C as the prime
    cutoff P →∞ .  This is our analog of *spectral-resolution
    convergence*; expected rate ~ (P log P)^{-1}.

(D) Energy / conserved-quantity analog: the integrated total
        T(N) = Σ_{odd n   N} r(n)
    must agree with the analytic prediction T_pred(N).  Drift is the
    arithmetic equivalent of energy non-conservation.

(E) Parameter sensitivity: replace 2 by k   {1,2,3,4,5,6,7} and
    look for counterexamples to the generalized conjecture
    "every odd n > k+4 = p + k·q".  k=2 is Levy.  We expect k=1 to
    FAIL massively (n must equal 2 + prime), demonstrating that the
    experimental pipeline can correctly REFUTE.

(F) Existence-only sweep up to N_HUGE = 3·10 with the small-q
    witness method, plus a deep secondary check on any unwitnessed n.

(G) Distribution / moments of r(n) at the highest resolution.

EXPECTED IF TRUE : min r(n)   1 at every scale; ratio r/r_pred → 1;
log-log slope = 1; T(N)/T_pred(N) → 1; zero counterexamples in (A,F);
sensitivity test shows k=1 fails, k=2 succeeds, etc.

EXPECTED IF FALSE: at least one odd n>5 with r(n)=0 in (A) or (F).


[1] Building prime sieve up to N_HUGE = 30,000,000 ...
    primes   30,000,000: 1,857,859  (0.1s)

[2] Exact r(n) via FFT convolution at multiple resolutions ...
    N=  250,000: |odd|=124,997 min=1 max=4936 mean=1136.1 failures=0  (0.0s)
    N=  500,000: |odd|=249,997 min=1 max=9257 mean=1999.4 failures=0  (0.1s)
    N=1,000,000: |odd|=499,997 min=1 max=16209 mean=3545.0 failures=0  (0.1s
       )
    N=2,000,000: |odd|=999,997 min=1 max=31586 mean=6327.8 failures=0  (0.2s
       )

[3] Convergence: machine-integer agreement on overlaps
```

```
    max |r_250000(n) - r_500000(n)| over n   250,000 = 0
    max |r_500000(n) - r_1000000(n)| over n   500,000 = 0
    max |r_1000000(n) - r_2000000(n)| over n   1,000,000 = 0

[4] Truncated Euler product  C(P) ...
    C( P      10) = 0.6835937500
    C( P      30) = 0.6651383964
    C( P     100) = 0.6613770845
    C( P     300) = 0.6604871082
    C( P    1000) = 0.6602457440
    C( P    3000) = 0.6601862929
    C( P   10000) = 0.6601682965
    C( P  100000) = 0.6601623455
    C( P1000000) = 0.6601618606
    using  C   0.6601618372  (limit 0.66016181584...)

[5] Singular factor S(n) for odd n   2 ·10 ...
    done (0.2s)

[6] r_actual / r_pred for odd n   [1000, 2,000,000]:
    samples=999,500  mean=0.6236  median=0.6198  std=0.0140  min=0.4997  max
       =0.9073
    log r = a·log(n/log²n) + b →   a=0.9767 (predicted 1)  b=0.3476  R²
       =0.837951

[7] 'Energy' / global integrated count ...Σ
     r(n)  actual    = 6,327,791,195Σ
     r(n)  predicted = 10,235,083,874
    relative error    = 0.3818
    Drift across dyadic windows (T_actual / T_pred):
      [    1000,   10000): ratio = 0.6989
      [   10000,  100000): ratio = 0.6557
      [  100000,  500000): ratio = 0.6324
      [  500000,1000000): ratio = 0.6224
      [1000000,2000000): ratio = 0.6154

[
... [truncated]
```

5 Experiment Code (Basic)

```
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import numpy as np
import random
import math
import time
from collections import Counter

# ============================================================
```

```
# EXPERIMENT PLAN
# ==============================================================
print("=== EXPERIMENT PLAN ===")
print("""
Conjecture (Levy's conjecture, 1963):
    Every odd integer n > 5 can be expressed as n = p + 2q,
    where p and q are (not necessarily distinct) primes.

This is a famous open problem closely related to Goldbach's conjecture.
We will collect strong empirical evidence via several complementary tests:

  TEST 1 (Exhaustive sweep, small n):
      Verify the conjecture for ALL odd n in [7, N_small].
      A single failure here would refute the conjecture.

  TEST 2 (Counterexample search at large random n):
      Sample many large odd integers (up to ~10^8) and check the
      conjecture. This stresses the conjecture in a regime where
      structure is less obvious.

  TEST 3 (Representation count r(n)):
      Compute r(n) = #{(p,q) prime : n = p + 2q}.
      Plot r(n) vs n. The Hardy-Littlewood heuristic predicts
          r(n) ~ C * n / (log n)^2
      for an explicit constant C depending on n mod small primes.
      Verifying this asymptotic growth is strong evidence that
      r(n) -> infinity, so r(n) >= 1 a fortiori.

  TEST 4 (Minimum representation count over a window):
      Track min_{n<=N} r(n). If this minimum grows, counterexamples
      become extraordinarily unlikely.

  TEST 5 (Edge cases):
      Test the smallest odd integers (7, 9, 11, ...) explicitly
      and exhibit a witness (p, q) for each.

  TEST 6 (Statistical fit):
      Linear regression of log(r(n)) against log(n / log(n)^2)
      to check the predicted slope ~ 1.
""")

start_time = time.time()

# ==============================================================
# Sieve of Eratosthenes
# ==============================================================
def sieve(N):
    is_prime = np.ones(N + 1, dtype=bool)
    is_prime[:2] = False
    for i in range(2, int(math.isqrt(N)) + 1):
        if is_prime[i]:
            is_prime[i*i::i] = False
    return is_prime
```

```
N_MAX = 2_000_000  # cap for representation counts and exhaustive sweep
print(f"Building prime sieve up to {N_MAX:,} ...")
is_prime = sieve(N_MAX)
primes = np.flatnonzero(is_prime)
print(f"  {len(primes):,} primes found.")

# ============================================================
# TEST 1: Exhaustive sweep for all odd n in [7, N_MAX]
# ============================================================
print("\n--- TEST 1: Exhaustive verification for all odd n in [7, N_MAX] ---")

# Build set of values 2*q for q prime
two_q = 2 * primes
two_q_set_small = set(int(x) for x in two_q if x <= N_MAX)

N_CNT = 200_000  # smaller cap for full r(n)
print(f"Computing r(n) for all odd n up to {N_CNT:,} via prime-shift convolution...")
P_indicator = is_prime[:N_CNT + 1].astype(np.int32)  # indicator of primality
r = np.zeros(N_CNT + 1, dtype=np.int32)
for q in primes:
    twoq = 2 * int(q)
    if twoq > N_CNT:
        break
    r[twoq:] += P_indicator[:N_CNT + 1 - twoq]

odd_idx = np.arange(7, N_CNT + 1, 2)
r_odd = r[odd_idx]
failures = odd_idx[r_odd == 0]
print(f"  Odd integers tested in [7, {N_CNT}]: {len(odd_idx):,}")
print(f"  Failures (counterexamples): {len(failures)}")
if len(failures) > 0:
    print("  COUNTEREXAMPLES:", failures[:20])
print(f"  min r(n)  in range = {int(r_odd.min())}")
print(f"  mean r(n) in range = {float(r_odd.mean()):.2f}")
print(f"  max r(n)  in range = {int(r_odd.max())}")

# Existence-only sweep up to N_MAX (cheaper memory, only existence)
print(f"\nExistence-only sweep for all odd n up to {N_MAX:,} ...")
exists = np.zeros(N_MAX + 1, dtype=bool)
P_full = is_prime[:N_MAX + 1]
for q in primes:
    twoq = 2 * int(q)
    if twoq > N_MAX:
        break
    exists[twoq:] |= P_full[:N_MAX + 1 - twoq]

odd_all = np.arange(7, N_MAX + 1, 2)
fail_big = odd_all[~exists[odd_all]]
print(f"  Odd integers tested in [7, {N_MAX:,}]: {len(odd_all):,}")
print(f"  Counterexamples found: {len(fail_big)}")
if len(fail_big) > 0:
```

```
    print("␣␣FIRST␣FEW:", fail_big[:20])

# ==============================================================
# TEST 2: Random large odd integers
# ==============================================================
print("\n---␣TEST␣2:␣Random␣large␣odd␣integers␣up␣to␣10^8␣---")

def is_probable_prime(n):
    if n < 2:
        return False
    small = [2,3,5,7,11,13,17,19,23,29,31,37]
    for p in small:
        if n == p:
            return True
        if n % p == 0:
            return False
    d = n - 1
    r_ = 0
    while d % 2 == 0:
        d //= 2
        r_ += 1
    for a in [2,3,5,7,11,13,17,19,23,29,31,37]:
        if a >= n:
            continue
        x = pow(a, d, n)
        if x == 1 or x == n - 1:
            continue
        for _ in range(r_ - 1):
            x = (x * x) % n
            if x == n - 1:

# ... [truncated]
```

6 Experiment Code (Advanced)

```
import numpy as np
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from scipy.signal import fftconvolve
from scipy.stats import linregress
import time, math

t_global = time.time()

print("===␣ADVANCED␣EXPERIMENT␣PLAN␣===")
print("""
Conjecture (Lemoine / Levy, 1963): every odd n > 5 admits primes p,q with
n = p + 2q.  This is a number-theoretic conjecture, so the analog of a
"PDE solver" is the *exact, vectorized arithmetic operator* that counts
representations:
```

```
    r(n) := #{(p,q) prime : n = p + 2q}

WHAT WE SIMULATE (going far beyond the basic experiment):

(A) Exact r(n) via FFT-based prime-shift convolution at MULTIPLE
    resolutions N  {2.5e5, 5e5, 1e6, 2e6}.  This is the "grid
    refinement / convergence study": values must agree exactly on the
    overlap region (machine-integer convergence).

(B) Hardy-Littlewood / circle-method asymptotic test against the
    *full nonlinear* singular-series prediction
        r(n) ~ 2 C · n / log²n · S(n),
        S(n) = _{p|n,  p3} (p-1)/(p-2),
        C    = _{ p3}(1 - 1/(p-1)²)  (twin-prime constant).
    Computing S(n) for every odd n up to 2·10 via the actual prime
    factorization sieve is the analog of evaluating a closed-form
    reference solution at every grid point.

(C) Truncated Euler-product convergence of  C(P) → C as the prime
    cutoff P →∞ .  This is our analog of *spectral-resolution
    convergence*; expected rate ~ (P log P)^{-1}.

(D) Energy / conserved-quantity analog: the integrated total
        T(N) = Σ_{odd n  N} r(n)
    must agree with the analytic prediction T_pred(N).  Drift is the
    arithmetic equivalent of energy non-conservation.

(E) Parameter sensitivity: replace 2 by k  {1,2,3,4,5,6,7} and
    look for counterexamples to the generalized conjecture
    "every odd n > k+4 = p + k·q".  k=2 is Levy.  We expect k=1 to
    FAIL massively (n must equal 2 + prime), demonstrating that the
    experimental pipeline can correctly REFUTE.

(F) Existence-only sweep up to N_HUGE = 3·10 with the small-q
    witness method, plus a deep secondary check on any unwitnessed n.

(G) Distribution / moments of r(n) at the highest resolution.

EXPECTED IF TRUE : min r(n)  1 at every scale; ratio r/r_pred → 1;
log-log slope = 1; T(N)/T_pred(N) → 1; zero counterexamples in (A,F);
sensitivity test shows k=1 fails, k=2 succeeds, etc.

EXPECTED IF FALSE: at least one odd n>5 with r(n)=0 in (A) or (F).
""")

# -------------------- Sieve --------------------
N_FULL  = 2_000_000
N_HUGE  = 30_000_000
print(f"\n[1]␣Building␣prime␣sieve␣up␣to␣N_HUGE␣=␣{N_HUGE:,}␣...", flush=
    True)
t0 = time.time()
def sieve_bool(N):
    s = np.ones(N+1, dtype=bool); s[:2] = False
```

```
    for i in range(2, int(math.isqrt(N))+1):
        if s[i]:
            s[i*i::i] = False
    return s
is_prime_huge = sieve_bool(N_HUGE)
print(f"    primes  {N_HUGE:,}: {int(is_prime_huge.sum()):,}  ({time.time()-t0:.1f}s)")

# -------------------- r(n) at multiple resolutions --------------------
print("\n[2] Exact r(n) via FFT convolution at multiple resolutions ...", flush=True)
def compute_r(N):
    is_p = is_prime_huge[:N+1].astype(np.float64)
    Q = np.zeros(N+1, dtype=np.float64)
    half = N//2
    Q[0:2*half+1:2] = is_prime_huge[:half+1].astype(np.float64)
    C = fftconvolve(is_p, Q)
    return np.rint(C).astype(np.int64)

resolutions = [250_000, 500_000, 1_000_000, 2_000_000]
r_arrays = {}
for N in resolutions:
    t0 = time.time()
    C = compute_r(N)
    odds = np.arange(7, N+1, 2)
    r_odd = C[odds]
    r_arrays[N] = (odds, r_odd)
    nfail = int((r_odd == 0).sum())
    print(f"    N={N:>9,}: |odd|={len(odds):>7,} min={r_odd.min()} "
          f"max={r_odd.max()} mean={r_odd.mean():.1f} failures={nfail}  "
          f"({time.time()-t0:.1f}s)")

print("\n[3] Convergence: machine-integer agreement on overlaps")
for i in range(len(resolutions)-1):
    N1, N2 = resolutions[i], resolutions[i+1]
    o1, r1 = r_arrays[N1]; o2, r2 = r_arrays[N2]
    sub = r2[:len(o1)]
    print(f"    max |r_{N1}(n) - r_{N2}(n)| over n  {N1:,} = {int(np.abs(r1-sub).max())}")

# -------------------- Hardy-Littlewood --------------------
print("\n[4] Truncated Euler product  C(P) ...")
def C2_truncated(Pcut):
    pp = np.flatnonzero(is_prime_huge[:Pcut+1])
    pp = pp[pp >= 3]
    return float(np.prod(1.0 - 1.0/(pp.astype(np.float64)-1.0)**2))
P_cuts = [10, 30, 100, 300, 1000, 3000, 10000, 100000, 1000000]
C2_vals = []
for P in P_cuts:
    v = C2_truncated(P); C2_vals.append(v)
    print(f"     C( P{P:>7}) = {v:.10f}")
C2 = C2_truncated(2_000_000)
print(f"    using  C  {C2:.10f}  (limit 0.66016181584...)")
```

```
print("\n[5]␣Singular␣factor␣S(n)␣for␣odd␣n␣ ␣2 ·10␣...", flush=True)
t0 = time.time()
def compute_S(N):
    S = np.ones(N+1, dtype=np.float64)
    odd_primes = np.flatnonzero(is_prime_huge[:N+1])
    odd_primes = odd_primes[odd_primes >= 3]
    for p in odd_primes:
        S[p::p] *= (p-1)/(p-2)
    return S
S_full = compute_S(N_FULL)
print(f"␣␣␣␣done␣({time.time(
#␣...␣[truncated]
```

7 Conclusion

The conjecture remains formally open. Numerical experiments **support** the conjecture — no counterexamples were found across all tested parameter ranges. Further investigation (both formal and empirical) is warranted.