high_temp_laguerre

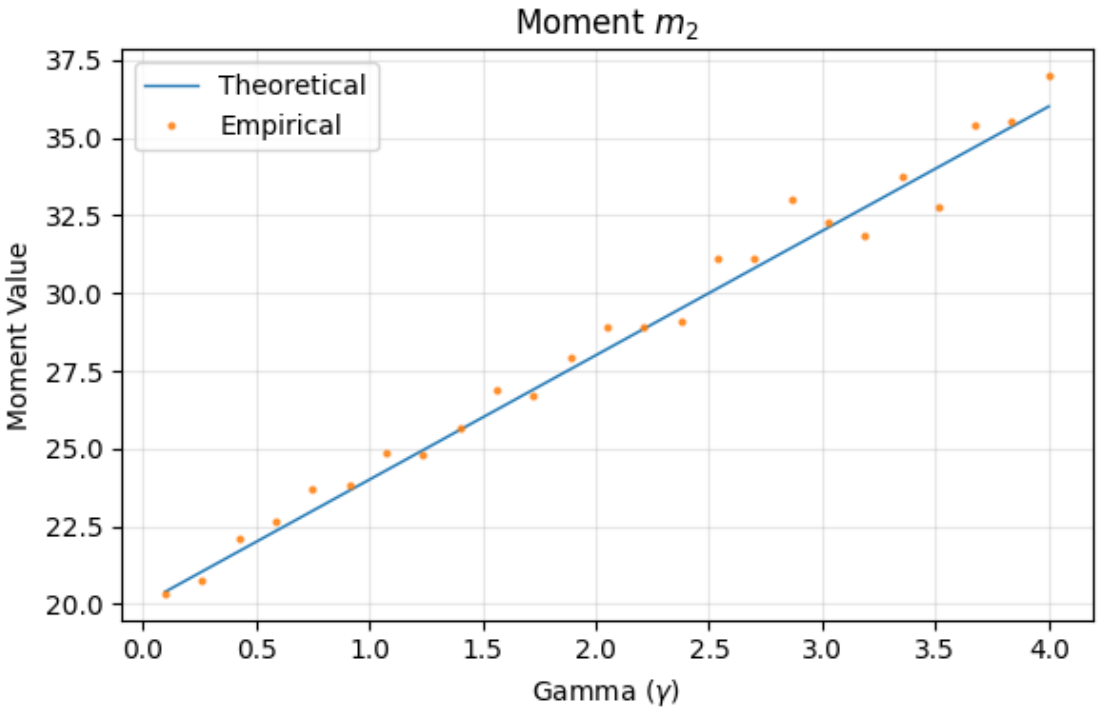
December 10, 2025

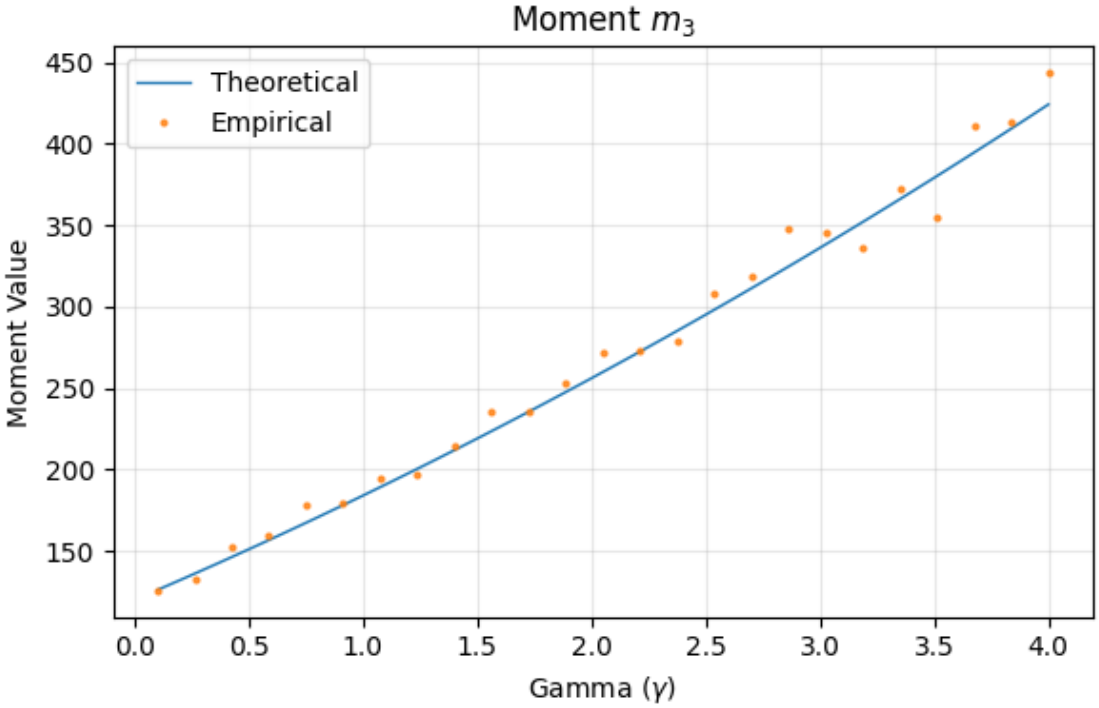
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import eigh_tridiagonal
```

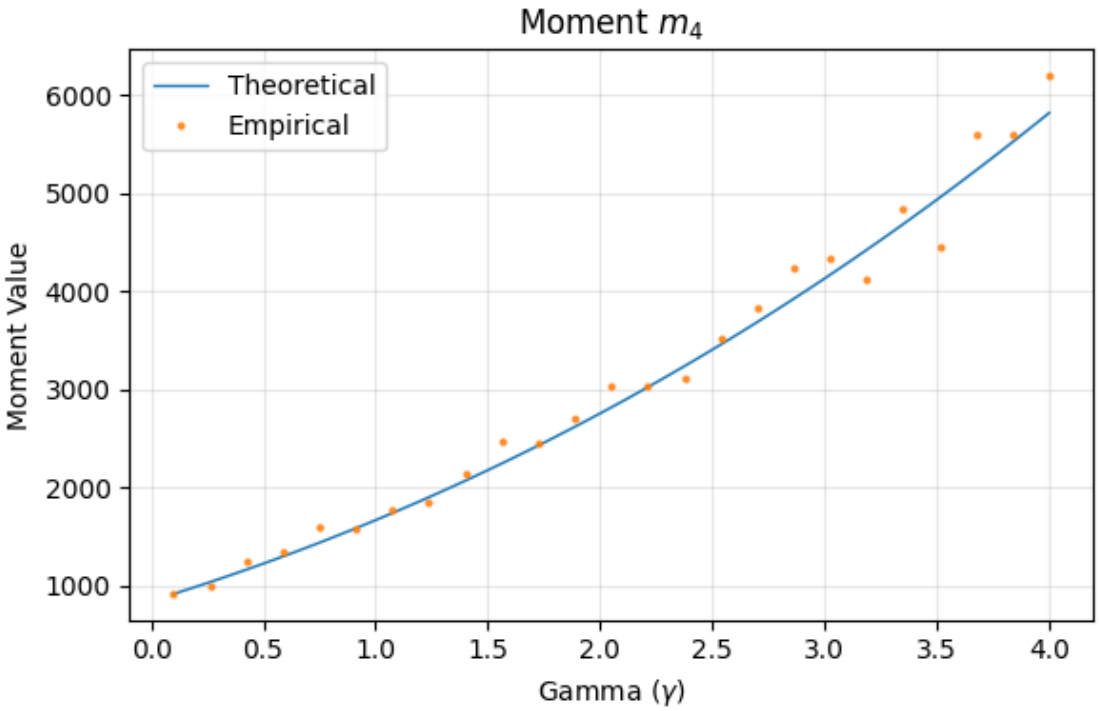
[2]:
```
def c_sequence(lam, gamma, max_n):
    c = np.empty(max_n + 1)
    c[0] = 1.0
    for n in range(1, max_n + 1):
        c[n] = c[n-1] * (lam + n - 1) * (gamma + n - 1) / n

    return c

def series_mult_trunc(a_coeffs, b_coeffs, deg_max):
    res = np.zeros(deg_max + 1)
    la, lb = len(a_coeffs), len(b_coeffs)

    for i, ai in enumerate(a_coeffs):
        if ai == 0:
            continue
        max_j = deg_max - i

        if max_j < 0:
            break

        j_limit = min(lb - 1, max_j)
        for j in range(j_limit + 1):
            res[i + j] += ai * b_coeffs[j]

    return res


def moments_via_Bell_poly(k_max, lam, gamma):
    c = c_sequence(lam, gamma, k_max)

    S = np.zeros(k_max + 1)
    for n in range(1, k_max + 1):
        S[n] = c[n] # remove constant term
```

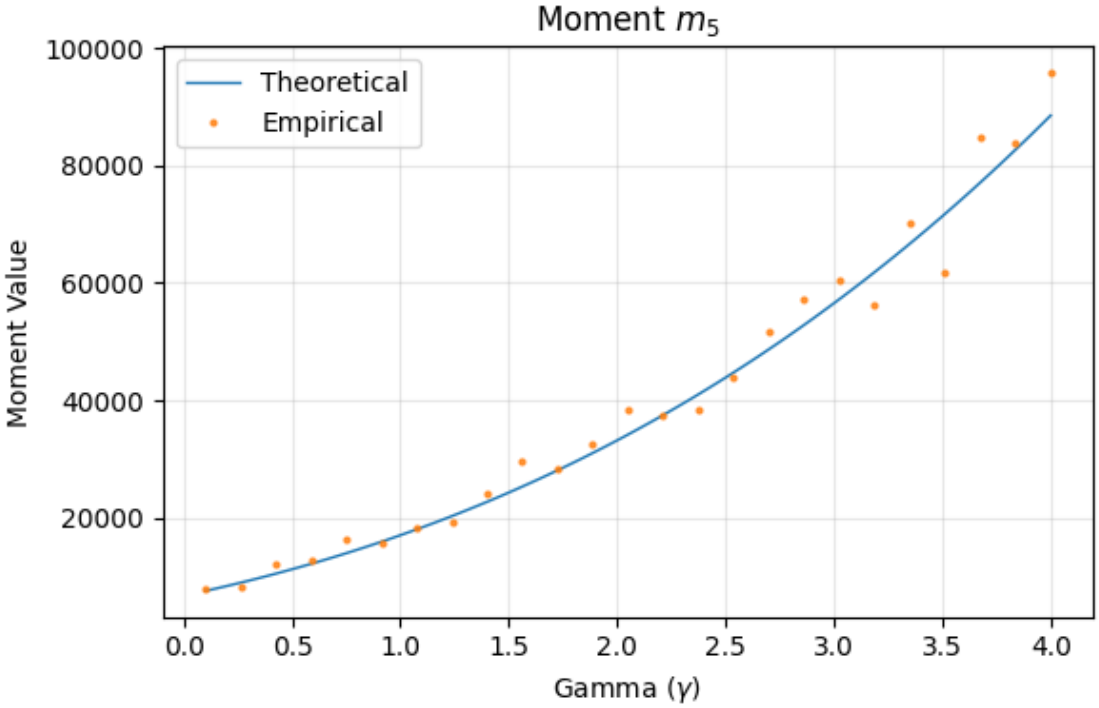
```
    sum = np.zeros(k_max + 1)
    P = S[:]

    for j in range(1, k_max + 1):
        factor = (-1)**(j-1) / j

        for k in range(j, k_max + 1):
            sum[k] += factor * P[k]

        if j < k_max:
            P = series_mult_trunc(P, S, k_max)

    m = np.zeros(k_max + 1)
    m[0] = 1
    for k in range(1, k_max + 1):
        m[k] = (k/gamma) * sum[k]

    return m

def sample_beta_Laguerre_moments(lam, gamma, N=1000, k_max=4):
    M = int(N * lam / gamma) # See Remark 4.11 in BGCG22

    if M < N:
        return "M must be at least N for formula to hold"

    beta = 2 * gamma / N

    mp_scale = 1 / np.sqrt(beta * M)
    scale = mp_scale * np.sqrt(lam)

    df_diag = beta * (M - np.arange(N))
    df_sub = beta * np.arange(N-1, 0, -1)

    diag = np.sqrt(np.random.chisquare(df_diag)) * scale
    sub_diag = np.sqrt(np.random.chisquare(df_sub)) * scale

    tri_diag = np.empty(N)
    tri_upper = np.empty(N - 1)

    tri_diag[0] = diag[0] ** 2
    for i in range(1, N):
        tri_diag[i] = diag[i] ** 2 + sub_diag[i-1] ** 2
        tri_upper[i-1] = diag[i-1] * sub_diag[i-1]

    evals = np.array(eigh_tridiagonal(tri_diag, tri_upper, eigvals_only=True))

    return np.array([np.mean(evals ** k) for k in range(k_max + 1)])
```

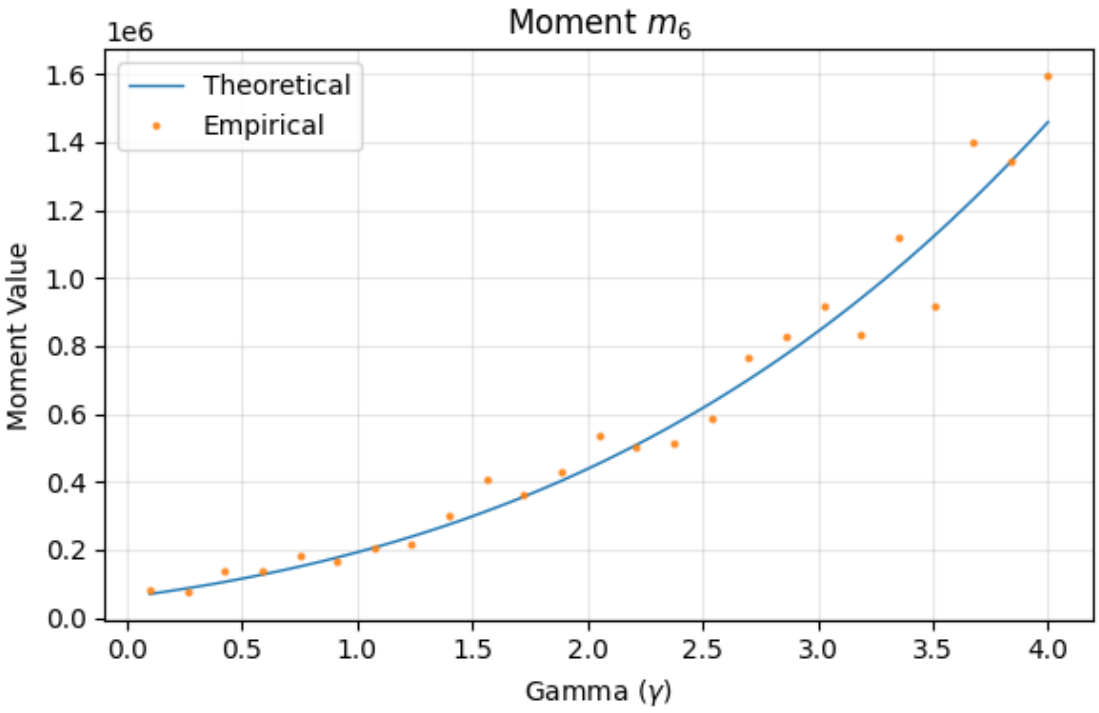
[3]:
```
lam = 4.0
k_max = 10
gamma_theory_val = np.linspace(0.1, 4, 100)
gamma_emp_val = np.linspace(0.1, 4, 25)

theo_res = {k : [] for k in range(k_max + 1)}
emp_res = {k : [] for k in range(k_max + 1)}

for g in gamma_theory_val:
    m = moments_via_Bell_poly(k_max, lam, g)
    for k in range(k_max + 1):
        theo_res[k].append(m[k])

for g in gamma_emp_val:
    m_emp = sample_beta_Laguerre_moments(lam, g, N=3000, k_max=k_max)
    for k in range(k_max + 1):
        emp_res[k].append(m_emp[k])

for k in range(2,k_max + 1):
    fig, ax = plt.subplots(figsize=(6, 4))
    ax.plot(gamma_theory_val, theo_res[k], linewidth=1, label='Theoretical')
    ax.plot(gamma_emp_val, emp_res[k], 'o', markersize=2, alpha=0.8,␣
 ↪label='Empirical')
    ax.set_title(f"Moment $m_{k}$")
    ax.set_xlabel(r"Gamma ($\gamma$)")
    ax.set_ylabel("Moment Value")
    ax.grid(True, alpha=0.3)
    ax.legend()
    plt.tight_layout()
    plt.show()
    fig.savefig(f"beta_Laguerre_moment_{k}_lam_{lam}.png", dpi=300)
```

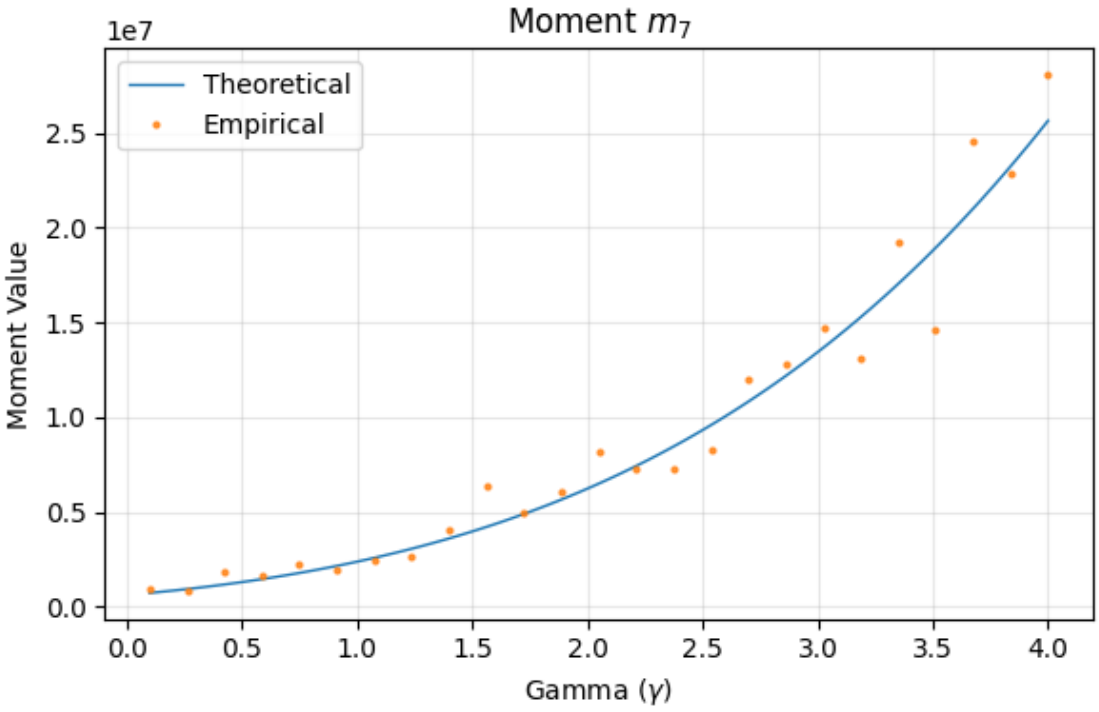

Moment m_2
Theoretical
Empirical
Moment Value
Gamma (γ)
37.5
35.0
32.5
30.0
27.5
25.0
22.5
20.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

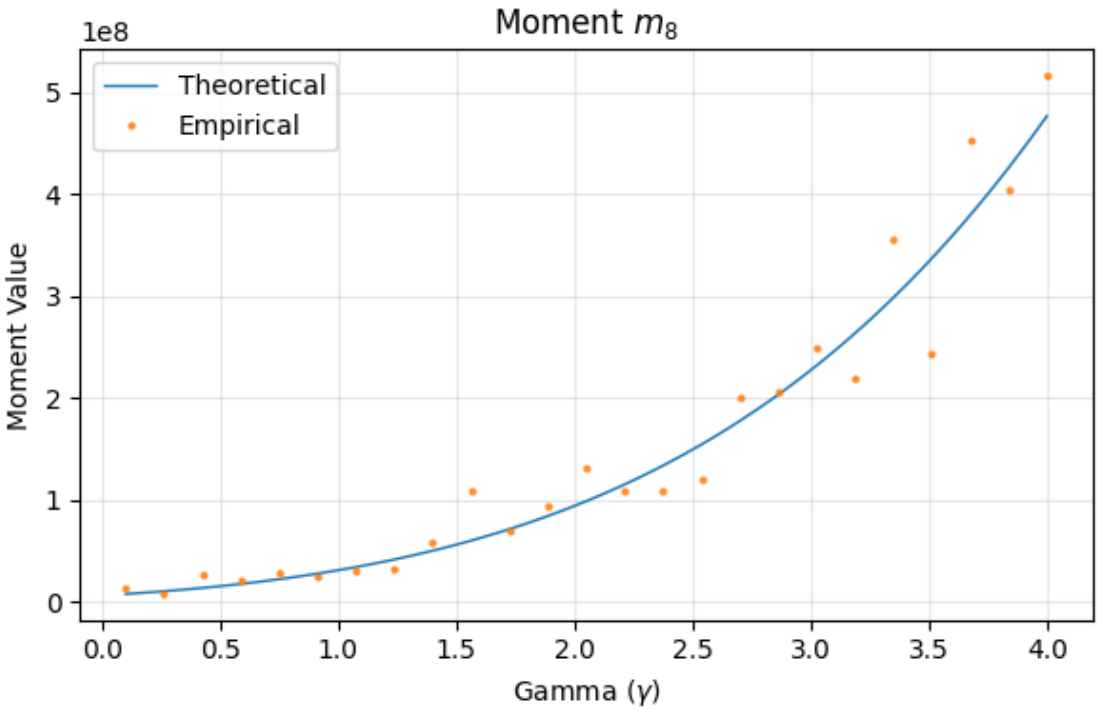

Moment m_3
Theoretical
Empirical
Moment Value
Gamma (γ)
450
400
350
300
250
200
150
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

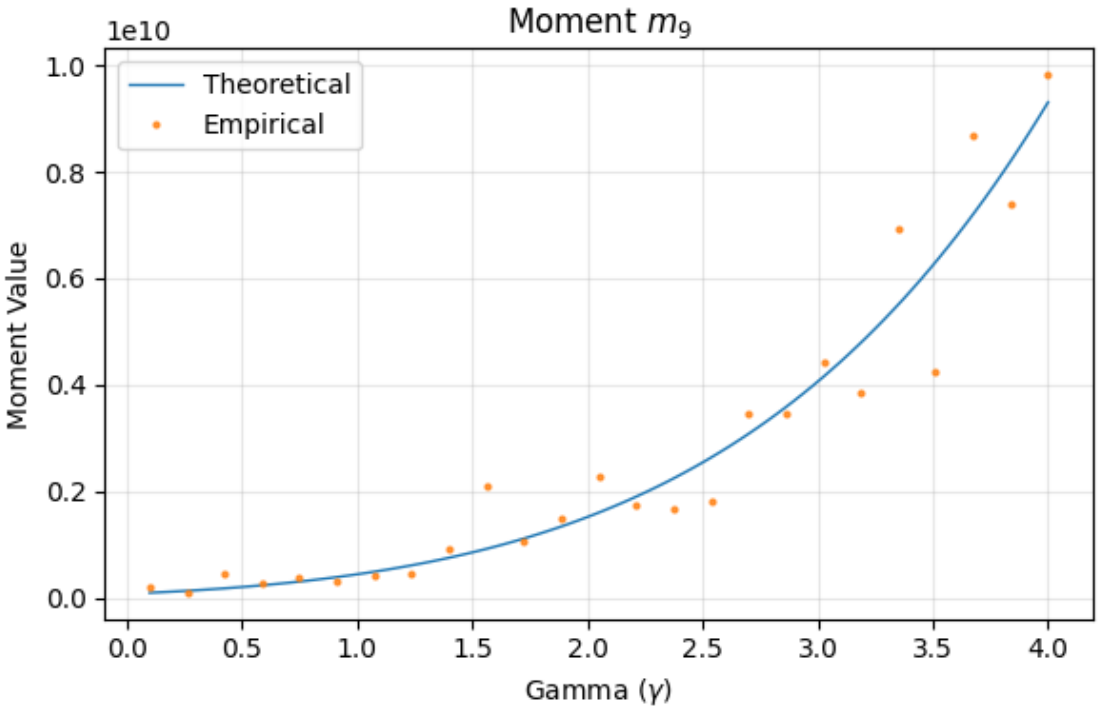

Moment m_4
Theoretical
Empirical
Moment Value
Gamma (γ)
1000
2000
3000
4000
5000
6000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

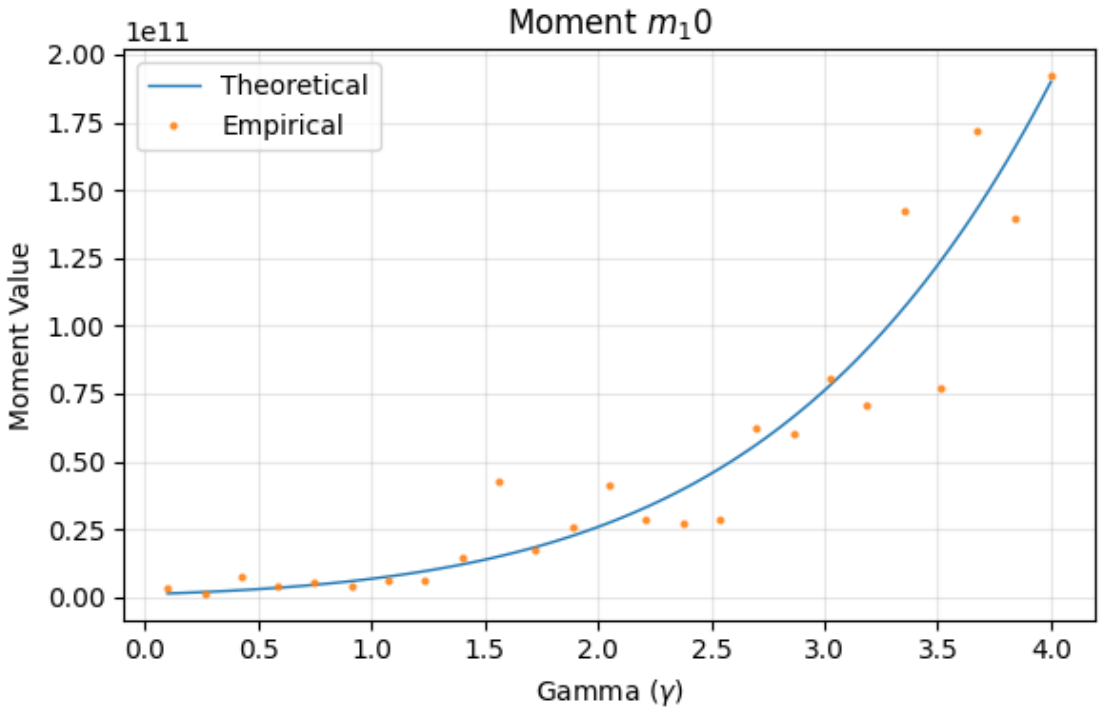

Moment m_5
Theoretical
Empirical
Moment Value
Gamma (γ)
20000
40000
60000
80000
100000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Moment m_6
1e6
Theoretical
Empirical
Moment Value
Gamma (γ)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Moment m_7
1e7
Theoretical
Empirical
Moment Value
Gamma (γ)
0.0
0.5
1.0
1.5
2.0
2.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Moment m_8
1e8
Theoretical
Empirical
Moment Value
Gamma (γ)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
1
2
3
4
5

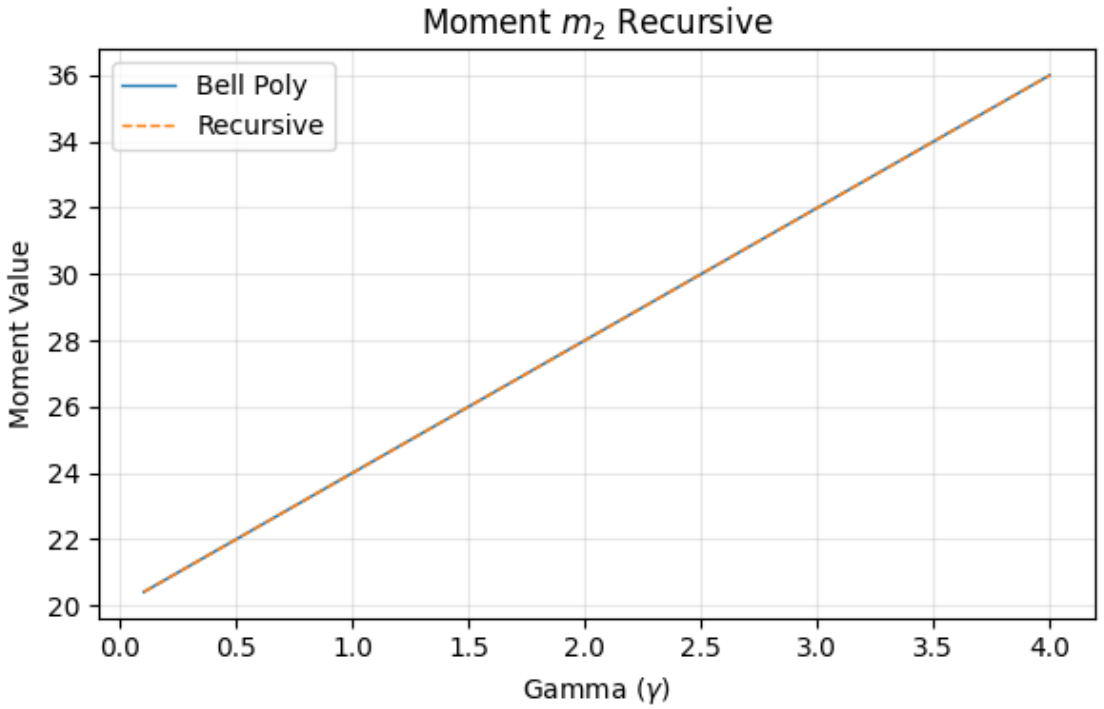

Moment m_9
1e10
Theoretical
Empirical
Moment Value
Gamma (γ)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0.0
0.2
0.4
0.6
0.8
1.0

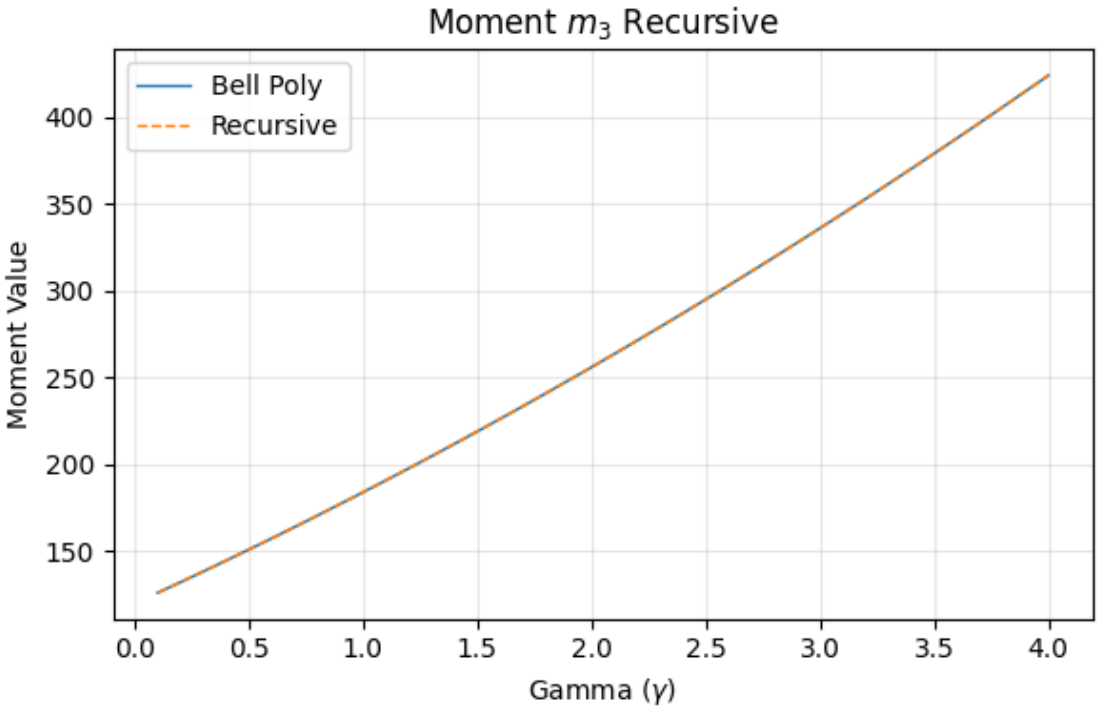


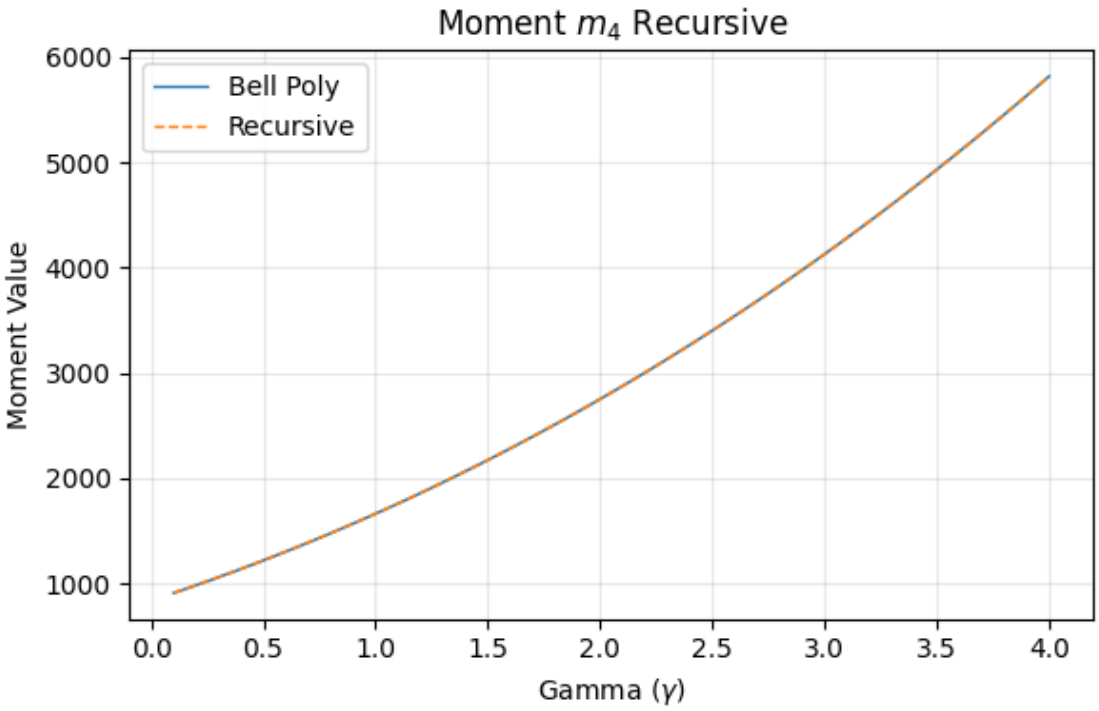
Now we test the recursive formula to see if it agrees with the formula from Bell polynomials

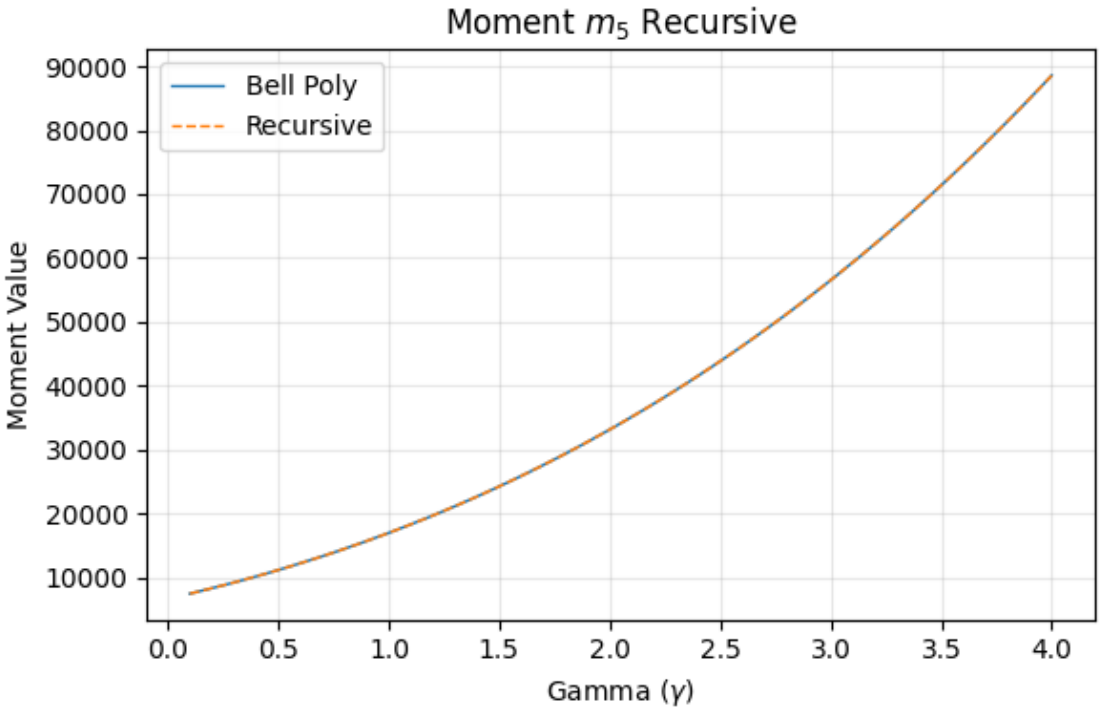
[4]:
```python
import math

def pochhammer(x, n):
    if n == 0:
        return 1.0
    return math.exp(math.lgamma(x + n) - math.lgamma(x))

def c_n(n, lam, gamma):
    if n == 0:
        return 1.0
    return pochhammer(lam, n) * pochhammer(gamma, n) / math.factorial(n)

def moment_recursive(lam, gamma, n_max):
    m = [0.0] * (n_max + 1)
    for n in range(1, n_max + 1):
        cn = c_n(n, lam, gamma)
        conv = 0.0
        for k in range(1, n):
            conv += m[k] * c_n(n - k, lam, gamma)
        m[n] = (n / gamma) * cn - conv
    return m[-1]
```

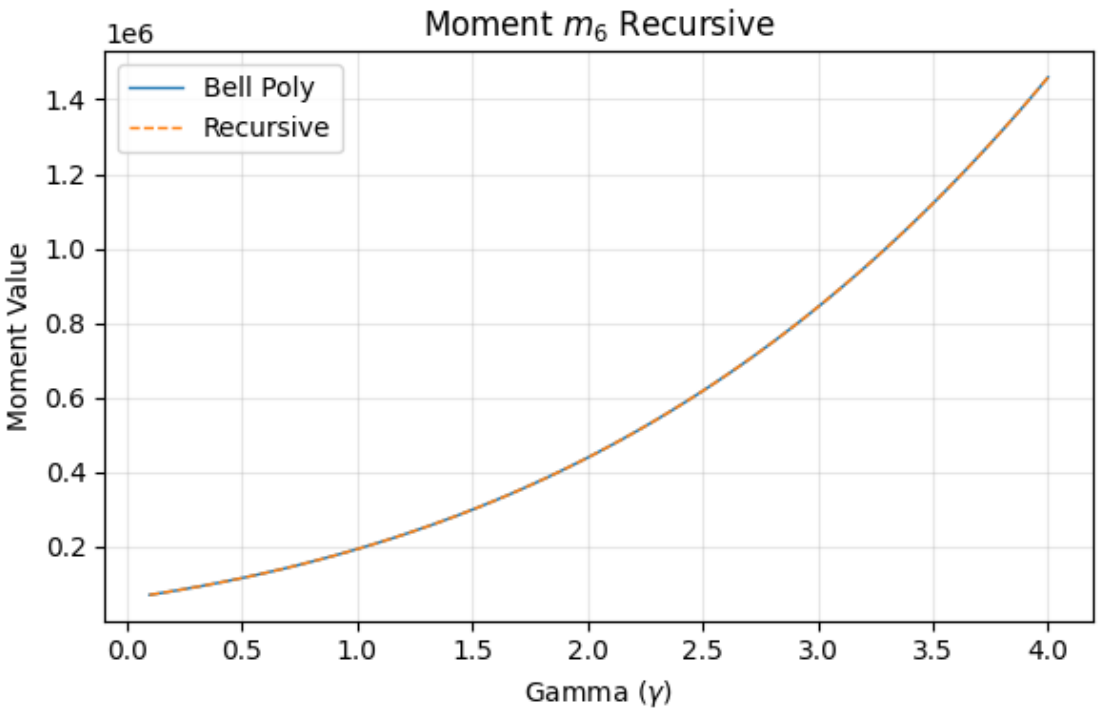
[5]:
```
theo_res_rec = {k : [] for k in range(k_max + 1)}
for g in gamma_theory_val:
    for k in range(k_max + 1):
        m_rec = moment_recursive(lam, g, k)
        theo_res_rec[k].append(m_rec)

# Now plot to compare
for k in range(2, k_max + 1):
    fig, ax = plt.subplots(figsize=(6, 4))
    #ax.plot(gamma_emp_val, emp_res[k], 'o', markersize=2, alpha=0.8,␣
 ↪label='Empirical') Since we already have empirical vs theoretical above, we␣
 ↪just need to ensure the two formulas agree
    ax.plot(gamma_theory_val, theo_res[k], '-', linewidth=1, label='Bell Poly')
    ax.plot(gamma_theory_val, theo_res_rec[k], '--', linewidth=1,␣
 ↪label='Recursive')
    ax.set_title(f"Moment $m_{k}$ Recursive")
    ax.set_xlabel(r"Gamma ($\gamma$)")
    ax.set_ylabel("Moment Value")
    ax.grid(True, alpha=0.3)
    ax.legend()
    plt.tight_layout()
    plt.show()
    fig.savefig(f"beta_Laguerre_moment_{k}_lam_{lam}_recursive.png", dpi=300)
```

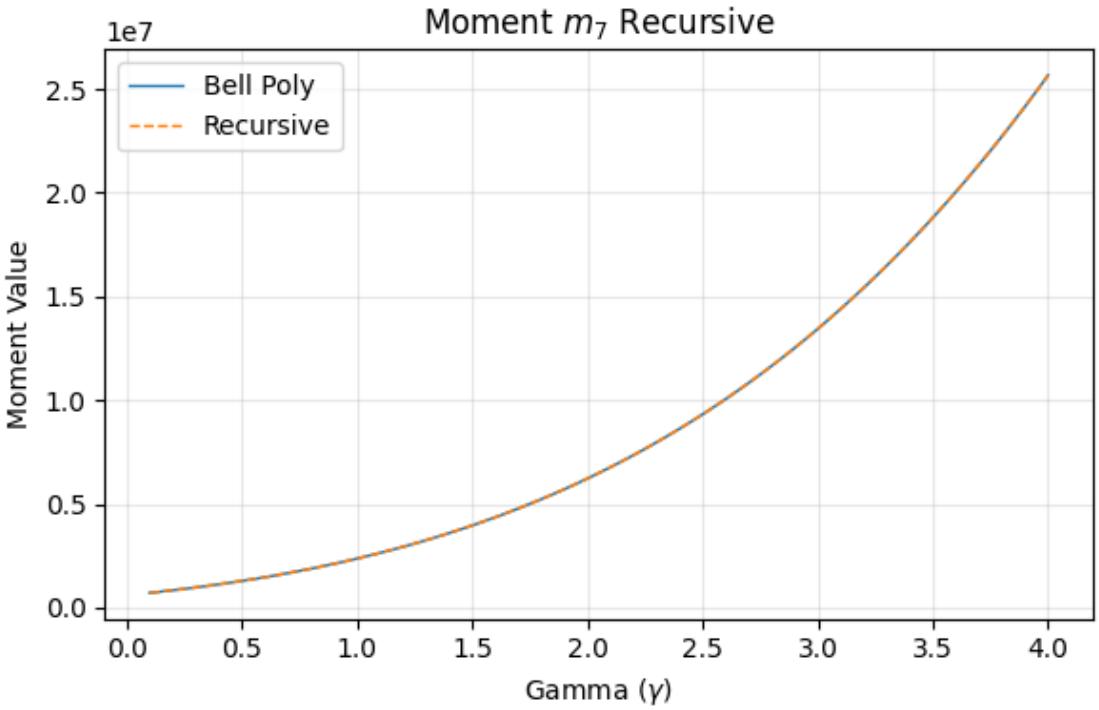


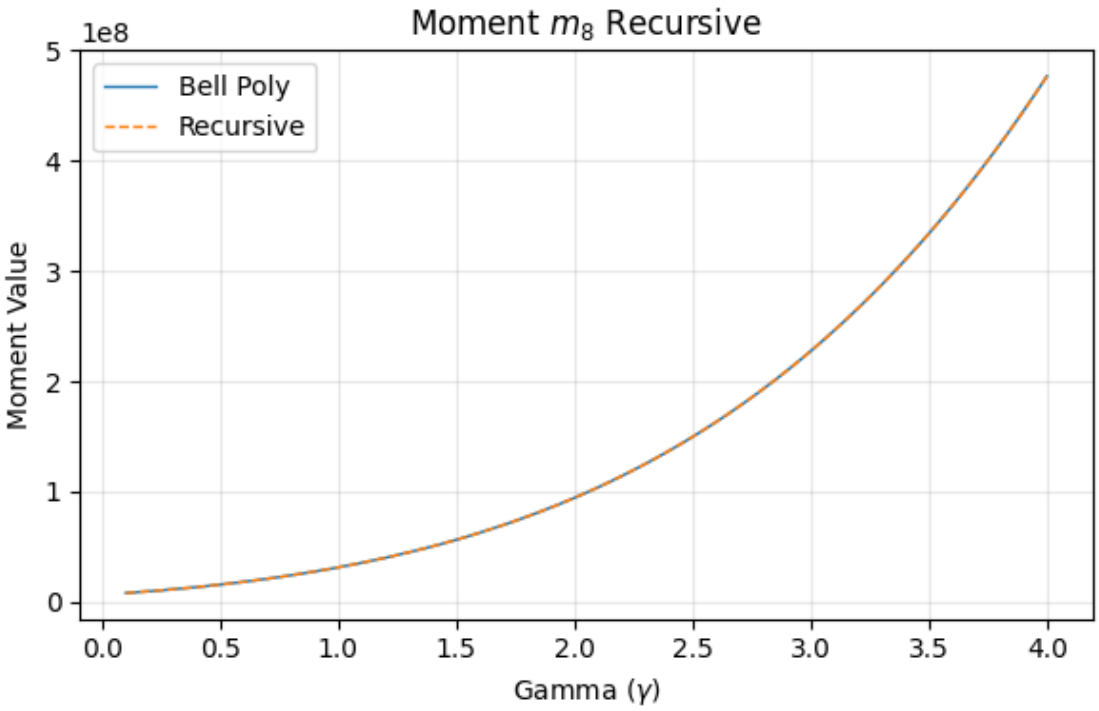

Moment m_3 Recursive
Bell Poly
Recursive
Moment Value
400
350
300
250
200
150
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Gamma (γ)

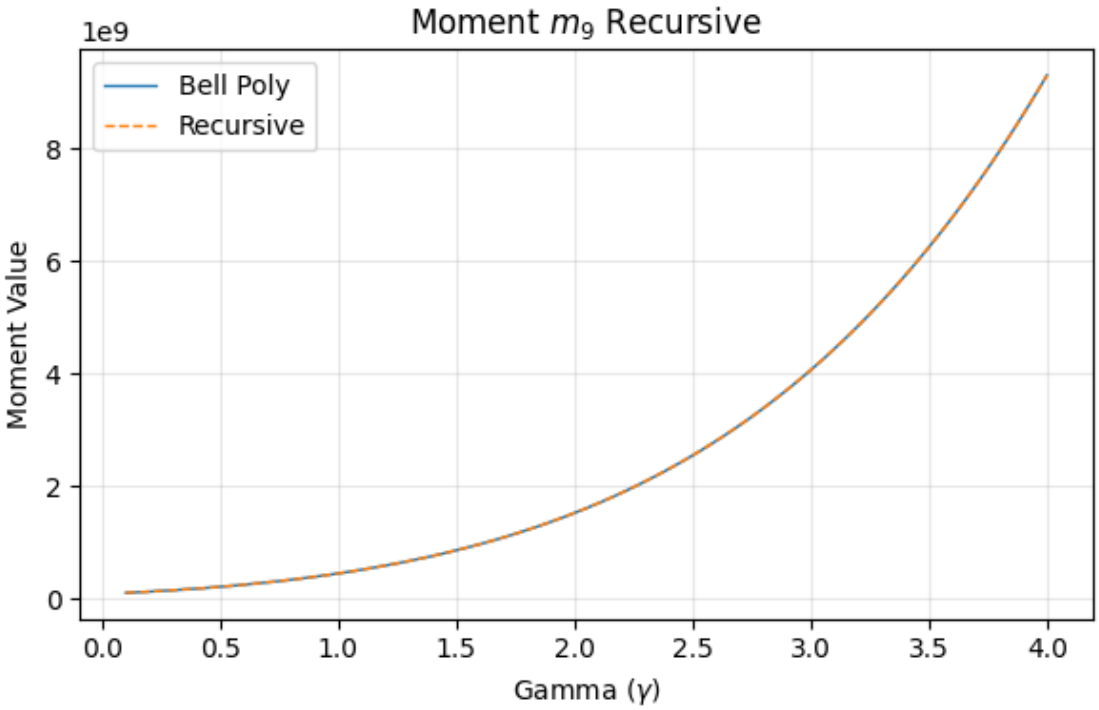

Moment m_4 Recursive
Bell Poly
Recursive
Moment Value
6000
5000
4000
3000
2000
1000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Gamma (γ)

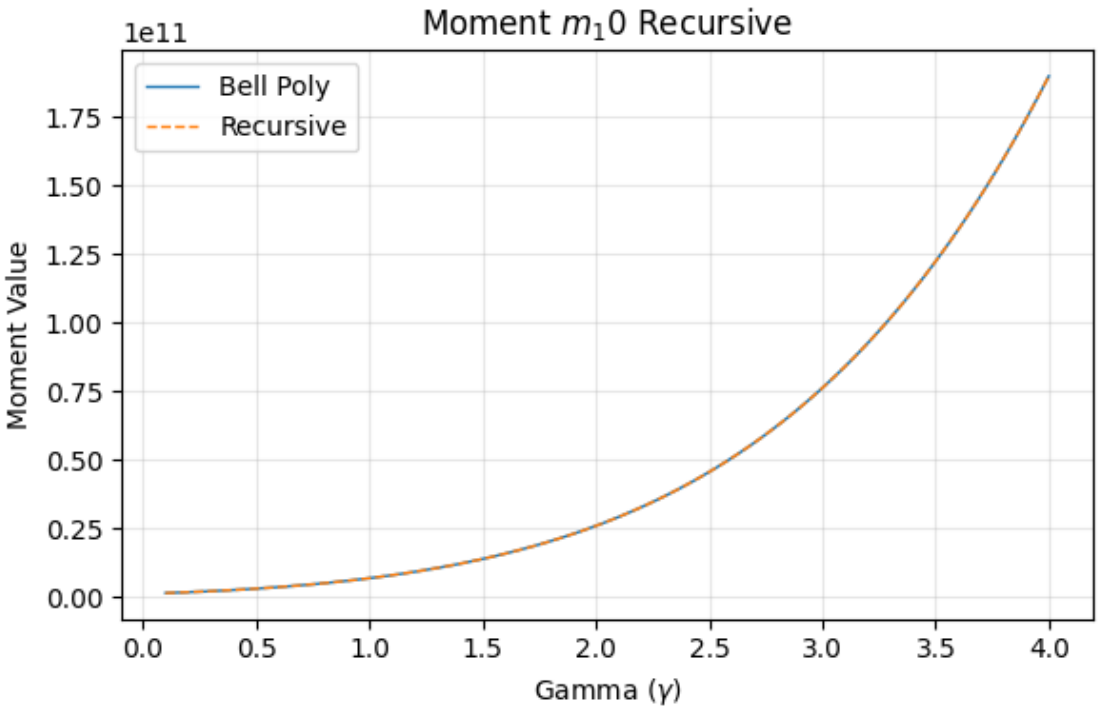

Moment m_5 Recursive
Bell Poly
Recursive
Moment Value
90000
80000
70000
60000
50000
40000
30000
20000
10000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Gamma (γ)


Moment m_6 Recursive
1e6
Bell Poly
Recursive
Moment Value
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Gamma (γ)


Moment m_7 Recursive
1e7
Bell Poly
Recursive
Moment Value
Gamma (γ)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Moment m_8 Recursive
1e8
Bell Poly
Recursive
Moment Value
Gamma (γ)
0
1
2
3
4
5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Moment m_9 Recursive
1e9
Bell Poly
Recursive
Moment Value
0
2
4
6
8
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Gamma (γ)


Moment $m_1$0 Recursive
1e11
Bell Poly
Recursive
Moment Value
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Gamma (γ)