In [181…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
from scipy.integrate import cumulative_trapezoid
from matplotlib.patches import Ellipse
from numpy.polynomial import Polynomial
from pandas.plotting import autocorrelation_plot
import corner
from IPython.display import display, HTML
# from functools import lru_cache

# Base Constants
H_0 = 70  # km/s/Mpc
c = 2.99792458e5  # km/s
M = -19.3146267582  # SN Absolute Magnitude
M_n = 25 - 5*np.log10(H_0) #Script_M
Omega_R = 1e-8

# Test Guess Constants
Omega_K = 0.0001
Omega_M = 0.27 - Omega_R - Omega_K
Omega_DE = 1 - Omega_M
w = -1

# Supernova data
data = pd.read_csv('SCPUnion2.1_mu_vs_z.csv', sep='\t', comment='#', usecols=[0, 1, 2, 3], names=['SNe', 'z', '

z = data['z'].values
m = data['m'].values
m_err = data['m_err'].values

# Covariance matrix
data_cov = (pd.read_csv('SCPUnion2.1_covmat_nosys.csv', sep=r'\s+', header=None)).values

# Calculation of the cumulative integral r(z)
def r_scalar_incremental(z, Omega_M, Omega_DE, w, Omega_K):

    # Creating a finer array from the sorted and unique z values
    z_min, z_max = 0, np.max(z)
    z_fine = np.linspace(z_min, z_max, 580)

    # Interpolate original integrand values to the finer z array
    original_integrand = 1 / (H_0 * np.sqrt(np.abs(Omega_M * (1 + z_fine)**3 + Omega_DE * (1 + z_fine)**(3 * (1

    # Adjust integrand based on Omega_K
    if Omega_K == 0:
        integrand_values_fine = c * original_integrand
    elif Omega_K < 0:
        integrand_values_fine = c * (1 / (H_0 * np.sqrt(np.abs(Omega_K)))) * np.sinh(original_integrand / (1 /
    else:
        integrand_values_fine = c * (1 / (H_0 * np.sqrt(Omega_K))) * np.sin(original_integrand / (1 / (H_0 * np

    # Compute the cumulative integral using the finer differential elements and integrand
    cumulative_integral_fine = cumulative_trapezoid(integrand_values_fine, z_fine, initial=0)

    interpolator = interp1d(z_fine, cumulative_integral_fine, kind='cubic')
    r_values = interpolator(z)

    return r_values

# Calculation of d_l for each z_i
def d_L(z, Omega_M, Omega_DE, w, Omega_K):
    return (1 + z) * r_scalar_incremental(z, Omega_M, Omega_DE, w, Omega_K)

# Calculation of m(z, {pj})
def m_th(z_i, Omega_M, Omega_DE, w, M_n, Omega_K):
    return 5 * np.log10(H_0*d_L(z_i, Omega_M, Omega_DE, w, Omega_K)) + M_n

# Calculation of (m - m^th)
def m_diff_vector(z_i, m, Omega_M, Omega_DE, w, M_n, Omega_K):
    return m - m_th(z_i, Omega_M, Omega_DE, w, M_n, Omega_K)

m_diff_vector(z, m, Omega_M, Omega_DE, w, M_n, Omega_K)
```

Out[181…

```
array([-1.33539416e-01, -5.64629333e-02, -4.74090852e-02, -5.54134333e-02,
        2.35881326e-01,  2.15694647e-01, -2.75547597e-01,  2.85384203e-01,
       -2.12474969e-02,  1.25590718e-01, -1.24979394e-01, -6.78907122e-02,
        1.15639099e-01, -3.03291578e-01, -5.67829106e-02,  1.61873051e-01,
       -1.66405386e-01, -8.89794407e-02, -2.09429353e-01, -1.86675595e-01,
       -3.45190659e-02, -9.00791395e-02, -1.27337036e-01,  8.64919216e-02,
        4.25760474e-02, -1.96388359e-02,  1.28613883e-02, -2.27936364e-02,
```

-3.01025705e-01, -5.31069626e-02, 2.14622427e-01, -1.79733185e-01,
-2.32718369e-01, -7.82399568e-02, 2.47873441e-01, 8.65743672e-02,
-8.61571124e-02, 5.93537545e-02, 1.05339088e-01, 3.37507065e-01,
-7.29604490e-02, 2.34296038e-02, 1.12386049e-01, -3.03557986e-01,
-4.21851019e-01, 4.01384553e-02, -3.37693526e-01, -1.01405626e-01,
 3.80661399e-01, 6.18550269e-02, 9.88789649e-02, 2.73204540e-02,
-1.38989841e-01, 1.37777923e-01, -2.45294566e-03, -3.41636465e-01,
 4.70999234e-02, -7.62981126e-02, -5.10682852e-02, 2.68863635e-01,
 9.62237395e-02, -7.68212256e-02, -7.98831591e-04, -3.24469724e-02,
-2.62390256e-02, -1.28556535e-01, -5.52770868e-02, -6.55426905e-03,
-3.81853478e-01, -3.78121288e-01, 1.12260587e-02, 5.48374213e-02,
 1.64644269e-01, 7.23816702e-02, -3.21490599e-01, 8.32546171e-02,
 4.57656200e-01, 2.05944760e-01, 4.98668081e-01, 1.14966768e-01,
 4.41521008e-01, -9.90381529e-03, -3.25845741e-01, 7.58030753e-02,
 1.58415606e-01, -3.79748663e-02, 7.43797686e-02, -1.55157529e-01,
 1.50665694e-01, 4.02507455e-03, 4.62556179e-02, 6.26612769e-02,
 4.98240819e-02, 2.39051978e-02, -1.34944085e-01, 1.27275603e-01,
 1.03757863e-01, 1.46968126e-01, -5.61745007e-02, 9.95134948e-02,
-2.73604711e-02, 2.00205486e-01, -5.22271299e-02, 7.54085812e-02,
 4.85638251e-01, 4.02093550e-02, 1.26624957e-01, -2.44238969e-02,
 3.07632566e-01, 3.15674175e-02, 2.84570306e-01, 5.99051547e-01,
-1.52805435e-01, 3.41363033e-02, -4.43725561e-01, 1.28905818e-01,
 1.25731539e-01, 2.08237076e-02, 2.87102948e-01, 1.33491039e-01,
 1.11804111e-02, -1.48225289e-01, 1.45548580e-01, 1.63423310e-01,
 4.75911394e-03, -2.54103029e-02, 2.67186250e-02, -9.72520444e-02,
-1.59625997e-01, -5.99004828e-02, 1.41662633e-01, 6.43514602e-02,
 3.15147883e-01, 1.25032956e-01, -1.30434797e-01, 8.83031778e-02,
-9.19603065e-02, -4.90371106e-02, -1.49890840e-01, 4.46652434e-01,
-1.16210300e-01, -4.58400659e-03, -3.08523600e-01, -2.50289260e-01,
 7.06426169e-02, -6.63392589e-02, 1.41964388e-01, 2.00771711e-01,
 4.67860431e-02, -1.09404800e-01, -2.51259213e-01, -1.98368269e-01,
-1.09311444e-01, -3.69557190e-01, 6.81049041e-02, -1.01708577e-01,
-1.32082019e-01, -1.20557606e-01, -6.50695783e-02, -1.18390573e-01,
 7.52005204e-02, 5.15456733e-02, 1.66909300e-01, -8.75493032e-02,
-8.22590237e-02, 1.30364055e-01, -8.52866796e-02, -1.07363745e-01,
 2.25093935e-02, -5.64303121e-02, 2.54654724e-02, 3.45473474e-01,
-2.76674752e-03, 1.05059581e-01, -3.08243861e-01, 1.06100328e-01,
-4.86487585e-02, -5.41621918e-02, 3.32887222e-02, 2.04683691e-01,
 8.40103545e-02, -1.46692777e-01, -1.01253770e-02, -1.76573098e-01,
 1.74320196e-02, -3.01533015e-01, 5.09430285e-01, -4.05924498e-02,
 3.45734731e-02, -9.39239470e-02, 1.47918972e-01, -1.19243611e-01,
-1.46165853e-01, -8.58416085e-02, 1.34536895e-01, 1.91545144e-01,
-1.33120064e-01, -1.46695049e-01, -1.45368473e-03, 1.83413351e-01,
 6.57810015e-02, 7.81296299e-02, 4.57054593e-01, -7.34245165e-02,
 1.22632170e-01, 1.70701491e-01, 2.77540005e-02, 5.42162856e-02,
-3.30970218e-02, -1.20938469e-01, -2.79152512e-02, -4.88065180e-04,
-9.79977213e-02, -1.58600195e-01, -1.08238412e-01, -8.88587902e-02,
 2.71704143e-01, 6.16983679e-02, 1.65587113e-01, -7.50224646e-02,
-1.27962918e-03, -2.55807687e-01, -2.27061663e-01, -3.78033728e-02,
-1.48312614e-02, -8.61834332e-02, 6.08266243e-02, -1.19728882e-02,
-3.78398178e-02, 1.32969497e-02, -6.17046368e-02, -1.63320737e-01,
 1.04574573e-01, -3.05453307e-02, -9.54559048e-02, -1.25461867e-01,
 7.30536117e-02, 1.61427930e-02, 1.64241689e-02, -2.17282269e-02,
 8.39991929e-02, 3.92665387e-01, 2.24914953e-01, 1.04421199e-02,
-1.39685471e-01, -4.64618887e-02, 2.82386117e-02, -3.00529850e-02,
-1.06651200e-01, -1.89814025e-02, -1.33174423e-01, 3.68686951e-02,
-8.31442714e-03, 2.90798576e-01, 6.72379891e-03, 6.99540336e-02,
-4.05623976e-02, -2.90278896e-01, 1.33148324e-02, -3.45066743e-02,
-1.67927649e-01, -1.07786335e-01, 3.96545619e-02, -6.13242516e-02,
 1.43865067e-01, -2.22273209e-01, 1.92190033e-01, -1.20889963e-01,
-2.10532450e-01, -1.23656893e-01, 1.47844268e-02, 1.25419472e-01,
 4.34260106e-02, 1.37711596e-01, 3.07728518e-02, -2.96529254e-02,
-1.14788976e-03, 4.21419660e-02, 6.43424065e-02, -3.34232738e-02,
 4.05635566e-02, 1.02089270e-01, -4.37985390e-03, -1.35712292e-01,
-1.11824068e-01, 3.80883905e-01, 2.98556471e-01, 5.21226599e-02,
-4.70361998e-02, -2.67694929e-02, -2.75390718e-01, 1.17160144e-01,
 2.91685452e-02, 1.45973711e-02, -2.01997880e-02, 2.72494786e-01,
-4.91912848e-02, -8.20488119e-02, 1.99874980e-01, -5.71437598e-01,
 3.74957433e-01, -1.40539840e-01, -6.89112688e-03, 4.92117935e-01,
 5.08245738e-01, 3.29168185e-01, 9.95148336e-02, 7.89577270e-02,
-1.23613872e+00, 1.80188912e-01, -9.32404276e-02, 6.19117308e-01,
-6.16898580e-01, -1.76511502e-01, 5.15187351e-01, 1.11930146e-01,
-3.23809999e-02, -1.24158940e-01, -1.63897442e-01, 1.65492315e+00,
 4.79121400e-01, 1.80833578e+00, 1.42225189e+00, -2.98585319e-01,
-4.61302617e-01, 1.06365042e+00, 6.29372397e-01, -8.83596719e-02,
 2.62469825e-01, 1.41299130e-01, -1.38759780e-01, -4.43607314e-01,
 4.18165129e-02, 5.82976791e-01, 3.93944374e-01, -1.70788797e-02,
-2.88809009e-01, 6.09632538e-01, -6.84399115e-01, 7.10196271e-01,
-2.68857678e-01, 7.95393277e-01, -6.57822106e-01, 4.29571775e-01,
-2.82198414e-01, 3.64497365e-01, -6.53564595e-01, 2.69646591e-02,
-5.79533257e-01, 3.75309851e-01, 4.87975335e-01, 5.99098020e-01,
 8.89106281e-02, 3.47436785e-01, -1.99627294e-01, -1.92544530e-01,
 7.70954059e-02, -1.54125116e-01, -4.24800779e-01, 4.27136864e-02,

```
-4.79821139e-01,  6.96550992e-02, -2.28637322e-01,  2.11876713e-01,
-7.75030537e-02,  7.98882041e-02,  2.35628400e-01, -2.24852042e-02,
-1.19546319e-01, -1.73386867e-01,  2.93828436e-01,  2.26746552e-01,
 5.63046077e-02,  6.06290931e-02, -5.43826765e-02,  1.32706195e-01,
-6.20419777e-02,  1.90111171e-01,  4.38085406e-01,  2.05122990e-01,
-4.16047398e-01,  2.06518229e-01, -3.90824674e-01, -2.17611350e-01,
 9.52445685e-02,  4.15016069e-01,  7.84457578e-02,  7.85001853e-02,
-1.15618673e-02,  7.10505633e-02,  5.36169886e-02,  1.41562551e-01,
 1.11032928e-01,  1.18533183e-01, -1.39915441e-01,  3.34674899e-02,
 2.32121366e-01,  1.70948836e-01,  3.99975375e-02,  6.38553702e-02,
-1.93298355e-01, -1.29083751e-02,  1.25981032e-02,  1.56248014e-01,
-2.12094871e-01,  1.38349026e-01, -2.35132285e-01, -2.56453264e-01,
 5.70088648e-03,  1.34536424e-02, -5.57377563e-02, -2.29205607e-01,
-2.63289205e-01, -4.82871223e-02, -1.68617836e-01,  4.30016030e-01,
 5.85248209e-02, -6.78770288e-02, -2.39626631e-01,  1.81368119e-02,
-1.54426197e-01,  3.87163697e-01, -7.93749719e-02, -1.71727976e-02,
-1.52780176e-01, -2.30744680e-01, -5.53123812e-01,  1.89716407e-03,
 1.60363538e-01,  1.77733497e-01,  3.61694798e-01, -1.14753396e-02,
-2.59484720e-01, -1.99933394e-01, -2.42794502e-02,  2.81821999e-02,
 4.21119963e-01, -5.37283189e-03,  6.11540547e-02, -5.38552609e-02,
 1.44771822e-02, -8.09147041e-02,  6.20014825e-02, -1.23451024e-01,
-2.03445211e-01, -3.87900855e-02,  2.03592749e-02, -7.69814301e-02,
 4.68505813e-01,  3.16653123e-02,  4.83492598e-01, -1.04470523e-01,
 1.37259543e-01, -2.88312921e-01,  6.89411950e-01, -1.41536492e-02,
-1.22252175e-01, -7.34182530e-02, -4.62646837e-02,  9.82929858e-02,
-5.11583230e-04,  1.29572718e-01,  8.30676260e-02,  4.11269142e-03,
 2.94305267e-02,  2.38085866e-01, -1.43578379e-01,  4.87593584e-02,
-4.49000961e-03, -5.22480764e-01, -1.41428463e-01,  5.78546038e-03,
-1.92474667e-01, -2.55107270e-01,  2.84443746e-01, -4.19421194e-01,
-3.55786588e-01, -1.78672748e-02,  2.07312721e-01, -1.13280361e-01,
 2.45002737e-02,  8.45042246e-01, -1.00974406e-01, -7.06628497e-01,
 5.07631450e-02,  3.24373284e-01, -2.06553771e-01, -4.55389472e-01,
 3.00589675e-01, -2.08594559e-01, -1.00133025e-01, -1.30686750e-01,
-2.92236707e-01, -9.00456725e-03, -1.32819300e-01,  3.44185799e-01,
 2.43769790e-01, -1.05104379e-02, -1.43513256e-01, -1.82459491e-01,
-2.28732524e-01, -1.76102600e-02, -4.08876880e-02,  4.79026516e-01,
-1.95616236e-01,  2.26054199e-01, -2.46962786e-01,  3.61557136e-01,
 4.58302246e-01, -2.77596913e-01,  1.47189283e-01,  9.88357909e-01,
-1.35013289e-01, -3.02559947e-01, -1.80385916e-02,  2.80056286e-01,
-2.10238487e-02, -1.83448346e-01, -5.76930688e-02,  2.51114069e-01,
 3.40486106e-03, -1.31474438e-01, -3.43224539e-02, -2.06066234e-01,
-2.12260823e-01, -1.06284752e-01, -3.62569349e-01, -1.39659914e-01,
 1.01348336e-01,  2.86634503e-01,  1.69302519e-01, -1.72505196e-01,
-1.04166186e-01,  5.55291943e-02,  2.62447609e-01,  4.05259142e-01,
-3.89123719e-02, -1.57240686e-01,  8.26479321e-02, -2.95566564e-02,
 6.69893902e-02,  3.18945836e-01, -2.49156354e-01, -2.66697655e-01,
 7.45237096e-01, -1.39797670e-02, -4.47470216e-02, -1.16736663e-01,
 1.64322516e-01, -1.22181021e-01,  4.02251725e-01, -2.16400530e-01,
-2.98510156e-01,  5.44239140e-01,  1.66230958e-01,  8.31999950e-02,
-1.33587079e-02,  1.33362483e-01, -1.53675827e-01, -4.17578121e-01,
 2.64191025e-01,  1.10425776e-01, -3.37924909e-02, -8.19918173e-02,
 3.24348840e-02,  2.00070960e-02, -1.57794902e-01, -3.70847952e-01,
-2.34353191e-01,  2.02264468e-01, -1.27241406e-01, -2.05688704e-01,
-1.44192275e-01, -2.76106650e-01,  2.20716735e-04,  1.08337781e-01,
 8.82983425e-02,  3.47213224e-02,  5.78096229e-01, -3.51337569e-01])
```

In [143…

```python
# FIM Case 1

def fisher_matrix(z, m_err, Omega_M, w, M_n):
    # Define parameters
    params = {'Omega_M': Omega_M, 'Omega_DE': 1-Omega_M-Omega_R-Omega_K, 'w': w, 'M_n': M_n}
    vary_params_case_one = ['Omega_M', 'w', 'M_n']
    param_keys = list(vary_params_case_one)
    n_params = len(param_keys)
    fisher = np.zeros((n_params, n_params))
    h = 1e-3

    for i in range(n_params):
        for j in range(n_params):
            base_params = params.copy() #remove

            # Modify parameters i and j
            base_params[param_keys[i]] += h
            m_th1 = m_th(z, **base_params, Omega_K=0)

            base_params[param_keys[i]] -= 2 * h
            m_th2 = m_th(z, **base_params, Omega_K=0)

            # Reset to original
            base_params[param_keys[i]] += h

            # Central difference for partial derivatives
            dm_dpi = (m_th1 - m_th2) / (2 * h)
```

```
            if i == j:  # Diagonal elements, derivative wrt same parameter
                fisher[i, j] = np.sum((dm_dpi**2) / (m_err**2))
            else:  # Off-diagonal elements, mixed partial derivatives
                base_params[param_keys[j]] += h
                m_th1 = m_th(z, **base_params, Omega_K=0)

                base_params[param_keys[j]] -= 2 * h
                m_th2 = m_th(z, **base_params, Omega_K=0)

                dm_dpj = (m_th1 - m_th2) / (2 * h)
                fisher[i, j] = np.sum((dm_dpi * dm_dpj) / (m_err**2))

    return fisher

F_flat = fisher_matrix(z, m_err, Omega_M, w, M_n)

print("Fisher Matrix (Flat Universe):")
print(F_flat)
```

```
Fisher Matrix (Flat Universe):
[[ 38488.08984695   6056.51401578 -26831.98300159]
 [  6056.51401578   1156.18464212  -3863.50573659]
 [-26831.98300159  -3863.50573659  19434.07720999]]
```

In [144…

```
# Calculation of Chi^2 (With FIM variance)

inv_cov_matrix = np.linalg.inv(data_cov)

def chi_squared(inv_cov, z, m, Omega_M, Omega_DE, w, M_n, Omega_K):

    m_diff = m_diff_vector(z, m, Omega_M, Omega_DE, w, M_n, Omega_K)
    # Calculate the chi-squared value
    chi_squared1 = np.dot(m_diff, np.dot(inv_cov, m_diff))

    return chi_squared1

# Redefine m_diff_vector with outside array for z
# Add data as a func arg
chi_squared(inv_cov_matrix, z, m, Omega_M, Omega_DE, w, M_n, Omega_K)
```

Out[144… 562.5625084965494

In [145…

```
# Calculation of the likelihood for Omega_M, w, and M_n

cov = data_cov
sign, log_det = np.linalg.slogdet(cov)
n = cov.shape[0]

def log_likelihood_w_m_mn(inv_cov, log_det, n, z, m, Omega_M, Omega_DE, w, M_n, Omega_K):
    # Calculate the log of the determinant and the sign of the determinant for the regularized matrix

    log_likelihood4 = -0.5 * n * np.log(2 * np.pi) - 0.5 * log_det - 0.5 * chi_squared(inv_cov, z, m, Omega_M, (

    return log_likelihood4

print(log_likelihood_w_m_mn(inv_cov_matrix, log_det, n, z, m, Omega_M, Omega_DE, w, M_n, Omega_K))
```

```
118.57234070013243
```

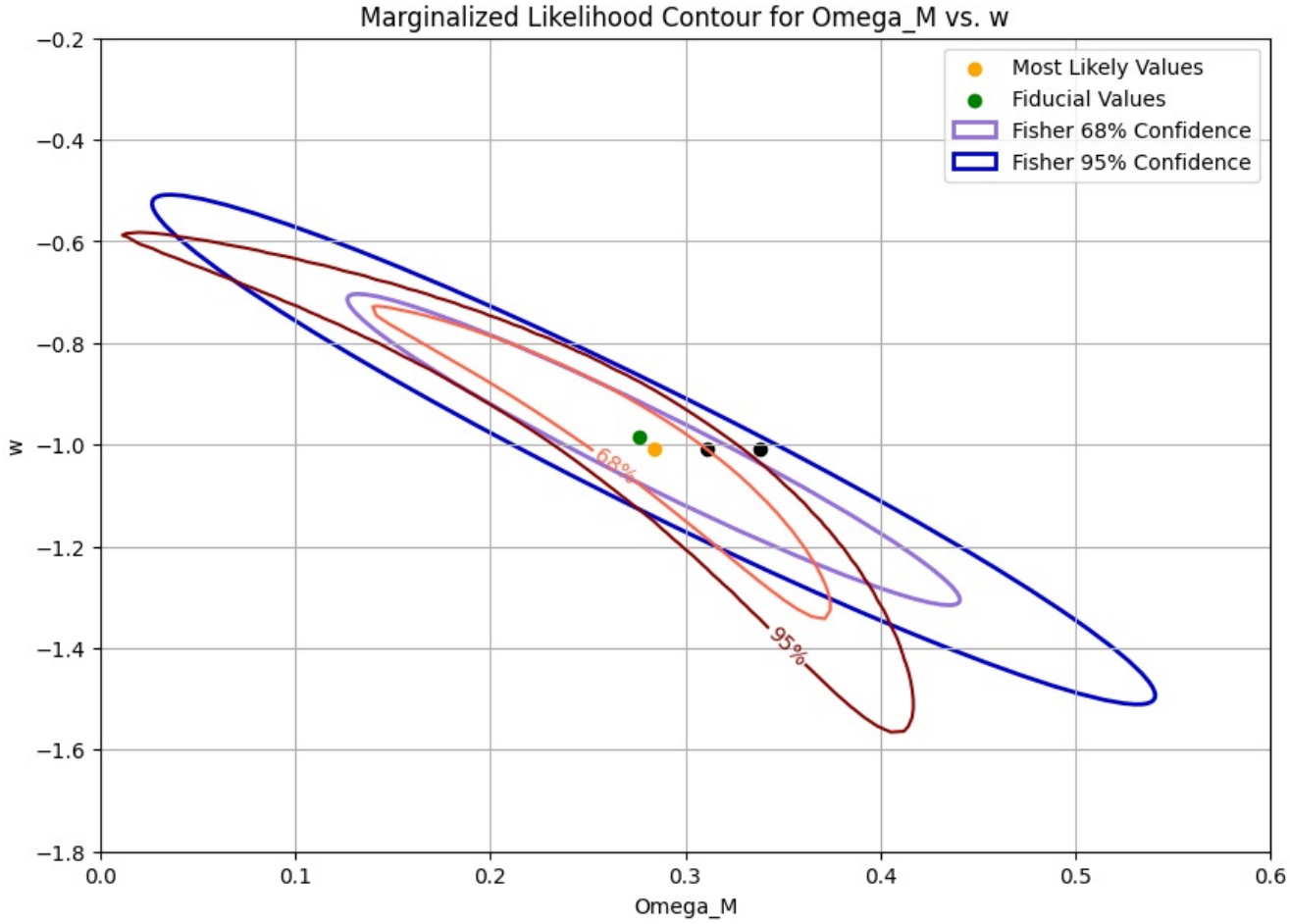
In [158…

```
# Case 1: Varying w, Omega_M, M_n

# Fixed values
fixed_omega_m = 0.276 # From Union 2.1 paper
fixed_w = -0.985 # From Union 2.1 paper
Omega_K = 0.0001
k = 0

# Ranges for Omega_M, w, and H_0
omega_m_min1, omega_m_max1 = 0, 0.5
w_min1, w_max1 = -1.8, -0.5
mn_min1, mn_max1 = 15.5, 16

# Values for Omega_M, w, and H_0
omega_m_range1 = np.linspace(omega_m_min1, omega_m_max1, 75)
w_range1 = np.linspace(w_min1, w_max1, 75)
mn_range1 = np.linspace(mn_min1, mn_max1, 50)

# Meshgrids
omega_m_grid1, w_grid1, mn_grid1 = np.meshgrid(omega_m_range1, w_range1, mn_range1, indexing='ij')

raveled_m_grid1 = np.ravel(omega_m_grid1)
raveled_w_grid1 = np.ravel(w_grid1)
raveled_mn_grid1 = np.ravel(mn_grid1)
```

```
# Calculate likelihoods, normalize
likelihoods_w_m_mn = np.array([log_likelihood_w_m_mn(inv_cov_matrix, log_det, n, z, m, Omega_M, 1 - Omega_M - Or
                        for Omega_M, w, M_n in zip(raveled_m_grid1, raveled_w_grid1, raveled_mn_grid1)])

likelihoods_w_m_mn = likelihoods_w_m_mn.reshape(omega_m_grid1.shape)
for row in likelihoods_w_m_mn: # Resolve bug in abnormal increase in likelihood
    row[0] = row[1]

likelihoods_w_m_mn_norm = np.exp(likelihoods_w_m_mn - np.max(likelihoods_w_m_mn))

# Marginalize over M_n
marg_likelihoods_w_m_mn = np.sum(likelihoods_w_m_mn_norm, axis=2)

# Most likely values
index_max_w_m_mn = np.argmax(marg_likelihoods_w_m_mn)
max_index_w_m_mn = np.unravel_index(index_max_w_m_mn, marg_likelihoods_w_m_mn.shape)

most_likely_omega_m1 = omega_m_grid1[max_index_w_m_mn][0]
most_likely_w1 = w_grid1[max_index_w_m_mn][0]

print(f"Most likely Omega_M: {most_likely_omega_m1}")
print(f"Most likely w: {most_likely_w1}")

# FIM Ellipses
fisher_at_most_likely = fisher_matrix(z, m_err, most_likely_omega_m1, most_likely_w1, M_n)

def plot_fisher_confidence_ellipse(ax, fisher_matrix, mean, confidence_level):
    cov_mat = np.linalg.inv(fisher_matrix)
    marg_cov_mat_mn = np.delete((np.delete(cov_mat, 2, axis=1)), 2, axis=0)

    fixed_fisher_mat_w = np.delete((np.delete(fisher_matrix, 1, axis=1)), 1, axis=0)
    fixed_cov_mat_w = np.linalg.inv(fixed_fisher_mat_w)

    sig_x_2 = marg_cov_mat_mn[0,0]
    sig_xy = marg_cov_mat_mn[1,0]
    sig_y_2 = marg_cov_mat_mn[1,1]

    term_1 = (sig_x_2 + sig_y_2) / 2
    term_2 = np.sqrt(((sig_x_2 - sig_y_2)**2 / 4) + sig_xy**2)

    a = np.sqrt(term_1 + term_2)
    b = np.sqrt(term_1 - term_2)
    theta = np.degrees(np.arctan2((2 * sig_xy), (sig_x_2 - sig_y_2)) / 2)

    if confidence_level == 2.30:
        lbl = 'Fisher 68% Confidence'
        color = 'mediumpurple'
    else:
        lbl = 'Fisher 95% Confidence'
        color = 'mediumblue'

    ellipse = Ellipse(xy=mean, width=2*np.sqrt(confidence_level)*a, height=2*np.sqrt(confidence_level)*b, angle=
    ax.add_patch(ellipse)

    std = np.sqrt(fixed_cov_mat_w[0,0])
    boundary_right_68 = most_likely_omega_m1 + std
    boundary_right_95 = most_likely_omega_m1 + 2*std

    return boundary_right_68, boundary_right_95

# Confidence levels
sorted_likelihoods_w_m_mn = np.sort(marg_likelihoods_w_m_mn.ravel())
cumulative_sum_w_m_mn = np.cumsum(sorted_likelihoods_w_m_mn)
norm_cumulative_sum_w_m_mn = cumulative_sum_w_m_mn / cumulative_sum_w_m_mn[-1]

level_68_w_m_mn = sorted_likelihoods_w_m_mn[np.searchsorted(norm_cumulative_sum_w_m_mn, 1 - 0.68)]
level_95_w_m_mn = sorted_likelihoods_w_m_mn[np.searchsorted(norm_cumulative_sum_w_m_mn, 1 - 0.95)]

# Contour plot
omega_m_grid1_slice = omega_m_grid1[:, :, 0]
w_grid1_slice = w_grid1[:, :, 0]
plt.figure(figsize=(10, 7))
ax = plt.gca()
cp = plt.contour(omega_m_grid1_slice, w_grid1_slice, marg_likelihoods_w_m_mn, levels=[level_95_w_m_mn, level_68_
plt.scatter([most_likely_omega_m1], [most_likely_w1], color='orange', label='Most Likely Values', zorder=5)
plt.clabel(cp, inline=True, fontsize=10, fmt={level_68_w_m_mn: '68%', level_95_w_m_mn: '95%'})
plt.scatter([fixed_omega_m], [fixed_w], color='green', label='Fiducial Values', zorder=5)

# Plot Fisher confidence ellipses
boundary_right_sig1, boundary_right_sig2 = plot_fisher_confidence_ellipse(ax, fisher_at_most_likely, (most_like
plot_fisher_confidence_ellipse(ax, fisher_at_most_likely, (most_likely_omega_m1, most_likely_w1), 6.17)
```

```
plt.scatter(boundary_right_sig1, most_likely_w1, color='black')
plt.scatter(boundary_right_sig2, most_likely_w1, color='black')
plt.title('Marginalized Likelihood Contour for Omega_M vs. w')
plt.xlabel('Omega_M')
plt.ylabel('w')
plt.xlim(0, 0.6)
plt.ylim(-1.8, -0.2)
plt.legend()
plt.grid(True)
plt.show()

# Differences in computed values to Union 2.1 values:
omega_m_diff = np.abs(fixed_omega_m - most_likely_omega_m1)
w_diff = np.abs(fixed_w - most_likely_w1)
print(f"Diff in Omega_M: {omega_m_diff}")
print(f"Diff in w: {w_diff}")

# Percentage error
omega_m_pe = omega_m_diff/fixed_omega_m * 100
w_pe = w_diff/fixed_w * 100
print(f"% err in Omega_M: {omega_m_pe}%")
print(f"% err in w: {w_pe}%")
```

```
Most likely Omega_M: 0.28378378378378377
Most likely w: -1.0094594594594595
```




```
Diff in Omega_M: 0.007783783783783749
Diff in w: 0.024459459459459487
% err in Omega_M: 2.8202115158636767%
% err in w: -2.4831938537522324%
```

In [6]:
```
# Case 1 MCMC

Omega_K = 0.0001

# Pre-generate necessary arrays
def pre_arrays(samples, fish_mat):
    inv_fish = np.linalg.inv(fish_mat)
    array0 = np.random.multivariate_normal(mean=[0,0,0], cov=inv_fish, size=samples)
    array1 = array0[:, 0]
    array2 = array0[:, 1]
    array3 = array0[:, 2]
    array4 = np.ones(samples)
    array5 = np.ones(samples)
    array6 = np.ones(samples)
    array7 = np.ones(samples)
    return array1, array2, array3, array4, array5, array6, array7
```

In [7]:
```python
# Gelman-Rubin R function

def gelman_rubin(chains):
    _, num_samples = chains.shape

    chain_means = np.mean(chains, axis=1)

    # Between-chain variance
    B = num_samples * np.var(chain_means, ddof=1)

    # Within-chain variance
    W = np.mean([np.var(chain, ddof=1) for chain in chains])

    # Estimate of target variance
    V = W * (num_samples - 1) / num_samples + B / num_samples

    # Gelman-Rubin R stat
    R = np.sqrt(V / W)

    return R
```

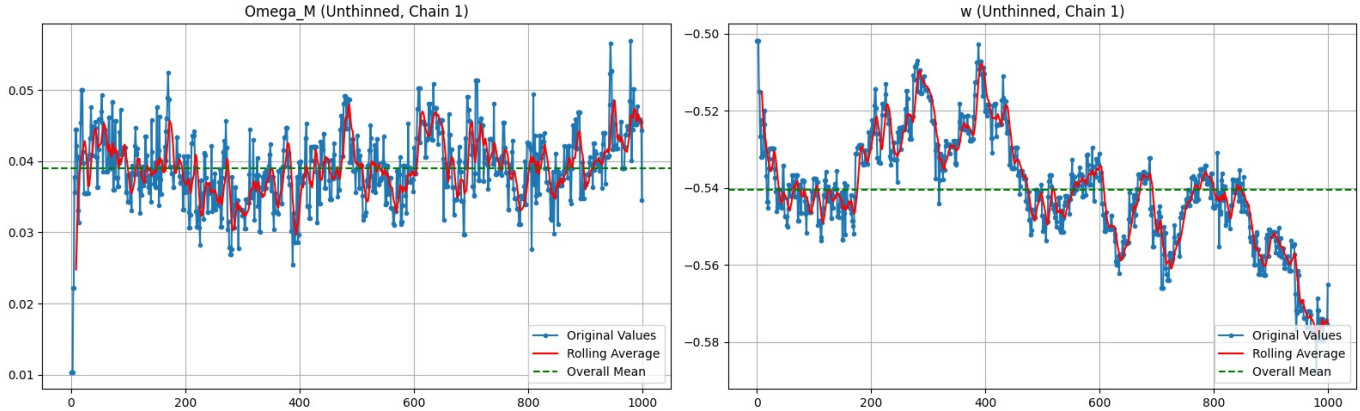
In [8]:
```python
# Step accept/reject

def accept_or_reject(samples, burn_in, fisher_mat, Omega_M, w, M_n):

    steps_Omega_M, steps_w, steps_M_n, Omega_M_values, w_values, M_n_values, log_likelihoods = pre_arrays(sample

    total_accepted_count = 0
    accepted_count = 0

    for i in range(samples):
        # Change to array for 3 parameters
        new_Omega_M = Omega_M + steps_Omega_M[i]
        new_w = w + ( steps_w[i] / 3 )
        new_M_n = M_n + ( steps_M_n[i] / 2 )
        likelihood = 0

        norm_num = np.exp(log_likelihood_w_m_mn(inv_cov_matrix, log_det, n, z, m, new_Omega_M, 1 - new_Omega_M,
        norm_denom = np.exp(log_likelihood_w_m_mn(inv_cov_matrix, log_det, n, z, m, Omega_M, 1 - Omega_M, w, M_

        if new_Omega_M >= 1 or new_Omega_M <= 0 or new_w > 0:
            r = 0
        else:
            r = norm_num / norm_denom

        # Accept or reject the new value
        if np.random.rand() < r:
            Omega_M, w, M_n, likelihood = new_Omega_M, new_w, new_M_n, norm_num
            total_accepted_count += 1

            if i >= burn_in:
                accepted_count += 1
        else:
            likelihood = norm_denom

        # Store values for all steps
        Omega_M_values[i] = Omega_M
        w_values[i] = w
        M_n_values[i] = M_n
        log_likelihoods[i] = np.log(likelihood)

        # Tracking
        if (i + 1) % 50000 == 0 and i >= burn_in:
            interim_percent_accepted = (accepted_count / ((i + 1) - burn_in)) * 100
            print(f"% of accepted values: {np.round(interim_percent_accepted, 3)} % @ {i+1}th iteration")

    # Burn-in
    Omega_M_values = Omega_M_values[burn_in:]
    w_values = w_values[burn_in:]
    M_n_values = M_n_values[burn_in:]
    log_likelihoods = log_likelihoods[burn_in:]

    percent_accepted = (accepted_count / (samples - burn_in)) * 100

    return Omega_M_values, w_values, M_n_values, log_likelihoods, percent_accepted


num_samples = 1000
burn_in_period = 15
selected_thinning_value = 2
fisher_matrix = fisher_at_most_likely
start_Omega_M = 0.28
start_w = -1
```

```
start_M_n = M_n

chain_1_Omega_M_values, chain_1_w_values, chain_1_M_n_values, chain_1_log_likelihoods, chain_1_percent_accepted

# Will deviating the starting value from its true value increase/decrease the % of accepted values?
```

```
% of accepted values: 20.91 % @ 200000th iteration
% of accepted values: 20.925 % @ 250000th iteration
% of accepted values: 21.005 % @ 300000th iteration
% of accepted values: 20.857 % @ 350000th iteration
% of accepted values: 20.89 % @ 400000th iteration
% of accepted values: 20.863 % @ 450000th iteration
% of accepted values: 20.845 % @ 500000th iteration
% of accepted values: 20.888 % @ 550000th iteration
% of accepted values: 20.923 % @ 600000th iteration
% of accepted values: 20.901 % @ 650000th iteration
% of accepted values: 20.858 % @ 700000th iteration
% of accepted values: 20.866 % @ 750000th iteration
% of accepted values: 20.853 % @ 800000th iteration
% of accepted values: 20.842 % @ 850000th iteration
% of accepted values: 20.859 % @ 900000th iteration
% of accepted values: 20.858 % @ 950000th iteration
% of accepted values: 20.883 % @ 1000000th iteration
```

In [9]:
```
# Chain 1 Thinning

thinned_chain_1_Omega_M_values = chain_1_Omega_M_values[::selected_thinning_value]
thinned_chain_1_w_values = chain_1_w_values[::selected_thinning_value]
thinned_chain_1_M_n_values = chain_1_M_n_values[::selected_thinning_value]
thinned_chain_1_log_likelihoods = chain_1_log_likelihoods[::selected_thinning_value]
```

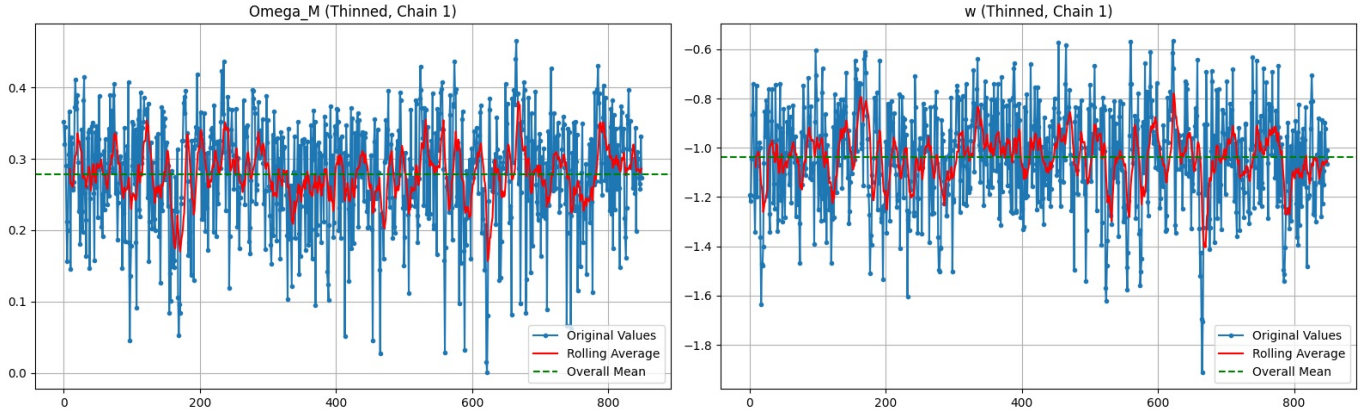
In [ ]:
```
# Chain 1 Unthinned Averages

fig, axs = plt.subplots(1, 2, figsize=(16, 5))  # Adjust the subplots for two plots side by side.

# Plot for Omega_M values
axs[0].plot(chain_1_Omega_M_values, marker='.', label='Original Values')
rolling_avg_Omega_M = pd.Series(chain_1_Omega_M_values).rolling(window=10).mean()
axs[0].plot(rolling_avg_Omega_M, color='red', label='Rolling Average')
mean_value_Omega_M = pd.Series(chain_1_Omega_M_values).mean()
axs[0].axhline(y=mean_value_Omega_M, color='green', linestyle='--', label='Overall Mean')
axs[0].grid()
axs[0].legend(loc='lower right')
axs[0].set_title('Omega_M (Unthinned, Chain 1)')

# Plot for w values
axs[1].plot(chain_1_w_values, marker='.', label='Original Values')
rolling_avg_w = pd.Series(chain_1_w_values).rolling(window=10).mean()
axs[1].plot(rolling_avg_w, color='red', label='Rolling Average')
mean_value_w = pd.Series(chain_1_w_values).mean()
axs[1].axhline(y=mean_value_w, color='green', linestyle='--', label='Overall Mean')
axs[1].grid()
axs[1].legend(loc='lower right')
axs[1].set_title('w (Unthinned, Chain 1)')

plt.tight_layout()
plt.show()

print(f'Unthinned Mean for Omega_M is: {np.round(mean_value_Omega_M, 3)}')
print(f'Unthinned Mean for w is: {np.round(mean_value_w, 3)}')

# Check chart to determine burn in and thinning # Don't use this one though ofc
```




```
Unthinned Mean for Omega_M is: 0.039
Unthinned Mean for w is: -0.54
```

In [11]:
```
# Chain 1 Thinned Averages
```

```
fig, axs = plt.subplots(1, 2, figsize=(16, 5))  # Adjust the subplots for two plots side by side.

# Plot for Omega_M values
axs[0].plot(thinned_chain_1_Omega_M_values, marker='.', label='Original Values')
rolling_avg_Omega_M = pd.Series(thinned_chain_1_Omega_M_values).rolling(window=10).mean()
axs[0].plot(rolling_avg_Omega_M, color='red', label='Rolling Average')
mean_value_Omega_M = pd.Series(thinned_chain_1_Omega_M_values).mean()
axs[0].axhline(y=mean_value_Omega_M, color='green', linestyle='--', label='Overall Mean')
axs[0].grid()
axs[0].legend(loc='lower right')
axs[0].set_title('Omega_M (Thinned, Chain 1)')

# Plot for w values
axs[1].plot(thinned_chain_1_w_values, marker='.', label='Original Values')
rolling_avg_w = pd.Series(thinned_chain_1_w_values).rolling(window=10).mean()
axs[1].plot(rolling_avg_w, color='red', label='Rolling Average')
mean_value_w = pd.Series(thinned_chain_1_w_values).mean()
axs[1].axhline(y=mean_value_w, color='green', linestyle='--', label='Overall Mean')
axs[1].grid()
axs[1].legend(loc='lower right')
axs[1].set_title('w (Thinned, Chain 1)')

plt.tight_layout()
plt.show()

print(f'Thinned Mean for Omega_M is: {np.round(mean_value_Omega_M, 3)}')
print(f'Thinned Mean for w is: {np.round(mean_value_w, 3)}')
```

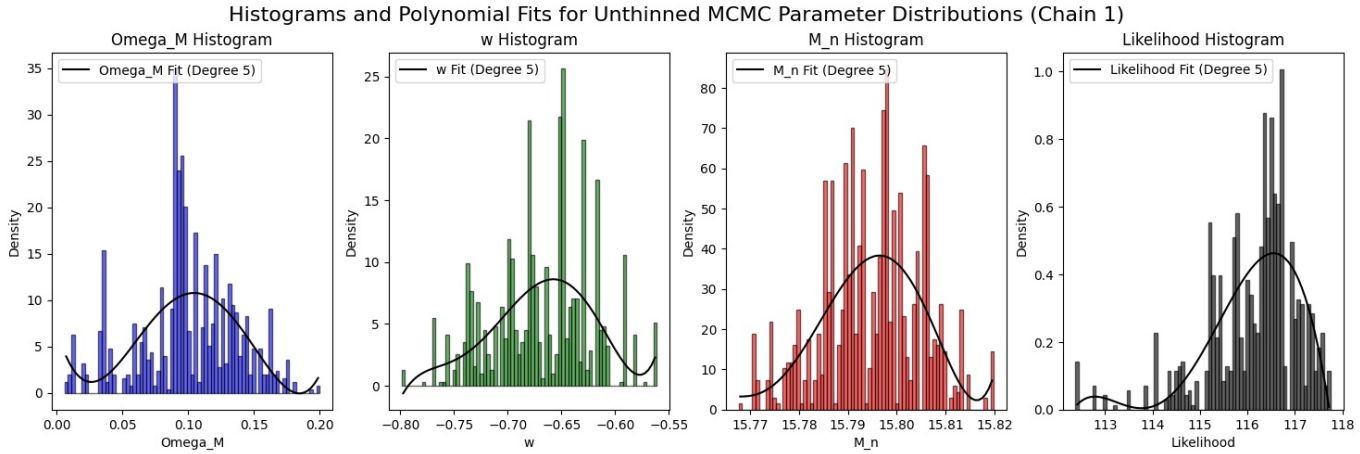



```
Thinned Mean for Omega_M is: 0.278
Thinned Mean for w is: -1.039
```

In [262…

```
# Chain 1 Unthinned Histograms

def plot_histograms_with_fit(degree, degree_max, params, chain_1_all_values):

    # Begin plot
    fig, axes = plt.subplots(1, len(params), figsize=(18, 5))
    num_bins = 75
    colors = ['blue', 'green', 'red', 'black']

    def fit_polynomial_and_find_peak(values, ax, poly_degree, degree_max, param_name, color):
        # Histogram data
        counts, bin_edges = np.histogram(values, bins=num_bins, density=True)
        bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2

        # Prepare array for peak values and fit polynomials
        peak_value_array = np.zeros(degree_max + 1)

        # Fit polynomial and find peaks for each degree
        for i in range(1, degree_max + 1):
            p = Polynomial.fit(bin_centers, counts, deg=i)
            fitted_values = p(bin_centers)
            peak_index = np.argmax(fitted_values)
            peak_value = bin_centers[peak_index]
            peak_value_array[i] = peak_value

        # Peak value Estimations
        avg_peak_value = np.mean(peak_value_array[2:])
        std_peak_value = np.std(peak_value_array[2:])

        # Fit and plot the polynomial of the specified degree (poly_degree) for visualization
        selected_poly = Polynomial.fit(bin_centers, counts, deg=poly_degree)
        selected_fitted_values = selected_poly(bin_centers)

        # Plot histogram and the selected polynomial fit
        ax.hist(values, bins=num_bins, color=color, alpha=0.6, edgecolor='black', density=True)
        ax.plot(bin_centers, selected_fitted_values, color='black', label=f'{param_name} Fit (Degree {poly_degr
        ax.set_title(f'{param_name} Histogram')
```

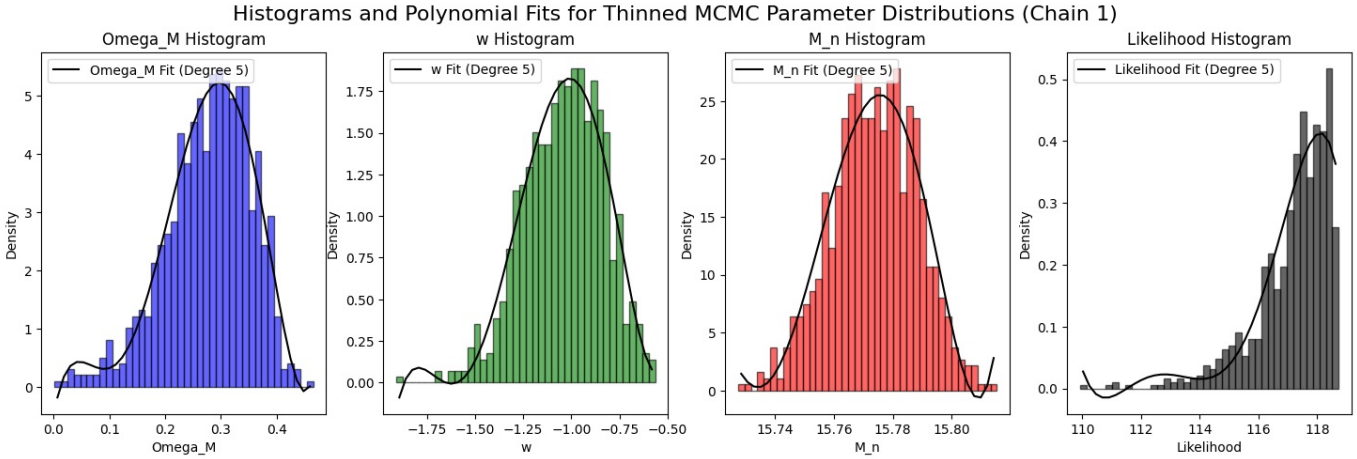
```
        ax.set_xlabel(param_name)
        ax.set_ylabel('Density')
        ax.legend(loc='upper left')

        return avg_peak_value, std_peak_value

    # Plot calls
    converged_values = np.zeros(len(params))
    converged_values_errors = np.zeros(len(params))

    for i, param in enumerate(params):
        converged_values[i], converged_values_errors[i] = fit_polynomial_and_find_peak(chain_1_all_values[i], a

    fig.suptitle('Histograms and Polynomial Fits for Unthinned MCMC Parameter Distributions (Chain 1)', fontsiz

    plt.show()

    return converged_values, converged_values_errors

# Plot histograms and fit polynomial curves for the 3 parameters
poly_degree = 5
max_degree = 12
params = ['Omega_M', 'w', 'M_n', 'Likelihood']
chain_1_all_values = np.array([chain_1_Omega_M_values, chain_1_w_values, chain_1_M_n_values, chain_1_log_likeli

conv_vals, conv_vals_errs = plot_histograms_with_fit(poly_degree, max_degree, params, chain_1_all_values)
```



In [259…

```
# Chain 1 Thinned Histograms

def plot_histograms_with_fit(degree, degree_max, params, chain_1_all_values):

    # Begin plot
    fig, axes = plt.subplots(1, len(params), figsize=(18, 5))
    num_bins = 40
    colors = ['blue', 'green', 'red', 'black']

    def fit_polynomial_and_find_peak(values, ax, poly_degree, degree_max, param_name, color):
        # Histogram data
        counts, bin_edges = np.histogram(values, bins=num_bins, density=True)
        bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2

        # Prepare array for peak values and fit polynomials
        peak_value_array = np.zeros(degree_max + 1)

        # Fit polynomial and find peaks for each degree
        for i in range(1, degree_max + 1):
            p = Polynomial.fit(bin_centers, counts, deg=i)
            fitted_values = p(bin_centers)
            peak_index = np.argmax(fitted_values)
            peak_value = bin_centers[peak_index]
            peak_value_array[i] = peak_value

        # Peak value Estimations
        avg_peak_value = np.mean(peak_value_array[2:])
        std_peak_value = np.std(peak_value_array[2:])

        # Fit and plot the polynomial of the specified degree (poly_degree) for visualization
        selected_poly = Polynomial.fit(bin_centers, counts, deg=poly_degree)
        selected_fitted_values = selected_poly(bin_centers)

        # Plot histogram and the selected polynomial fit
        ax.hist(values, bins=num_bins, color=color, alpha=0.6, edgecolor='black', density=True)
        ax.plot(bin_centers, selected_fitted_values, color='black', label=f'{param_name} Fit (Degree {poly_degr
        ax.set_title(f'{param_name} Histogram')
        ax.set_xlabel(param_name)
```

```
        ax.set_ylabel('Density')
        ax.legend(loc='upper left')

        return avg_peak_value, std_peak_value

    # Plot calls
    converged_values = np.zeros(len(params))
    converged_values_errors = np.zeros(len(params))

    for i, param in enumerate(params):
        converged_values[i], converged_values_errors[i] = fit_polynomial_and_find_peak(chain_1_all_values[i], a

    fig.suptitle('Histograms and Polynomial Fits for Thinned MCMC Parameter Distributions (Chain 1)', fontsize=

    plt.show()

    return converged_values, converged_values_errors

# Plot histograms and fit polynomial curves for the 3 parameters
poly_degree = 5
max_degree = 12
params = ['Omega_M', 'w', 'M_n', 'Likelihood']
chain_1_all_thinned_values = np.array([thinned_chain_1_Omega_M_values, thinned_chain_1_w_values, thinned_chain_

thinned_conv_vals, thinned_conv_vals_errs = plot_histograms_with_fit(poly_degree, max_degree, params, chain_1_a
```



In [14]:
```
# Chain 1 Unthinned Results, Using Polynomial Averaging

for i in range(len(conv_vals)):
    print(f"Estimated Unthinned {params[i]} by Polynomial Averaging: {np.round(conv_vals[i], 3)} ± {np.round(co
```

```
Estimated Unthinned Omega_M by Polynomial Averaging: 0.297 ± 0.013
Estimated Unthinned w by Polynomial Averaging: -1.0 ± 0.027
Estimated Unthinned M_n by Polynomial Averaging: 15.775 ± 0.003
Estimated Unthinned Likelihood by Polynomial Averaging: 118.315 ± 0.258
```

In [15]:
```
# Chain 1 Thinned Results, Using Polynomial Averaging

for i in range(len(thinned_conv_vals)):
    print(f"Estimated Thinned {params[i]} by Polynomial Averaging: {np.round(thinned_conv_vals[i], 3)} ± {np.ro
```

```
Estimated Thinned Omega_M by Polynomial Averaging: 0.299 ± 0.013
Estimated Thinned w by Polynomial Averaging: -0.985 ± 0.029
Estimated Thinned M_n by Polynomial Averaging: 15.774 ± 0.001
Estimated Thinned Likelihood by Polynomial Averaging: 118.286 ± 0.237
```

In [16]:
```
# Chain 1 Thinned/Unthinned Percent Errors, Using Polynomial Averaging

likely_error_thinned_unthinned_diff_chain_1 = np.zeros(len(params))

for i in range(len(params)):
    likely_error_thinned_unthinned_diff_chain_1[i] = ( np.abs(conv_vals[i] - thinned_conv_vals[i]) / thinned_co

    print(f'Percent error (Polynomial Averaging) between thinned and unthinned estimates for {params[i]}: {np.r
```

```
Percent error (Polynomial Averaging) between thinned and unthinned estimates for Omega_M: 0.884 %
Percent error (Polynomial Averaging) between thinned and unthinned estimates for w: -1.539 %
Percent error (Polynomial Averaging) between thinned and unthinned estimates for M_n: 0.003 %
Percent error (Polynomial Averaging) between thinned and unthinned estimates for Likelihood: 0.024 %
```

In [17]:
```
# Chain 1 Unthinned Results, Using Indexing

likely_log_likelihood_chain_1_index = np.argmax(chain_1_log_likelihoods)

tolerance = 0.002
```

```
for i, log_likelihood in enumerate(chain_1_log_likelihoods):
    if abs(log_likelihood - chain_1_log_likelihoods[likely_log_likelihood_chain_1_index]) < tolerance:
        likely_omega_m_chain_1 = chain_1_Omega_M_values[i]
        likely_w_chain_1 = chain_1_w_values[i]
        likely_M_n_chain_1 = chain_1_M_n_values[i]
        likely_log_likelihood_chain_1 = chain_1_log_likelihoods[i]

print(f'The most likely unthinned Omega_M value (Likelihood Indexing) for MCMC Chain 1 is: {np.round(likely_omeg
print(f'The most likely unthinned w value (Likelihood Indexing) for MCMC Chain 1 is: {np.round(likely_w_chain_1
print(f'The most likely unthinned M_n value (Likelihood Indexing) for MCMC Chain 1 is: {np.round(likely_M_n_cha
print(f'The most likely unthinned log likelihood (Likelihood Indexing) for MCMC Chain 1 is: {np.round(likely_lo
```

```
The most likely unthinned Omega_M value (Likelihood Indexing) for MCMC Chain 1 is: 0.284
The most likely unthinned w value (Likelihood Indexing) for MCMC Chain 1 is: -1.018
The most likely unthinned M_n value (Likelihood Indexing) for MCMC Chain 1 is: 15.774
The most likely unthinned log likelihood (Likelihood Indexing) for MCMC Chain 1 is: 118.74

The most likely unthinned w value (Likelihood Indexing) for MCMC Chain 1 is: -1.018
The most likely unthinned M_n value (Likelihood Indexing) for MCMC Chain 1 is: 15.774
The most likely unthinned log likelihood (Likelihood Indexing) for MCMC Chain 1 is: 118.74
```

In [18]:
```
# Chain 1 Thinned Results, Using Indexing

thinned_likely_log_likelihood_chain_1_index = np.argmax(thinned_chain_1_log_likelihoods)

tolerance = 0.002

for i, thinned_log_likelihood in enumerate(thinned_chain_1_log_likelihoods):
    if abs(thinned_log_likelihood - thinned_chain_1_log_likelihoods[thinned_likely_log_likelihood_chain_1_index
        thinned_likely_omega_m_chain_1 = thinned_chain_1_Omega_M_values[i]
        thinned_likely_w_chain_1 = thinned_chain_1_w_values[i]
        thinned_likely_M_n_chain_1 = thinned_chain_1_M_n_values[i]
        thinned_likely_log_likelihood_chain_1 = thinned_chain_1_log_likelihoods[i]

print(f'The most likely thinned Omega_M value (Likelihood Indexing) for MCMC Chain 1 is: {np.round(thinned_like
print(f'The most likely thinned w value (Likelihood Indexing) for MCMC Chain 1 is: {np.round(thinned_likely_w_c
print(f'The most likely thinned M_n value (Likelihood Indexing) for MCMC Chain 1 is: {np.round(thinned_likely_M
print(f'The most likely thinned log likelihood (Likelihood Indexing) for MCMC Chain 1 is: {np.round(thinned_lik
```

```
The most likely thinned Omega_M value (Likelihood Indexing) for MCMC Chain 1 is: 0.276
The most likely thinned w value (Likelihood Indexing) for MCMC Chain 1 is: -0.996
The most likely thinned M_n value (Likelihood Indexing) for MCMC Chain 1 is: 15.774
The most likely thinned log likelihood (Likelihood Indexing) for MCMC Chain 1 is: 118.737
```

In [19]:
```
# Chain 1 Thinned/Unthinned Percent Errors, Using Likelihood Indexing

likely_omega_m_error_thinned_unthinned_diff_chain_1 = ( np.abs(likely_omega_m_chain_1 - thinned_likely_omega_m_
likely_w_error_thinned_unthinned_diff_chain_1 = ( np.abs(likely_w_chain_1 - thinned_likely_w_chain_1) / thinned_
likely_M_n_error_thinned_unthinned_diff_chain_1 = ( np.abs(likely_M_n_chain_1 - thinned_likely_M_n_chain_1) / th
likely_log_likelihood_error_thinned_unthinned_diff_chain_1 = ( np.abs(likely_log_likelihood_chain_1 - thinned_li

print(f'Percent error (Likelihood Indexing) between thinned and unthinned estimates for Omega_M: {np.round(like
print(f'Percent error (Likelihood Indexing) between thinned and unthinned estimates for w: {np.round(likely_w_e
print(f'Percent error (Likelihood Indexing) between thinned and unthinned estimates for M_n: {np.round(likely_M_
print(f'Percent error (Likelihood Indexing) between thinned and unthinned estimates for log_likelihood: {np.rou
```

```
Percent error (Likelihood Indexing) between thinned and unthinned estimates for Omega_M: 2.733 %
Percent error (Likelihood Indexing) between thinned and unthinned estimates for w: -2.257 %
Percent error (Likelihood Indexing) between thinned and unthinned estimates for M_n: 0.006 %
Percent error (Likelihood Indexing) between thinned and unthinned estimates for log_likelihood: 0.003 %
```

In [20]:
```
# Chain 1 Unthinned Results, using Posteriors

# Unthinned Posteriors
posteriors = np.exp(chain_1_log_likelihoods - np.max(chain_1_log_likelihoods))
posteriors /= np.sum(posteriors)

# Parameter Estimations
Omega_M_estimate = np.sum(chain_1_Omega_M_values * posteriors)
w_estimate = np.sum(chain_1_w_values * posteriors)
M_n_estimate = np.sum(chain_1_M_n_values * posteriors)

# Parameter Uncertainties
Omega_M_std = np.sqrt(np.sum(posteriors * (chain_1_Omega_M_values - Omega_M_estimate) ** 2))
w_std = np.sqrt(np.sum(posteriors * (chain_1_w_values - w_estimate) ** 2))
M_n_std = np.sqrt(np.sum(posteriors * (chain_1_M_n_values - M_n_estimate) ** 2))

print(f"Estimated Unthinned Omega_M using Posteriors: {np.round(Omega_M_estimate, 3)} ± {np.round(Omega_M_std, 
print(f"Estimated Unthinned w using Posteriors: {np.round(w_estimate, 3)} ± {np.round(w_std, 3)}")
print(f"Estimated Unthinned M_n using Posteriors: {np.round(M_n_estimate, 3)} ± {np.round(M_n_std, 3)}")
```

```
Estimated Unthinned Omega_M using Posteriors: 0.278 ± 0.055
Estimated Unthinned w using Posteriors: -1.021 ± 0.146
Estimated Unthinned M_n using Posteriors: 15.774 ± 0.01
```

In [21]:
```
# Chain 1 Thinned Results, using Posteriors

# Thinned Posteriors
thinned_posteriors_chain_1 = np.exp(thinned_chain_1_log_likelihoods - np.max(thinned_chain_1_log_likelihoods))
thinned_posteriors_chain_1 /= np.sum(thinned_posteriors_chain_1)

# Parameter Estimations
thinned_Omega_M_estimate_chain_1 = np.sum(thinned_chain_1_Omega_M_values * thinned_posteriors_chain_1)
thinned_w_estimate_chain_1 = np.sum(thinned_chain_1_w_values * thinned_posteriors_chain_1)
thinned_M_n_estimate_chain_1 = np.sum(thinned_chain_1_M_n_values * thinned_posteriors_chain_1)

# Parameter Uncertainties
thinned_Omega_M_std = np.sqrt(np.sum(thinned_posteriors_chain_1 * (thinned_chain_1_Omega_M_values - thinned_Omeg
thinned_w_std = np.sqrt(np.sum(thinned_posteriors_chain_1 * (thinned_chain_1_w_values - thinned_w_estimate_chai
thinned_M_n_std = np.sqrt(np.sum(thinned_posteriors_chain_1 * (thinned_chain_1_M_n_values - thinned_M_n_estimat

print(f"Estimated Thinned Omega_M using Posteriors: {np.round(thinned_Omega_M_estimate_chain_1, 3)} ± {np.round
print(f"Estimated Thinned w using Posteriors: {np.round(thinned_w_estimate_chain_1, 3)} ± {np.round(thinned_w_s
print(f"Estimated Thinned M_n using Posteriors: {np.round(thinned_M_n_estimate_chain_1, 3)} ± {np.round(thinned_
```

```
Estimated Thinned Omega_M using Posteriors: 0.277 ± 0.056
Estimated Thinned w using Posteriors: -1.019 ± 0.15
Estimated Thinned M_n using Posteriors: 15.774 ± 0.011
```

In [22]:
```
# Chain 1 Thinned/Unthinned Percent Errors, Using Posteriors

posterior_likely_omega_m_error_thinned_unthinned_diff_chain_1 = ( np.abs(Omega_M_estimate - thinned_Omega_M_esti
posterior_likely_w_error_thinned_unthinned_diff_chain_1 = ( np.abs(w_estimate - thinned_w_estimate_chain_1) / th
posterior_likely_M_n_error_thinned_unthinned_diff_chain_1 = ( np.abs(M_n_estimate - thinned_M_n_estimate_chain_1

print(f'Percent error (with Posteriors) between thinned and unthinned estimates for Omega_M: {np.round(posterio
print(f'Percent error (with Posteriors) between thinned and unthinned estimates for w: {np.round(posterior_like
print(f'Percent error (with Posteriors) between thinned and unthinned estimates for M_n: {np.round(posterior_li

# Would an increase/decrease in these percent errors mean anything?
```

```
Percent error (with Posteriors) between thinned and unthinned estimates for Omega_M: 0.433 %
Percent error (with Posteriors) between thinned and unthinned estimates for w: -0.117 %
Percent error (with Posteriors) between thinned and unthinned estimates for M_n: 0.0 %
```

In [23]:
```
# Run multiple chains

def run_multiple_chains(num_chains, num_samples, burn_in_period, step_manual, Omega_M, w, M_n):
    all_chains_Omega_M = np.zeros((num_chains, num_samples-burn_in_period))
    all_chains_w = np.zeros((num_chains, num_samples-burn_in_period))
    all_chains_M_n = np.zeros((num_chains, num_samples-burn_in_period))
    all_chains_log_likelihoods = np.zeros((num_chains, num_samples-burn_in_period))
    accepted_percentages = np.zeros(num_chains)

    for i in range(num_chains):
        Omega_M_values, w_values, M_n_values, likelihoods, percent_accepted = accept_or_reject(num_samples, burn

        print(f'Chain {i+1} complete.')

        all_chains_Omega_M[i, :] = Omega_M_values
        all_chains_w[i, :] = w_values
        all_chains_M_n[i, :] = M_n_values
        all_chains_log_likelihoods[i, :] = likelihoods
        accepted_percentages[i] = percent_accepted

    # Compute R
    rhat_Omega_M = gelman_rubin(all_chains_Omega_M)
    rhat_w = gelman_rubin(all_chains_w)
    rhat_M_n = gelman_rubin(all_chains_M_n)

    return (all_chains_Omega_M, all_chains_w, all_chains_M_n, all_chains_log_likelihoods,
            rhat_Omega_M, rhat_w, rhat_M_n, accepted_percentages)


# Run multiple chains, checking convergence
num_chains = 7
all_chains_Omega_M, all_chains_w, all_chains_M_n, all_chains_log_likelihoods, r_Omega_M, r_w, r_M_n, accepted_p
run_multiple_chains(num_chains, num_samples, burn_in_period, fisher_matrix, start_Omega_M, start_w, start_M_n)
```

```
% of accepted values: 20.68 % @ 200000th iteration
% of accepted values: 20.892 % @ 250000th iteration
% of accepted values: 20.854 % @ 300000th iteration
% of accepted values: 21.017 % @ 350000th iteration
% of accepted values: 20.945 % @ 400000th iteration
% of accepted values: 21.046 % @ 450000th iteration
% of accepted values: 21.098 % @ 500000th iteration
% of accepted values: 21.071 % @ 550000th iteration
% of accepted values: 21.041 % @ 600000th iteration
% of accepted values: 21.071 % @ 650000th iteration
```

```
% of accepted values: 21.056 % @ 700000th iteration
% of accepted values: 21.032 % @ 750000th iteration
% of accepted values: 21.032 % @ 800000th iteration
% of accepted values: 20.965 % @ 850000th iteration
% of accepted values: 20.958 % @ 900000th iteration
% of accepted values: 20.952 % @ 950000th iteration
% of accepted values: 20.958 % @ 1000000th iteration
Chain 1 complete.
% of accepted values: 20.722 % @ 200000th iteration
% of accepted values: 20.769 % @ 250000th iteration
% of accepted values: 20.675 % @ 300000th iteration
% of accepted values: 20.787 % @ 350000th iteration
% of accepted values: 20.705 % @ 400000th iteration
% of accepted values: 20.776 % @ 450000th iteration
% of accepted values: 20.781 % @ 500000th iteration
% of accepted values: 20.801 % @ 550000th iteration
% of accepted values: 20.89 % @ 600000th iteration
% of accepted values: 20.856 % @ 650000th iteration
% of accepted values: 20.905 % @ 700000th iteration
% of accepted values: 20.928 % @ 750000th iteration
% of accepted values: 20.953 % @ 800000th iteration
% of accepted values: 20.935 % @ 850000th iteration
% of accepted values: 20.926 % @ 900000th iteration
% of accepted values: 20.935 % @ 950000th iteration
% of accepted values: 20.943 % @ 1000000th iteration
Chain 2 complete.
% of accepted values: 20.674 % @ 200000th iteration
% of accepted values: 20.696 % @ 250000th iteration
% of accepted values: 20.665 % @ 300000th iteration
% of accepted values: 20.788 % @ 350000th iteration
% of accepted values: 20.874 % @ 400000th iteration
% of accepted values: 20.905 % @ 450000th iteration
% of accepted values: 20.97 % @ 500000th iteration
% of accepted values: 20.977 % @ 550000th iteration
% of accepted values: 20.987 % @ 600000th iteration
% of accepted values: 21.002 % @ 650000th iteration
% of accepted values: 21.0 % @ 700000th iteration
% of accepted values: 21.004 % @ 750000th iteration
% of accepted values: 20.997 % @ 800000th iteration
% of accepted values: 20.971 % @ 850000th iteration
% of accepted values: 20.978 % @ 900000th iteration
% of accepted values: 20.96 % @ 950000th iteration
% of accepted values: 20.962 % @ 1000000th iteration
Chain 3 complete.
% of accepted values: 20.906 % @ 200000th iteration
% of accepted values: 20.907 % @ 250000th iteration
% of accepted values: 20.897 % @ 300000th iteration
% of accepted values: 20.954 % @ 350000th iteration
% of accepted values: 20.766 % @ 400000th iteration
% of accepted values: 20.818 % @ 450000th iteration
% of accepted values: 20.81 % @ 500000th iteration
% of accepted values: 20.874 % @ 550000th iteration
% of accepted values: 20.908 % @ 600000th iteration
% of accepted values: 20.885 % @ 650000th iteration
% of accepted values: 20.885 % @ 700000th iteration
% of accepted values: 20.828 % @ 750000th iteration
% of accepted values: 20.86 % @ 800000th iteration
% of accepted values: 20.892 % @ 850000th iteration
% of accepted values: 20.897 % @ 900000th iteration
% of accepted values: 20.891 % @ 950000th iteration
% of accepted values: 20.871 % @ 1000000th iteration
Chain 4 complete.
% of accepted values: 21.38 % @ 200000th iteration
% of accepted values: 21.032 % @ 250000th iteration
% of accepted values: 21.136 % @ 300000th iteration
% of accepted values: 21.176 % @ 350000th iteration
% of accepted values: 21.13 % @ 400000th iteration
% of accepted values: 21.15 % @ 450000th iteration
% of accepted values: 21.124 % @ 500000th iteration
% of accepted values: 21.166 % @ 550000th iteration
% of accepted values: 21.163 % @ 600000th iteration
% of accepted values: 21.133 % @ 650000th iteration
% of accepted values: 21.104 % @ 700000th iteration
% of accepted values: 21.12 % @ 750000th iteration
% of accepted values: 21.1 % @ 800000th iteration
% of accepted values: 21.113 % @ 850000th iteration
% of accepted values: 21.109 % @ 900000th iteration
% of accepted values: 21.116 % @ 950000th iteration
% of accepted values: 21.091 % @ 1000000th iteration
Chain 5 complete.
% of accepted values: 21.112 % @ 200000th iteration
% of accepted values: 20.99 % @ 250000th iteration
% of accepted values: 21.004 % @ 300000th iteration
```

```
% of accepted values: 20.968 % @ 350000th iteration
% of accepted values: 20.968 % @ 400000th iteration
% of accepted values: 20.959 % @ 450000th iteration
% of accepted values: 20.985 % @ 500000th iteration
% of accepted values: 21.034 % @ 550000th iteration
% of accepted values: 21.084 % @ 600000th iteration
% of accepted values: 21.064 % @ 650000th iteration
% of accepted values: 21.057 % @ 700000th iteration
% of accepted values: 21.038 % @ 750000th iteration
% of accepted values: 21.021 % @ 800000th iteration
% of accepted values: 21.064 % @ 850000th iteration
% of accepted values: 21.098 % @ 900000th iteration
% of accepted values: 21.054 % @ 950000th iteration
% of accepted values: 21.049 % @ 1000000th iteration
Chain 6 complete.
% of accepted values: 21.384 % @ 200000th iteration
% of accepted values: 21.104 % @ 250000th iteration
% of accepted values: 21.123 % @ 300000th iteration
% of accepted values: 21.092 % @ 350000th iteration
% of accepted values: 21.012 % @ 400000th iteration
% of accepted values: 21.108 % @ 450000th iteration
% of accepted values: 21.089 % @ 500000th iteration
% of accepted values: 21.082 % @ 550000th iteration
% of accepted values: 21.013 % @ 600000th iteration
% of accepted values: 21.012 % @ 650000th iteration
% of accepted values: 21.019 % @ 700000th iteration
% of accepted values: 21.009 % @ 750000th iteration
% of accepted values: 20.955 % @ 800000th iteration
% of accepted values: 20.932 % @ 850000th iteration
% of accepted values: 20.946 % @ 900000th iteration
% of accepted values: 20.959 % @ 950000th iteration
% of accepted values: 20.938 % @ 1000000th iteration
Chain 7 complete.
Accepted value percentage for chain 1: 20.957529411764707 %
Accepted value percentage for chain 2: 20.942588235294117 %
Accepted value percentage for chain 3: 20.962235294117647 %
Accepted value percentage for chain 4: 20.87070588235294 %
Accepted value percentage for chain 5: 21.091058823529412 %
Accepted value percentage for chain 6: 21.049294117647058 %
Accepted value percentage for chain 7: 20.938470588235294 %
Gelman-Rubin R for Omega_M: 1.001
Gelman-Rubin R for w: 1.001
Gelman-Rubin R for M_n: 1.0
```

In [150…

```python
# Accepted value %s for each chain
for i, accepted_percent in enumerate(accepted_percentages):
    print(f"Accepted value % for chain {i+1}: {accepted_percent} %")

# Average accepted value %s
print(f'Average accepted value % for all chains: {np.mean(accepted_percentages)} %')

# Gelman-Rubin R stats
print('')
print(f"Gelman-Rubin R for Omega_M: {np.round(r_Omega_M, 5)}")
print(f"Gelman-Rubin R for w: {np.round(r_w, 5)}")
print(f"Gelman-Rubin R for M_n: {np.round(r_M_n, 5)}")
```

```
Accepted value % for chain 1: 20.957529411764707 %
Accepted value % for chain 2: 20.942588235294117 %
Accepted value % for chain 3: 20.962235294117647 %
Accepted value % for chain 4: 20.87070588235294 %
Accepted value % for chain 5: 21.091058823529412 %
Accepted value % for chain 6: 21.049294117647058 %
Accepted value % for chain 7: 20.938470588235294 %
Average accepted value % for all chains: 20.97312605042017 %

Gelman-Rubin R for Omega_M: 1.00079
Gelman-Rubin R for w: 1.00094
Gelman-Rubin R for M_n: 1.00043
```

In [86]:

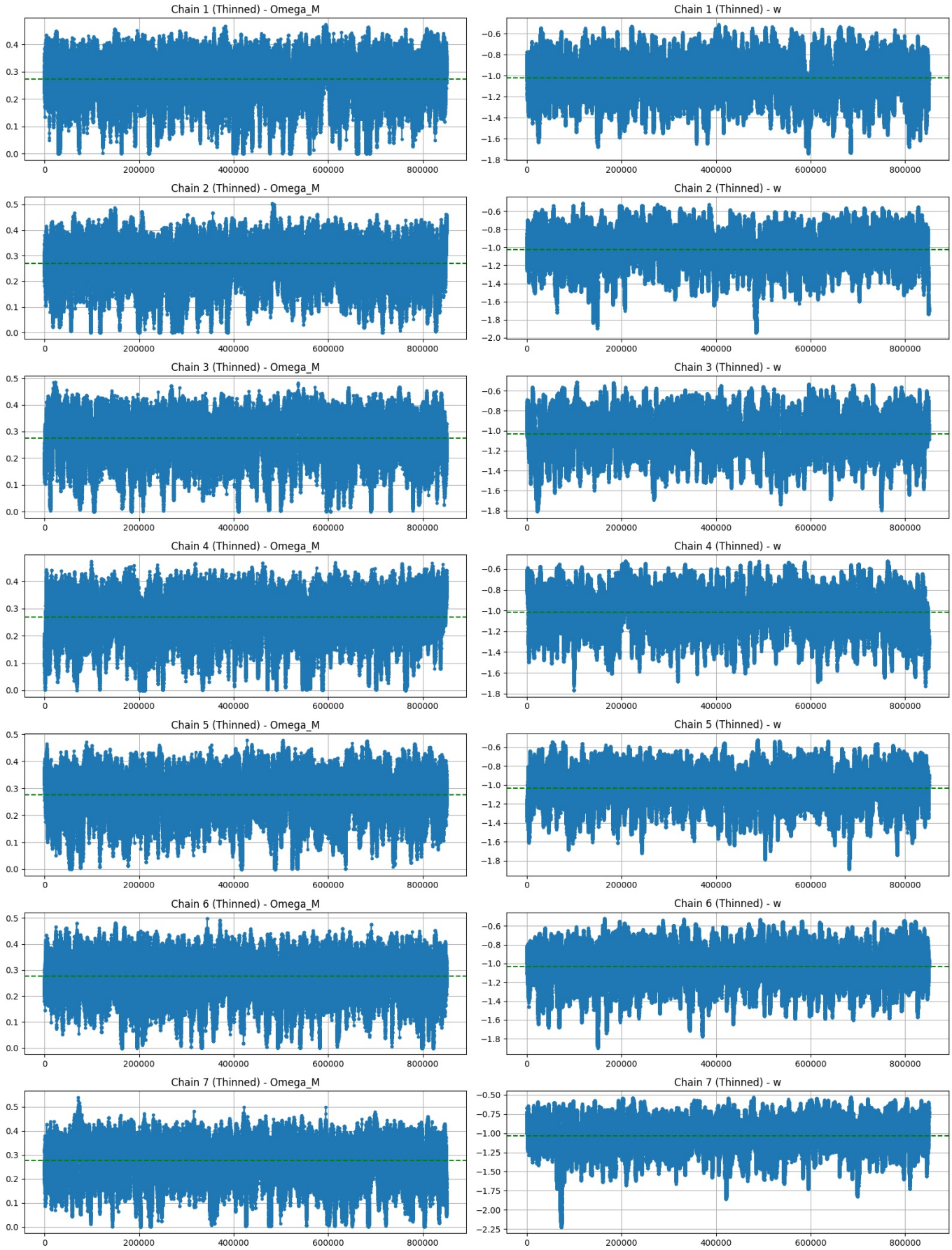
```python
# Thinning All Chains

thinned_all_chains_Omega_M_values = all_chains_Omega_M[:, ::selected_thinning_value]
thinned_all_chains_w_values = all_chains_w[:, ::selected_thinning_value]
thinned_all_chains_M_n_values = all_chains_M_n[:, ::selected_thinning_value]
thinned_all_chains_log_likelihood_values = all_chains_log_likelihoods[:, ::selected_thinning_value]
```

In [87]:

```python
# Unthinned Averages

fig, axs = plt.subplots(num_chains, 2, figsize=(16, num_chains * 3))

mean_values_Omega_M = np.zeros(num_chains)
std_values_Omega_M = np.zeros(num_chains)
mean_values_w = np.zeros(num_chains)
```

```
std_values_w = np.zeros(num_chains)
mean_values_M_n = np.zeros(num_chains)
std_values_M_n = np.zeros(num_chains)

# Plotting
for i in range(len(all_chains_Omega_M)):

    # Omega_M
    axs[i, 0].plot(all_chains_Omega_M[i], marker='.')
    mean_value_Omega_M = np.mean(all_chains_Omega_M[i])
    std_value_Omega_M = np.std(all_chains_Omega_M[i])
    axs[i, 0].axhline(y=mean_value_Omega_M, color='green', linestyle='--', label='Overall Mean')
    axs[i, 0].set_title(f'Chain {i+1} (Unthinned) - Omega_M')
    axs[i, 0].grid()

    # w
    axs[i, 1].plot(all_chains_w[i], marker='.')
    mean_value_w = np.mean(all_chains_w[i])
    std_value_w = np.std(all_chains_w[i])
    axs[i, 1].axhline(y=mean_value_w, color='green', linestyle='--', label='Overall Mean')
    axs[i, 1].set_title(f'Chain {i+1} (Thinned) - w')
    axs[i, 1].grid()

    # M_n
    mean_value_M_n = np.mean(all_chains_M_n[i])
    std_value_M_n = np.std(all_chains_M_n[i])

    # Print mean and standard deviation for each chain
    print(f'Chain {i+1} Mean Omega_M value: {np.round(mean_value_Omega_M, 3)}, ± {np.round(std_value_Omega_M, 3
    print(f'Chain {i+1} Mean w value: {np.round(mean_value_w, 3)}, ± {np.round(std_value_w, 3)}')
    print('')

    # Store values in arrays
    mean_values_Omega_M[i] = mean_value_Omega_M
    std_values_Omega_M[i] = std_value_Omega_M

    mean_values_w[i] = mean_value_w
    std_values_w[i] = std_value_w

    mean_values_M_n[i] = mean_value_M_n
    std_values_M_n[i] = std_value_M_n

# Mean and Std.
mean_mean_value_Omega_M = np.mean(mean_values_Omega_M)
mean_mean_value_w = np.mean(mean_values_w)
mean_mean_value_M_n = np.mean(mean_values_M_n)

overall_std_Omega_M = np.mean(std_values_Omega_M)
overall_std_w = np.mean(std_values_w)
overall_std_M_n = np.mean(std_values_M_n)

# Overall means and stds
print('')
print(f'Mean Omega_M value across {num_chains} unthinned chains: {np.round(mean_mean_value_Omega_M, 3)} ± {np.r
print(f'Mean w value across {num_chains} unthinned chains: {np.round(mean_mean_value_w, 3)} ± {np.round(overall_

plt.tight_layout()
plt.show()
```

```
Chain 1 Mean Omega_M value: 0.272, ± 0.078
Chain 1 Mean w value: -1.022, ± 0.196

Chain 2 Mean Omega_M value: 0.271, ± 0.079
Chain 2 Mean w value: -1.021, ± 0.208

Chain 3 Mean Omega_M value: 0.276, ± 0.078
Chain 3 Mean w value: -1.035, ± 0.206

Chain 4 Mean Omega_M value: 0.27, ± 0.078
Chain 4 Mean w value: -1.014, ± 0.194

Chain 5 Mean Omega_M value: 0.276, ± 0.074
Chain 5 Mean w value: -1.032, ± 0.198

Chain 6 Mean Omega_M value: 0.277, ± 0.074
Chain 6 Mean w value: -1.033, ± 0.197

Chain 7 Mean Omega_M value: 0.277, ± 0.078
Chain 7 Mean w value: -1.038, ± 0.216


Mean Omega_M value across 7 thinned chains: 0.274 ± 0.077
Mean w value across 7 thinned chains: -1.028 ± 0.202
```

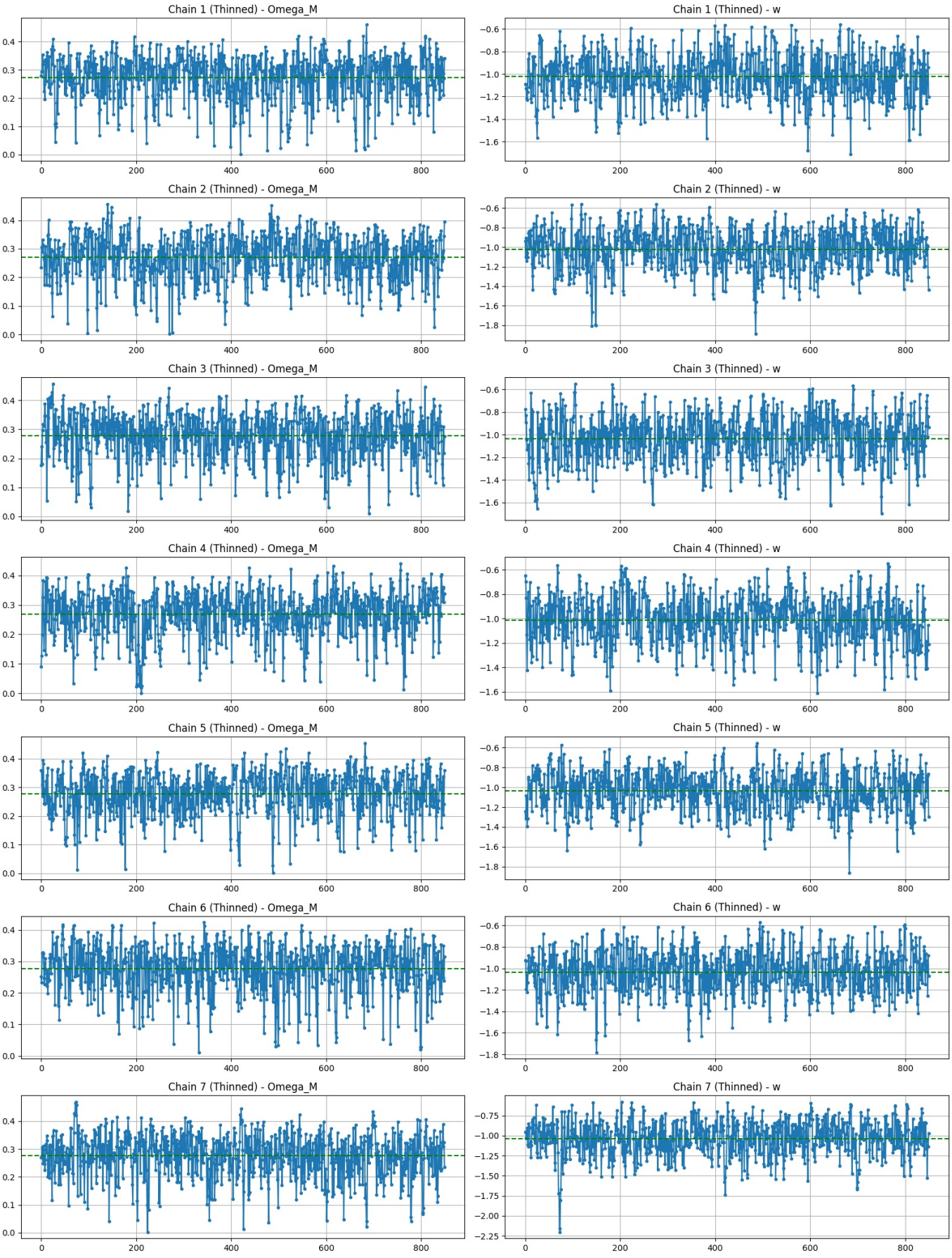


In [125…

```
# Thinned Averages

fig, axs = plt.subplots(num_chains, 2, figsize=(16, num_chains * 3))

thinned_mean_values_Omega_M = np.zeros(num_chains)
thinned_std_values_Omega_M = np.zeros(num_chains)
thinned_mean_values_w = np.zeros(num_chains)
thinned_std_values_w = np.zeros(num_chains)
thinned_mean_values_M_n = np.zeros(num_chains)
thinned_std_values_M_n = np.zeros(num_chains)

# Plotting
```

```python
for i in range(len(thinned_all_chains_Omega_M_values)):

    # Omega_M
    axs[i, 0].plot(thinned_all_chains_Omega_M_values[i], marker='.')
    thinned_mean_value_Omega_M = np.mean(thinned_all_chains_Omega_M_values[i])
    thinned_std_value_Omega_M = np.std(thinned_all_chains_Omega_M_values[i])
    axs[i, 0].axhline(y=thinned_mean_value_Omega_M, color='green', linestyle='--', label='Overall Mean')
    axs[i, 0].set_title(f'Chain {i+1} (Thinned) - Omega_M')
    axs[i, 0].grid()

    # w
    axs[i, 1].plot(thinned_all_chains_w_values[i], marker='.')
    thinned_mean_value_w = np.mean(thinned_all_chains_w_values[i])
    thinned_std_value_w = np.std(thinned_all_chains_w_values[i])
    axs[i, 1].axhline(y=thinned_mean_value_w, color='green', linestyle='--', label='Overall Mean')
    axs[i, 1].set_title(f'Chain {i+1} (Thinned) - w')
    axs[i, 1].grid()

    # M_n
    thinned_mean_value_M_n = np.mean(thinned_all_chains_M_n_values[i])
    thinned_std_value_M_n = np.std(thinned_all_chains_M_n_values[i])

    # Print mean and standard deviation for each chain
    print(f'Chain {i+1} Mean Omega_M value: {np.round(thinned_mean_value_Omega_M, 3)}, ± {np.round(thinned_std_
    print(f'Chain {i+1} Mean w value: {np.round(thinned_mean_value_w, 3)}, ± {np.round(thinned_std_value_w, 3)}
    print('')

    # Store values in arrays
    thinned_mean_values_Omega_M[i] = thinned_mean_value_Omega_M
    thinned_std_values_Omega_M[i] = thinned_std_value_Omega_M

    thinned_mean_values_w[i] = thinned_mean_value_w
    thinned_std_values_w[i] = thinned_std_value_w

    thinned_mean_values_M_n[i] = thinned_mean_value_M_n
    thinned_std_values_M_n[i] = thinned_std_value_M_n

# Mean and Std.
thinned_mean_mean_value_Omega_M = np.mean(thinned_mean_values_Omega_M)
thinned_mean_mean_value_w = np.mean(thinned_mean_values_w)
thinned_mean_mean_value_M_n = np.mean(thinned_mean_values_M_n)

thinned_overall_std_Omega_M = np.mean(thinned_std_values_Omega_M)
thinned_overall_std_w = np.mean(thinned_std_values_w)
thinned_overall_std_M_n = np.mean(thinned_std_values_M_n)

# Overall means and stds
print('')
print(f'Mean Omega_M value across {num_chains} thinned chains: {np.round(thinned_mean_mean_value_Omega_M, 3)} ±
print(f'Mean w value across {num_chains} thinned chains: {np.round(thinned_mean_mean_value_w, 3)} ± {np.round(t

plt.tight_layout()
plt.show()
```

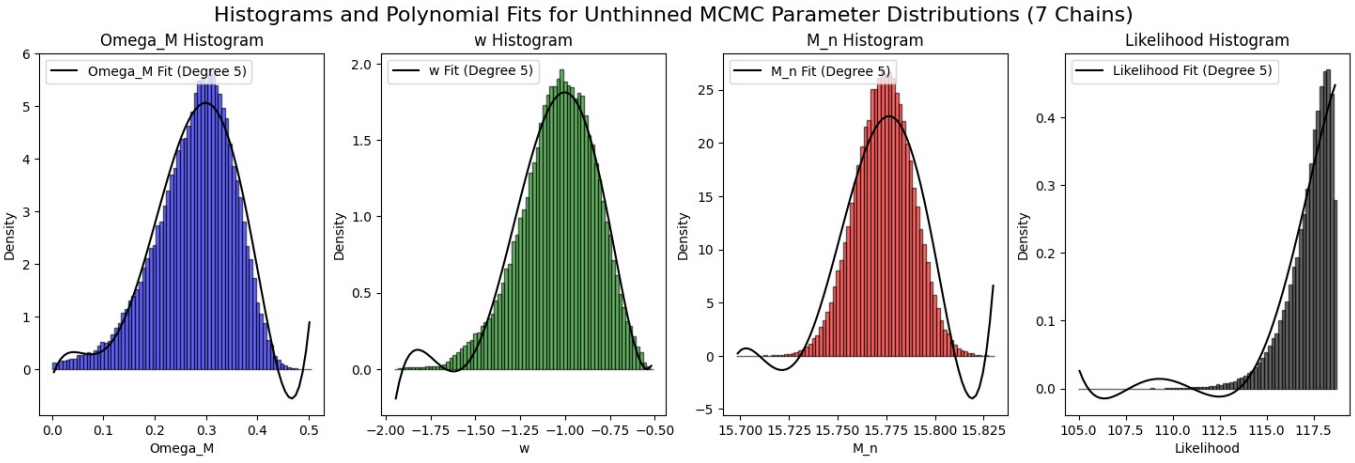
```
Chain 1 Mean Omega_M value: 0.273, ± 0.078
Chain 1 Mean w value: -1.023, ± 0.194

Chain 2 Mean Omega_M value: 0.271, ± 0.078
Chain 2 Mean w value: -1.022, ± 0.208

Chain 3 Mean Omega_M value: 0.278, ± 0.078
Chain 3 Mean w value: -1.035, ± 0.207

Chain 4 Mean Omega_M value: 0.269, ± 0.078
Chain 4 Mean w value: -1.012, ± 0.194

Chain 5 Mean Omega_M value: 0.277, ± 0.074
Chain 5 Mean w value: -1.034, ± 0.194

Chain 6 Mean Omega_M value: 0.277, ± 0.075
Chain 6 Mean w value: -1.034, ± 0.196

Chain 7 Mean Omega_M value: 0.276, ± 0.077
Chain 7 Mean w value: -1.034, ± 0.214


Mean Omega_M value across 7 thinned chains: 0.274 ± 0.077
Mean w value across 7 thinned chains: -1.028 ± 0.201
```



In [126…

```
# Unthinned Histograms All Chains

def plot_histograms_with_fit(degree, degree_max, params, chain_1_all_values):

    # Begin plot
    fig, axes = plt.subplots(1, len(params), figsize=(18, 5))
    num_bins = 75
```

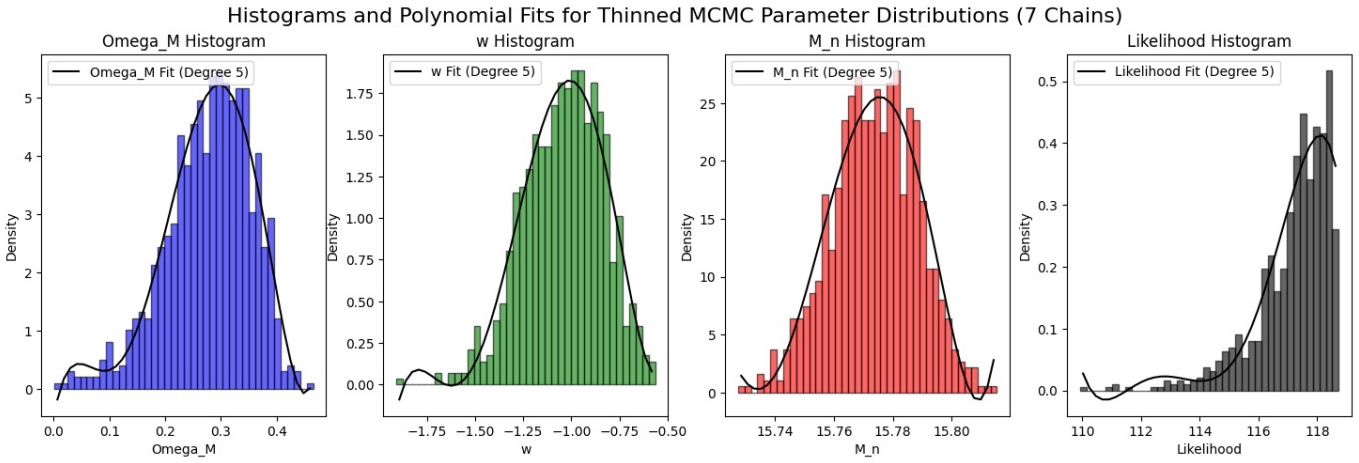
```
    colors = ['blue', 'green', 'red', 'black']

    def fit_polynomial_and_find_peak(values, ax, poly_degree, degree_max, param_name, color):
        # Histogram data
        counts, bin_edges = np.histogram(values, bins=num_bins, density=True)
        bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2

        # Prepare array for peak values and fit polynomials
        peak_value_array = np.zeros(degree_max + 1)

        # Fit polynomial and find peaks for each degree
        for i in range(1, degree_max + 1):
            p = Polynomial.fit(bin_centers, counts, deg=i)
            fitted_values = p(bin_centers)
            peak_index = np.argmax(fitted_values)
            peak_value = bin_centers[peak_index]
            peak_value_array[i] = peak_value

        # Peak value Estimations
        avg_peak_value = np.mean(peak_value_array[2:])
        std_peak_value = np.std(peak_value_array[2:])

        # Fit and plot the polynomial of the specified degree (poly_degree) for visualization
        selected_poly = Polynomial.fit(bin_centers, counts, deg=poly_degree)
        selected_fitted_values = selected_poly(bin_centers)

        # Plot histogram and the selected polynomial fit
        ax.hist(values, bins=num_bins, color=color, alpha=0.6, edgecolor='black', density=True)
        ax.plot(bin_centers, selected_fitted_values, color='black', label=f'{param_name} Fit (Degree {poly_degr
        ax.set_title(f'{param_name} Histogram')
        ax.set_xlabel(param_name)
        ax.set_ylabel('Density')
        ax.legend(loc='upper left')

        return avg_peak_value, std_peak_value

    # Plot calls
    converged_values = np.zeros(len(params))
    converged_values_errors = np.zeros(len(params))

    for i, param in enumerate(params):
        converged_values[i], converged_values_errors[i] = fit_polynomial_and_find_peak(chain_1_all_values[i], a

    fig.suptitle(f'Histograms and Polynomial Fits for Unthinned MCMC Parameter Distributions ({num_chains} Chai

    plt.show()

    return converged_values, converged_values_errors

# Plot histograms and fit polynomial curves for the 3 parameters
poly_degree = 5
max_degree = 12
params = ['Omega_M', 'w', 'M_n', 'Likelihood']
chain_1_all_values = np.array([chain_1_Omega_M_values, chain_1_w_values, chain_1_M_n_values, chain_1_log_likeli

conv_vals, conv_vals_errs = plot_histograms_with_fit(poly_degree, max_degree, params, chain_1_all_values)
```



In [127…

```
# Thinned Histograms All Chains

def plot_histograms_with_fit(degree, degree_max, params, chain_1_all_values):

    # Begin plot
    fig, axes = plt.subplots(1, len(params), figsize=(18, 5))
    num_bins = 40
    colors = ['blue', 'green', 'red', 'black']
```

```
    def fit_polynomial_and_find_peak(values, ax, poly_degree, degree_max, param_name, color):
        # Histogram data
        counts, bin_edges = np.histogram(values, bins=num_bins, density=True)
        bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2

        # Prepare array for peak values and fit polynomials
        peak_value_array = np.zeros(degree_max + 1)

        # Fit polynomial and find peaks for each degree
        for i in range(1, degree_max + 1):
            p = Polynomial.fit(bin_centers, counts, deg=i)
            fitted_values = p(bin_centers)
            peak_index = np.argmax(fitted_values)
            peak_value = bin_centers[peak_index]
            peak_value_array[i] = peak_value

        # Peak value Estimations
        avg_peak_value = np.mean(peak_value_array[2:])
        std_peak_value = np.std(peak_value_array[2:])

        # Fit and plot the polynomial of the specified degree (poly_degree) for visualization
        selected_poly = Polynomial.fit(bin_centers, counts, deg=poly_degree)
        selected_fitted_values = selected_poly(bin_centers)

        # Plot histogram and the selected polynomial fit
        ax.hist(values, bins=num_bins, color=color, alpha=0.6, edgecolor='black', density=True)
        ax.plot(bin_centers, selected_fitted_values, color='black', label=f'{param_name} Fit (Degree {poly_degr
        ax.set_title(f'{param_name} Histogram')
        ax.set_xlabel(param_name)
        ax.set_ylabel('Density')
        ax.legend(loc='upper left')

        return avg_peak_value, std_peak_value

    # Plot calls
    converged_values = np.zeros(len(params))
    converged_values_errors = np.zeros(len(params))

    for i, param in enumerate(params):
        converged_values[i], converged_values_errors[i] = fit_polynomial_and_find_peak(chain_1_all_values[i], a

    fig.suptitle(f'Histograms and Polynomial Fits for Thinned MCMC Parameter Distributions ({num_chains} Chains

    plt.show()

    return converged_values, converged_values_errors

# Plot histograms and fit polynomial curves for the 3 parameters
poly_degree = 5
max_degree = 12
params = ['Omega_M', 'w', 'M_n', 'Likelihood']
thinned_chain_1_all_values = np.array([thinned_chain_1_Omega_M_values, thinned_chain_1_w_values, thinned_chain_

thinned_conv_vals, thinned_conv_vals_errs = plot_histograms_with_fit(poly_degree, max_degree, params, thinned_cl
```



In [128…

```
# All Chains Unthinned Results, Raw Samples

print(f'Mean Omega_M value (Raw Samples) across {num_chains} unthinned chains: {np.round(mean_mean_value_Omega_
print(f'Mean w value (Raw Samples) across {num_chains} unthinned chains: {np.round(mean_mean_value_w, 3)}')
print(f'Mean M_n value (Raw Samples) across {num_chains} unthinned chains: {np.round(mean_mean_value_M_n, 3)}')
```

```
Mean Omega_M value (Raw Samples) across 7 unthinned chains: 0.274
Mean w value (Raw Samples) across 7 unthinned chains: -1.028
Mean M_n value (Raw Samples) across 7 unthinned chains: 15.774
```

In [129…
```
# All Chains Thinned Results, Raw Samples

print(f'Mean Omega_M (Raw Samples) value across {num_chains} thinned chains: {np.round(thinned_mean_mean_value_
print(f'Mean w value (Raw Samples) across {num_chains} thinned chains: {np.round(thinned_mean_mean_value_w, 3)}
print(f'Mean M_n value (Raw Samples) across {num_chains} thinned chains: {np.round(thinned_mean_mean_value_M_n,
```

```
Mean Omega_M (Raw Samples) value across 7 thinned chains: 0.274 ± 0.077
Mean w value (Raw Samples) across 7 thinned chains: -1.028 ± 0.201
Mean M_n value (Raw Samples) across 7 thinned chains: 15.774 ± 0.015
```

In [130…
```
# All Chains Thinned/Unthinned Percent Errors, Raw Samples

likely_raw_sample_error_thinned_unthinned_diff_all_chains_Omega_M = ( np.abs(mean_mean_value_Omega_M - thinned_
likely_raw_sample_error_thinned_unthinned_diff_all_chains_w = ( np.abs(mean_mean_value_w - thinned_mean_mean_va
likely_raw_sample_error_thinned_unthinned_diff_all_chains_M_n = ( np.abs(mean_mean_value_M_n - thinned_mean_mea

print(f'Percent error (Raw Samples) between thinned and unthinned estimates for Omega_M: {np.round(likely_raw_s
print(f'Percent error (Raw Samples) between thinned and unthinned estimates for w: {np.round(likely_raw_sample_
print(f'Percent error (Raw Samples) between thinned and unthinned estimates for M_n: {np.round(likely_raw_sampl
```

```
Percent error (Raw Samples) between thinned and unthinned estimates for Omega_M: 0.057 %
Percent error (Raw Samples) between thinned and unthinned estimates for w: -0.008 %
Percent error (Raw Samples) between thinned and unthinned estimates for M_n: 0.001 %
```

In [131…
```
# All Chains Unthinned Results, Polynomial Averaging

for i in range(len(conv_vals)):
    print(f"Estimated {params[i]} (Polynomial Averaging): {np.round(conv_vals[i], 3)} ± {np.round(conv_vals_err
```

```
Estimated Omega_M (Polynomial Averaging): 0.297 ± 0.013
Estimated w (Polynomial Averaging): -1.0 ± 0.027
Estimated M_n (Polynomial Averaging): 15.775 ± 0.003
Estimated Likelihood (Polynomial Averaging): 118.315 ± 0.258
```

In [132…
```
# All Chains Thinned Results, Polynomial Averaging

for i in range(len(thinned_conv_vals)):
    print(f"Estimated {params[i]} (Polynomial Averaging): {np.round(thinned_conv_vals[i], 3)} ± {np.round(thinn
```

```
Estimated Omega_M (Polynomial Averaging): 0.299 ± 0.013
Estimated w (Polynomial Averaging): -0.985 ± 0.029
Estimated M_n (Polynomial Averaging): 15.774 ± 0.001
Estimated Likelihood (Polynomial Averaging): 118.286 ± 0.237
```

In [133…
```
# All Chains Thinned/Unthinned Percent Errors, Polynomial Averaging

likely_error_thinned_unthinned_diff_all_chains = np.zeros(len(params))

for i in range(len(params)):
    likely_error_thinned_unthinned_diff_all_chains[i] = ( np.abs(conv_vals[i] - thinned_conv_vals[i]) / thinned_

    print(f'Percent error (Polynomial Averaging) between thinned and unthinned estimates for {params[i]}: {np.r
```

```
Percent error (Polynomial Averaging) between thinned and unthinned estimates for Omega_M: 0.884 %
Percent error (Polynomial Averaging) between thinned and unthinned estimates for w: -1.539 %
Percent error (Polynomial Averaging) between thinned and unthinned estimates for M_n: 0.003 %
Percent error (Polynomial Averaging) between thinned and unthinned estimates for Likelihood: 0.024 %
```

In [134…
```
# All Chains Unthinned Results, Likelihood Indexing

log_likelihood_flat = all_chains_log_likelihoods.flatten()
log_likelihood_all_chains_index = np.argmax(log_likelihood_flat)
max_log_likelihood_value = log_likelihood_flat[log_likelihood_all_chains_index]

tolerance_thinned = 0.1

for i in range(all_chains_log_likelihoods.shape[0]):
    for j in range(all_chains_log_likelihoods.shape[1]):
        if abs(all_chains_log_likelihoods[i, j] - max_log_likelihood_value) < tolerance_thinned:
            likely_omega_m_all_chains = all_chains_Omega_M[i, j]
            likely_w_all_chains = all_chains_w[i, j]
            likely_M_n_all_chains = all_chains_M_n[i, j]
            likely_log_likelihood_all_chains = all_chains_log_likelihoods[i, j]

print(f'The most likely unthinned Omega_M value (Likelihood Indexing) for MCMC Chain 1 is: {np.round(likely_ome
print(f'The most likely unthinned w value (Likelihood Indexing) for MCMC Chain 1 is: {np.round(likely_w_all_cha
print(f'The most likely unthinned M_n value (Likelihood Indexing) for MCMC Chain 1 is: {np.round(likely_M_n_all
print(f'The most likely unthinned log likelihood (Likelihood Indexing) for MCMC Chain 1 is: {np.round(likely_lo
```

```
The most likely unthinned Omega_M value (Likelihood Indexing) for MCMC Chain 1 is: 0.26
The most likely unthinned w value (Likelihood Indexing) for MCMC Chain 1 is: -0.968
The most likely unthinned M_n value (Likelihood Indexing) for MCMC Chain 1 is: 15.775
The most likely unthinned log likelihood (Likelihood Indexing) for MCMC Chain 1 is: 118.681
```

In [135…
```
# All Chains Thinned Results, Likelihood Indexing
```

```
thinned_log_likelihood_flat = thinned_all_chains_log_likelihood_values.flatten()
thinned_log_likelihood_all_chains_index = np.argmax(thinned_log_likelihood_flat)
max_log_likelihood_value = thinned_log_likelihood_flat[thinned_log_likelihood_all_chains_index]

tolerance_thinned = 0.1

for i in range(thinned_all_chains_log_likelihood_values.shape[0]):
    for j in range(thinned_all_chains_log_likelihood_values.shape[1]):
        if abs(thinned_all_chains_log_likelihood_values[i, j] - max_log_likelihood_value) < tolerance_thinned:
            likely_thinned_omega_m_all_chains = thinned_all_chains_Omega_M_values[i, j]
            likely_thinned_w_all_chains = thinned_all_chains_w_values[i, j]
            likely_thinned_M_n_all_chains = thinned_all_chains_M_n_values[i, j]
            likely_thinned_log_likelihood_all_chains = thinned_all_chains_log_likelihood_values[i, j]

print(f'The most likely thinned Omega_M value (Likelihood Indexing) for MCMC Chain 1 is: {np.round(likely_thinn
print(f'The most likely thinned w value (Likelihood Indexing) for MCMC Chain 1 is: {np.round(likely_thinned_w_a
print(f'The most likely thinned M_n value (Likelihood Indexing) for MCMC Chain 1 is: {np.round(likely_thinned_M_
print(f'The most likely thinned log likelihood value (Likelihood Indexing) for MCMC Chain 1 is: {np.round(likel
```

```
The most likely thinned Omega_M value (Likelihood Indexing) for MCMC Chain 1 is: 0.299
The most likely thinned w value (Likelihood Indexing) for MCMC Chain 1 is: -1.061
The most likely thinned M_n value (Likelihood Indexing) for MCMC Chain 1 is: 15.77
The most likely thinned log likelihood value (Likelihood Indexing) for MCMC Chain 1 is: 118.665
```

In [136…

```
# All Chains Thinned/Unthinned Percent Errors, Likelihood Indexing

likely_omega_m_error_thinned_unthinned_diff_all_chains = ( np.abs(likely_omega_m_all_chains - likely_thinned_om
likely_w_error_thinned_unthinned_diff_all_chains = ( np.abs(likely_w_chain_1 - likely_thinned_w_all_chains) / l
likely_M_n_error_thinned_unthinned_diff_all_chains = ( np.abs(likely_M_n_chain_1 - likely_thinned_M_n_all_chain
likely_log_likelihood_error_thinned_unthinned_diff_all_chains = ( np.abs(likely_log_likelihood_all_chains - lik

print(f'Percent error (Likelihood Indexing) between thinned and unthinned estimates for Omega_M: {np.round(like
print(f'Percent error (Likelihood Indexing) between thinned and unthinned estimates for w: {np.round(likely_w_e
print(f'Percent error (Likelihood Indexing) between thinned and unthinned estimates for M_n: {np.round(likely_M
print(f'Percent error (Likelihood Indexing) between thinned and unthinned estimates for log_likelihood: {np.rou
```

```
Percent error (Likelihood Indexing) between thinned and unthinned estimates for Omega_M: 13.024 %
Percent error (Likelihood Indexing) between thinned and unthinned estimates for w: -3.985 %
Percent error (Likelihood Indexing) between thinned and unthinned estimates for M_n: 0.024 %
Percent error (Likelihood Indexing) between thinned and unthinned estimates for log_likelihood: 0.014 %
```

In [137…

```
# All Chains Unthinned Results, Posteriors

# Unthinned Posteriors
posteriors_all_chains = np.exp(all_chains_log_likelihoods - np.max(all_chains_log_likelihoods))
posteriors_all_chains /= np.sum(posteriors_all_chains)

# Parameter Estimations
Omega_M_estimate_all_chains = np.sum(all_chains_Omega_M * posteriors_all_chains)
w_estimate_all_chains = np.sum(all_chains_w * posteriors_all_chains)
M_n_estimate_all_chains = np.sum(all_chains_M_n * posteriors_all_chains)

# Parameter Uncertainties
Omega_M_std_all_chains = np.sqrt(np.sum(posteriors_all_chains * (all_chains_Omega_M - Omega_M_estimate_all_chai
w_std_all_chains = np.sqrt(np.sum(posteriors_all_chains * (all_chains_w - w_estimate_all_chains) ** 2))
M_n_std_all_chains = np.sqrt(np.sum(posteriors_all_chains * (all_chains_M_n - M_n_estimate_all_chains) ** 2))

print(f"Estimated Omega_M (Posterior): {np.round(Omega_M_estimate_all_chains, 3)} ± {np.round(Omega_M_std_all_c
print(f"Estimated w (Posterior): {np.round(w_estimate_all_chains, 3)} ± {np.round(w_std_all_chains, 3)}")
print(f"Estimated M_n (Posterior): {np.round(M_n_estimate_all_chains, 3)} ± {np.round(M_n_std_all_chains, 3)}")
```

```
Estimated Omega_M (Posterior): 0.277 ± 0.055
Estimated w (Posterior): -1.018 ± 0.144
Estimated M_n (Posterior): 15.774 ± 0.01
```

In [214…

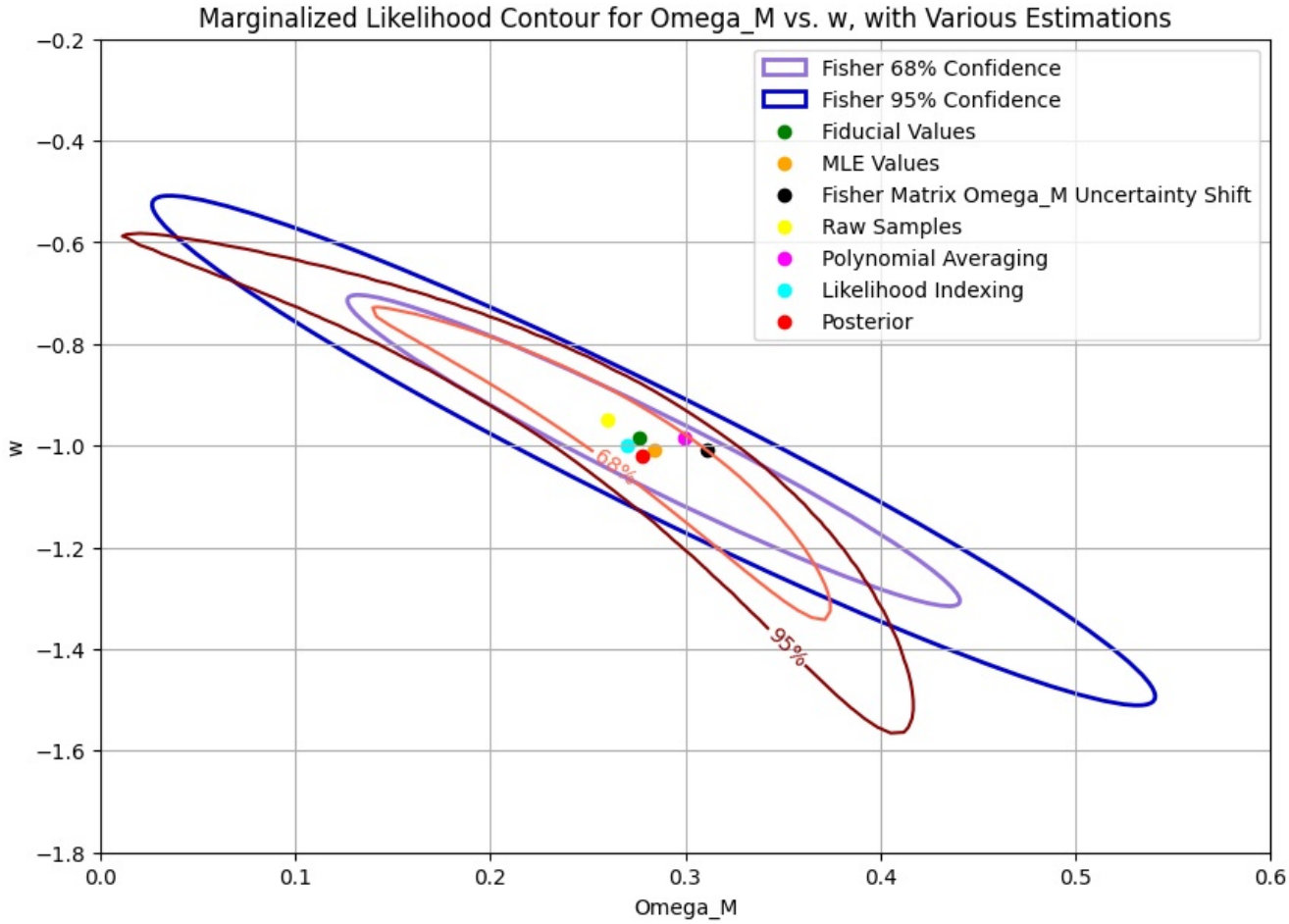
```
# All Chains Thinned Results, Posteriors

# Unthinned Posteriors
thinned_posteriors_all_chains = np.exp(thinned_all_chains_log_likelihood_values - np.max(thinned_all_chains_log
thinned_posteriors_all_chains /= np.sum(thinned_posteriors_all_chains)

# Parameter Estimations
thinned_Omega_M_estimate_all_chains = np.sum(thinned_all_chains_Omega_M_values * thinned_posteriors_all_chains)
thinned_w_estimate_all_chains = np.sum(thinned_all_chains_w_values * thinned_posteriors_all_chains)
thinned_M_n_estimate_all_chains = np.sum(thinned_all_chains_M_n_values * thinned_posteriors_all_chains)

# Parameter Uncertainties
thinned_Omega_M_std_all_chains = np.sqrt(np.sum(thinned_posteriors_all_chains * (thinned_all_chains_Omega_M_valu
thinned_w_std_all_chains = np.sqrt(np.sum(thinned_posteriors_all_chains * (thinned_all_chains_w_values - thinne
thinned_M_n_std_all_chains = np.sqrt(np.sum(thinned_posteriors_all_chains * (thinned_all_chains_M_n_values - th

print(f"Estimated Omega_M (Posterior): {np.round(thinned_Omega_M_estimate_all_chains, 3)} ± {np.round(thinned_O
print(f"Estimated w (Posterior): {np.round(thinned_w_estimate_all_chains, 3)} ± {np.round(thinned_w_std_all_cha
print(f"Estimated M_n (Posterior): {np.round(thinned_M_n_estimate_all_chains, 3)} ± {np.round(thinned_M_n_std_a
```

```
Estimated Omega_M (Posterior): 0.278 ± 0.055
Estimated w (Posterior): -1.019 ± 0.144
Estimated M_n (Posterior): 15.774 ± 0.01
```

In [215…
```python
# All Chains Thinned/Unthinned Percent Errors, Using Posteriors

posterior_likely_omega_m_error_thinned_unthinned_diff_all_chains = ( np.abs(Omega_M_estimate_all_chains - thinn
posterior_likely_w_error_thinned_unthinned_diff_all_chains = ( np.abs(w_estimate_all_chains - thinned_w_estimat
posterior_likely_M_n_error_thinned_unthinned_diff_all_chains = ( np.abs(M_n_estimate_all_chains - thinned_M_n_e

print(f'Percent error (with Posteriors) between thinned and unthinned estimates for Omega_M: {np.round(posterio
print(f'Percent error (with Posteriors) between thinned and unthinned estimates for w: {np.round(posterior_like
print(f'Percent error (with Posteriors) between thinned and unthinned estimates for M_n: {np.round(posterior_li

# Would an increase/decrease in these percent errors mean anything?
```

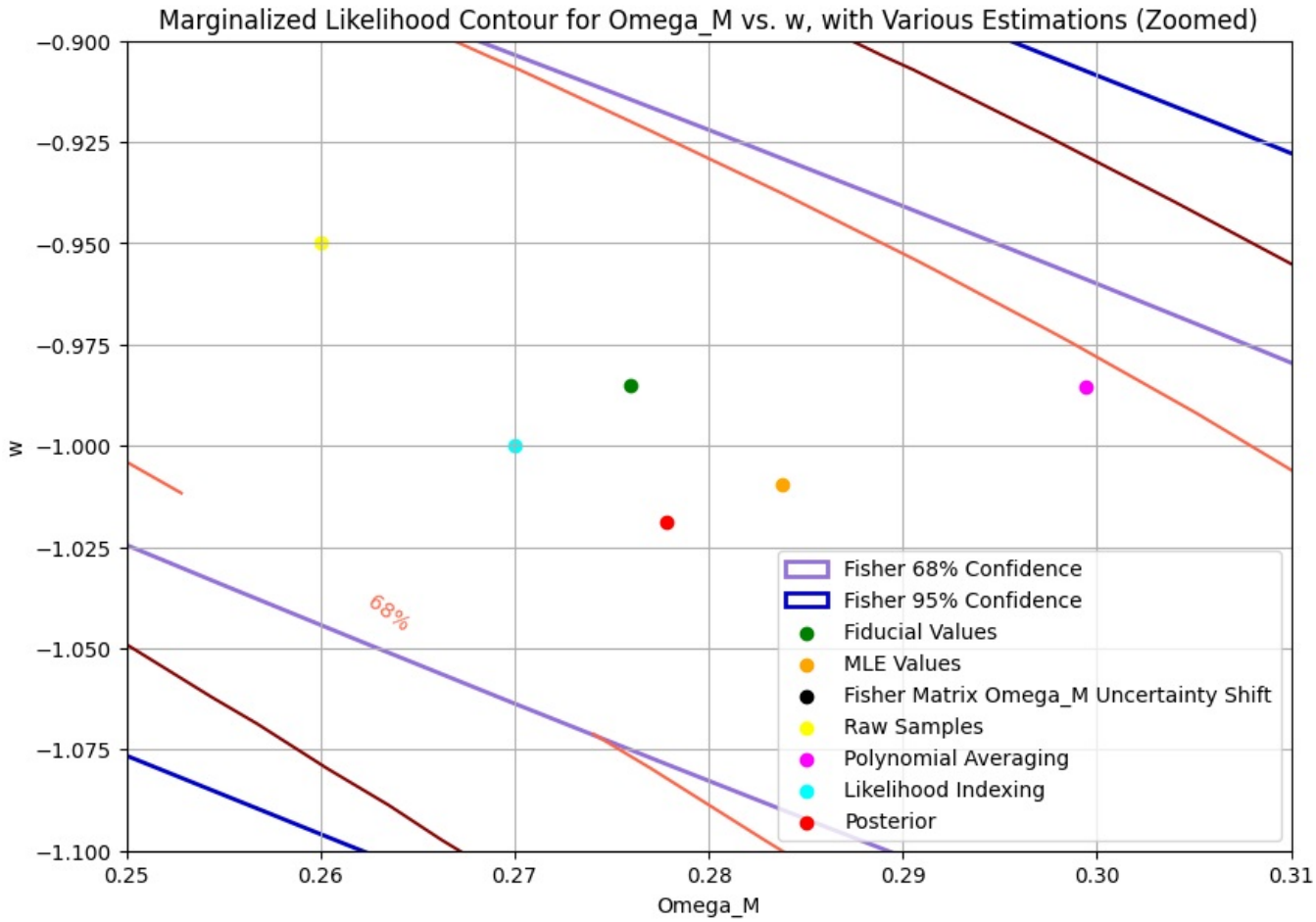
```
Percent error (with Posteriors) between thinned and unthinned estimates for Omega_M: 0.213 %
Percent error (with Posteriors) between thinned and unthinned estimates for w: -0.083 %
Percent error (with Posteriors) between thinned and unthinned estimates for M_n: 0.0 %
```

In [216…
```python
# Plot with all analyses and estimations

# Contour plot
plt.figure(figsize=(10, 7))
ax = plt.gca()
cp = plt.contour(omega_m_grid1_slice, w_grid1_slice, marg_likelihoods_w_m_mn, levels=[level_95_w_m_mn, level_68
plt.clabel(cp, inline=True, fontsize=10, fmt={level_68_w_m_mn: '68%', level_95_w_m_mn: '95%'})

# Plot Fisher confidence ellipses
boundary_right_sig1, boundary_right_sig2 = plot_fisher_confidence_ellipse(ax, fisher_at_most_likely, (most_like
plot_fisher_confidence_ellipse(ax, fisher_at_most_likely, (most_likely_omega_m1, most_likely_w1), 6.17)

# Plot Estimations
plt.scatter(fixed_omega_m, fixed_w, color='green', label='Fiducial Values')
plt.scatter(most_likely_omega_m1, most_likely_w1, color='orange', label='MLE Values')
plt.scatter(boundary_right_sig1, most_likely_w1, color='black', label='Fisher Matrix Omega_M Uncertainty Shift'
plt.scatter(thinned_mean_mean_value_Omega_M, thinned_mean_mean_value_w, color='yellow', label='Raw Samples')
plt.scatter(thinned_conv_vals[0], thinned_conv_vals[1], color='magenta', label='Polynomial Averaging')
plt.scatter(likely_thinned_omega_m_all_chains, likely_thinned_w_all_chains, color='cyan', label='Likelihood Ind
plt.scatter(thinned_Omega_M_estimate_all_chains, thinned_w_estimate_all_chains, color='red', label='Posterior')

# Plot
plt.title('Marginalized Likelihood Contour for Omega_M vs. w, with Various Estimations')
plt.xlabel('Omega_M')
plt.ylabel('w')
plt.xlim(0, 0.6)
plt.ylim(-1.8, -0.2)
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```



In [217…

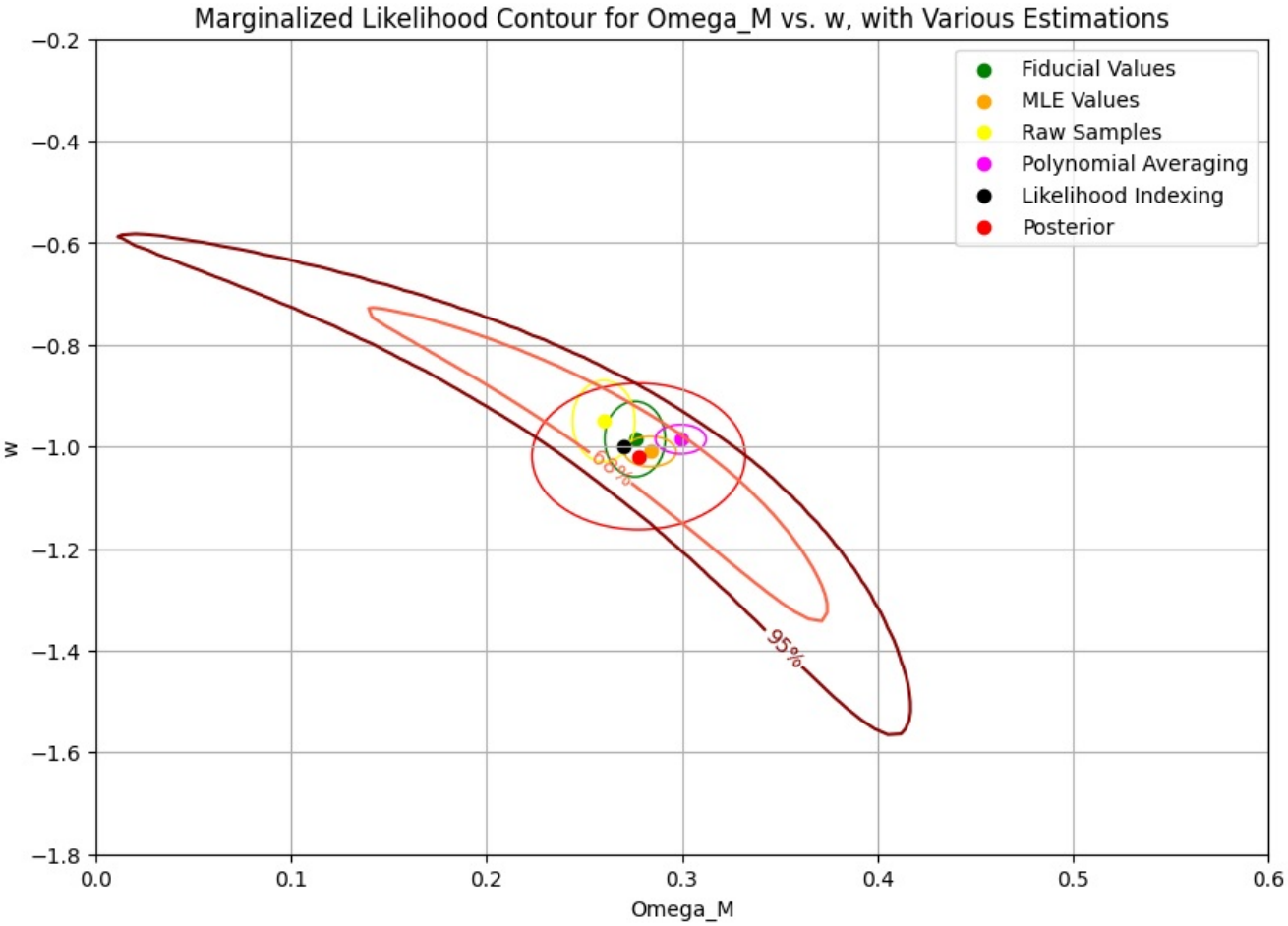
```python
# Plot with all analyses and estimations (Zoomed in)

# Contour plot
plt.figure(figsize=(10, 7))
ax = plt.gca()
cp = plt.contour(omega_m_grid1_slice, w_grid1_slice, marg_likelihoods_w_m_mn, levels=[level_95_w_m_mn, level_68_
plt.clabel(cp, inline=True, fontsize=10, fmt={level_68_w_m_mn: '68%', level_95_w_m_mn: '95%'})

# Plot Fisher confidence ellipses
boundary_right_sig1, boundary_right_sig2 = plot_fisher_confidence_ellipse(ax, fisher_at_most_likely, (most_like
plot_fisher_confidence_ellipse(ax, fisher_at_most_likely, (most_likely_omega_m1, most_likely_w1), 6.17)

# Plot Estimations
plt.scatter(fixed_omega_m, fixed_w, color='green', label='Fiducial Values')
plt.scatter(most_likely_omega_m1, most_likely_w1, color='orange', label='MLE Values')
plt.scatter(boundary_right_sig1, most_likely_w1, color='black', label='Fisher Matrix Omega_M Uncertainty Shift'
plt.scatter(thinned_mean_mean_value_Omega_M, thinned_mean_mean_value_w, color='yellow', label='Raw Samples')
plt.scatter(thinned_conv_vals[0], thinned_conv_vals[1], color='magenta', label='Polynomial Averaging')
plt.scatter(likely_thinned_omega_m_all_chains, likely_thinned_w_all_chains, color='cyan', label='Likelihood Ind
plt.scatter(thinned_Omega_M_estimate_all_chains, thinned_w_estimate_all_chains, color='red', label='Posterior')

# Plot
plt.title('Marginalized Likelihood Contour for Omega_M vs. w, with Various Estimations (Zoomed)')
plt.xlabel('Omega_M')
plt.ylabel('w')
plt.xlim(0.25, 0.31)
plt.ylim(-1.1, -0.9)
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



In [ ]:
```
# Std. Estimation Methods Arrays

thinned_Omega_M_estimation_stds = np.array([ \
    0.0155,                          # Fiducial Value
    0.013232117398141879,            # MLE Value
    thinned_overall_std_Omega_M,     # Raw Samples
    thinned_conv_vals_errs[0],       # Polynomial Averaging
    0,                               # Likelihood Indexing
    thinned_Omega_M_std_all_chains   # Posteriors
])

thinned_w_estimation_stds = np.array([
    0.074,                       # Fiducial Value
    0.02973710624537613,         # MLE Value
    thinned_overall_std_w,       # Raw Samples
    thinned_conv_vals_errs[1],   # Polynomial Averaging
    0,                           # Likelihood Indexing
    thinned_w_std_all_chains     # Posteriors
])

thinned_M_n_estimation_stds = np.array([
    0,                               # Fiducial Value
    0.00714790672990374,             # MLE Value
    thinned_overall_std_M_n,         # Raw Samples
    thinned_conv_vals_errs[2],       # Polynomial Averaging
    0,                               # Likelihood Indexing
    thinned_M_n_std_all_chains       # Posteriors
])
```

In [247…
```
# Plot with all analyses, estimations, and uncertainties

# Contour plot
plt.figure(figsize=(10, 7))
ax = plt.gca()
cp = plt.contour(omega_m_grid1_slice, w_grid1_slice, marg_likelihoods_w_m_mn, levels=[level_95_w_m_mn, level_68_
plt.clabel(cp, inline=True, fontsize=10, fmt={level_68_w_m_mn: '68%', level_95_w_m_mn: '95%'})

estimation_points = [
    (fixed_omega_m, fixed_w),
    (most_likely_omega_m1, most_likely_w1),
    (thinned_mean_mean_value_Omega_M, thinned_mean_mean_value_w),
    (thinned_conv_vals[0], thinned_conv_vals[1]),
    (likely_thinned_omega_m_all_chains, likely_thinned_w_all_chains),
    (thinned_Omega_M_estimate_all_chains, thinned_w_estimate_all_chains)
]
```

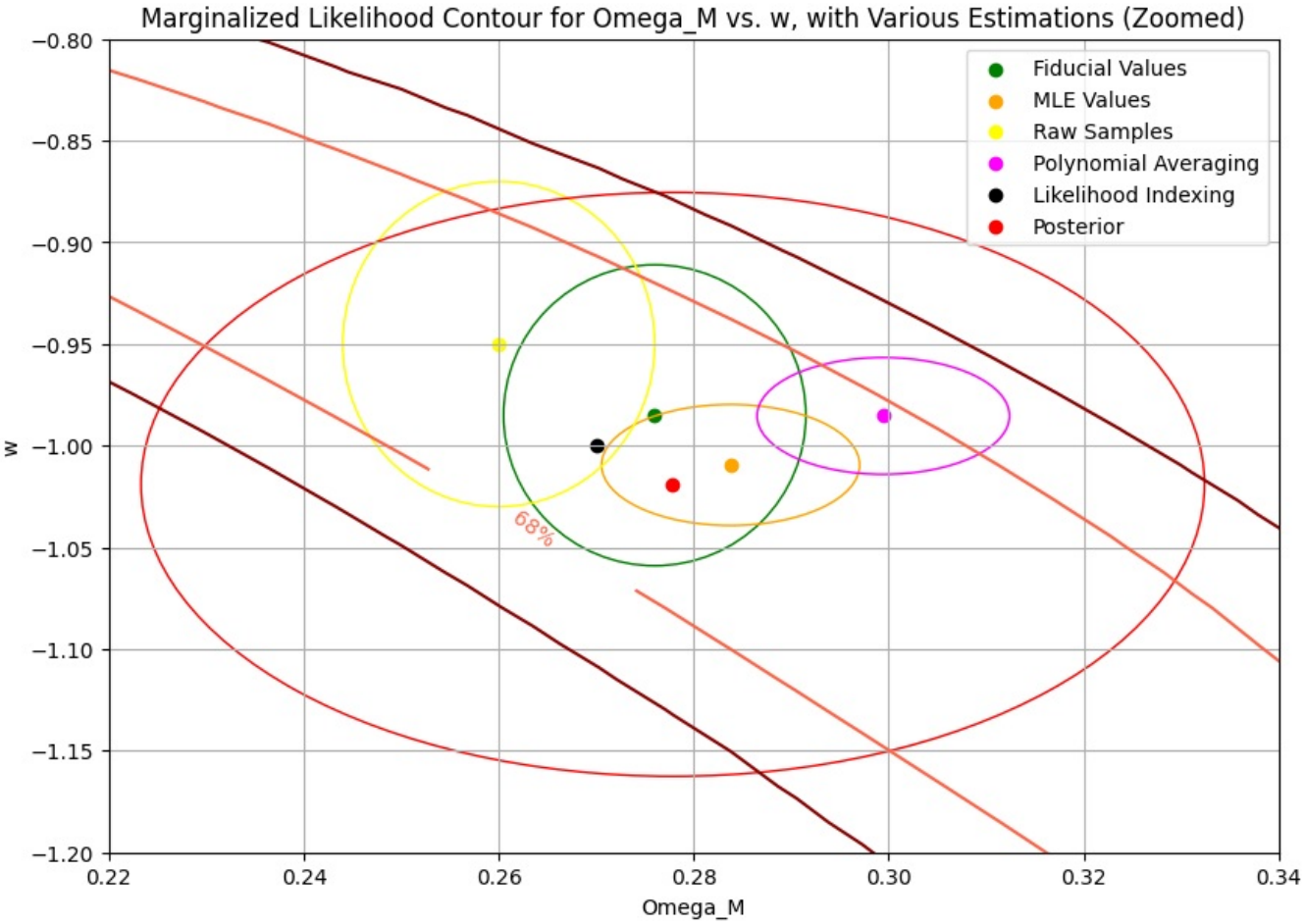
```
colors = ['green', 'orange', 'yellow', 'magenta', 'black', 'red']
labels = ['Fiducial Values', 'MLE Values', 'Raw Samples', 'Polynomial Averaging', 'Likelihood Indexing', 'Poste

for i, ((x, y), color, label) in enumerate(zip(estimation_points, colors, labels)):
    std_x = thinned_Omega_M_estimation_stds[i]
    std_y = thinned_w_estimation_stds[i]
    ellipse = Ellipse((x, y), 2*std_x, 2*std_y, edgecolor=color, facecolor='none')
    ax.add_patch(ellipse)
    plt.scatter(x, y, color=color, label=label)

# Plot
plt.title('Marginalized Likelihood Contour for Omega_M vs. w, with Various Estimations')
plt.xlabel('Omega_M')
plt.ylabel('w')
plt.xlim(0, 0.6)
plt.ylim(-1.8, -0.2)
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```



In [248…

```
# Plot with all analyses, estimations, and uncertainties (Zoomed in)

# Contour plot
plt.figure(figsize=(10, 7))
ax = plt.gca()
cp = plt.contour(omega_m_grid1_slice, w_grid1_slice, marg_likelihoods_w_m_mn, levels=[level_95_w_m_mn, level_68_
plt.clabel(cp, inline=True, fontsize=10, fmt={level_68_w_m_mn: '68%', level_95_w_m_mn: '95%'})

estimation_points = [
    (fixed_omega_m, fixed_w),
    (most_likely_omega_m1, most_likely_w1),
    (thinned_mean_mean_value_Omega_M, thinned_mean_mean_value_w),
    (thinned_conv_vals[0], thinned_conv_vals[1]),
    (likely_thinned_omega_m_all_chains, likely_thinned_w_all_chains),
    (thinned_Omega_M_estimate_all_chains, thinned_w_estimate_all_chains)
]

colors = ['green', 'orange', 'yellow', 'magenta', 'black', 'red']
labels = ['Fiducial Values', 'MLE Values', 'Raw Samples', 'Polynomial Averaging', 'Likelihood Indexing', 'Poste

for i, ((x, y), color, label) in enumerate(zip(estimation_points, colors, labels)):
    std_x = thinned_Omega_M_estimation_stds[i]
    std_y = thinned_w_estimation_stds[i]
    ellipse = Ellipse((x, y), 2*std_x, 2*std_y, edgecolor=color, facecolor='none')
    ax.add_patch(ellipse)
```

```
    plt.scatter(x, y, color=color, label=label)

# Plot
plt.title('Marginalized Likelihood Contour for Omega_M vs. w, with Various Estimations (Zoomed)')
plt.xlabel('Omega_M')
plt.ylabel('w')
plt.xlim(0.22, 0.34)
plt.ylim(-1.2, -0.8)
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```



In [ ]:
```
# fmin scipy optimize to get MLE (try different guesses)
# compare with MCMC
```

In [231…
```
# Dataframe of Unthinned Estimated Values and % Errors

# Values
unthinned_omega_m_estimates = np.array([
    fixed_omega_m,                # Fiducial Value
    most_likely_omega_m1,         # MLE Value
    mean_mean_value_Omega_M,      # Raw Samples
    conv_vals[0],                 # Polynomial Averaging
    likely_omega_m_all_chains,    # Likelihood Indexing
    Omega_M_estimate_all_chains   # Posteriors
])

unthinned_w_estimates = np.array([
    fixed_w,                # Fiducial Value
    most_likely_w1,         # MLE Value
    mean_mean_value_w,      # Raw Samples
    conv_vals[1],           # Polynomial Averaging
    likely_w_all_chains,    # Likelihood Indexing
    w_estimate_all_chains   # Posteriors
])

# DataFrame
df_unthinned_estimates = pd.DataFrame({
    "Method": ["Fiducial Value", "MLE Value", "Raw Samples", "Polynomial Averaging", "Likelihood Indexing", "Pos
    "Unthinned Omega_M Estimates": unthinned_omega_m_estimates,
    "Unthinned w Estimates": unthinned_w_estimates,
})

# Formatting
df_unthinned_estimates_sorted = df_unthinned_estimates.sort_values(by="Unthinned Omega_M Estimates", ascending=
```

```
for col in df_unthinned_estimates_sorted.columns:
    if col != "Method":
        df_unthinned_estimates_sorted[col] = df_unthinned_estimates_sorted[col].apply(lambda x: f'{x:.3f}')

html = df_unthinned_estimates_sorted.to_html(index=False)

display(HTML(html))
```

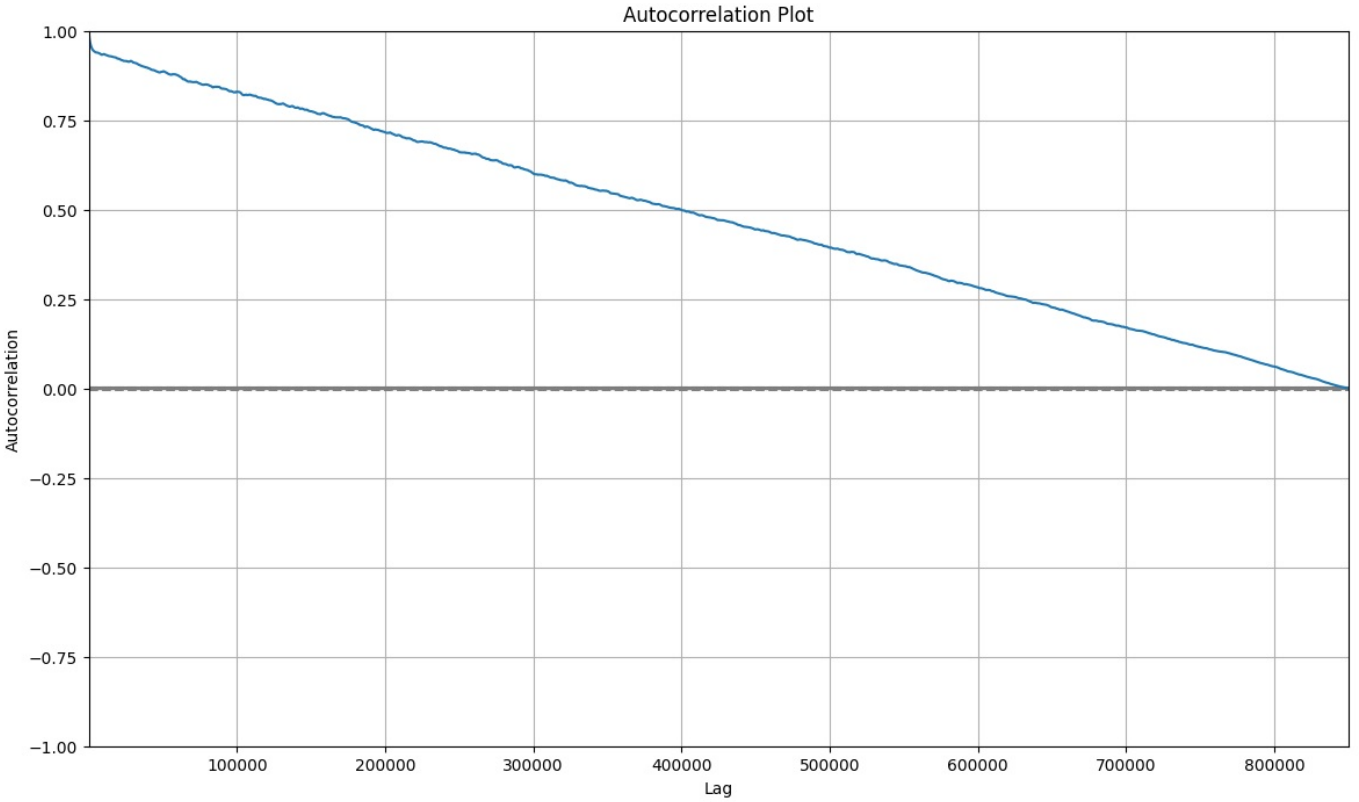
Method	Unthinned Omega_M Estimates	Unthinned w Estimates
Polynomial Averaging	0.297	-1.000
MLE Value	0.284	-1.009
Posteriors	0.277	-1.018
Fiducial Value	0.276	-0.985
Raw Samples	0.274	-1.028
Likelihood Indexing	0.260	-0.968

In [258…

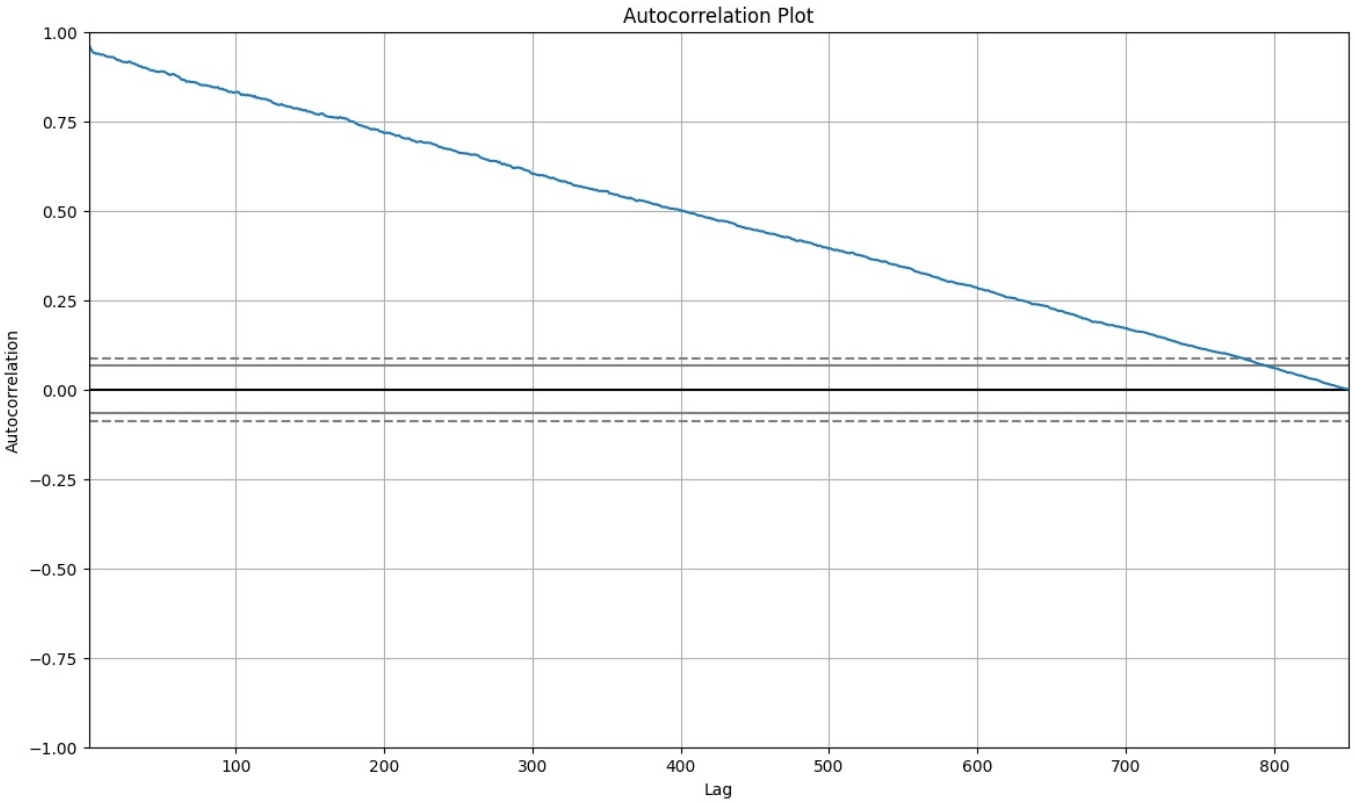
```
# Dataframe of Thinned Estimated Values and % Errors

# Values
thinned_omega_m_estimates = np.array([
    fixed_omega_m,                          # Fiducial Value
    most_likely_omega_m1,                   # MLE Value
    thinned_mean_mean_value_Omega_M,        # Raw Samples
    thinned_conv_vals[0],                   # Polynomial Averaging
    likely_thinned_omega_m_all_chains,      # Likelihood Indexing
    thinned_Omega_M_estimate_all_chains     # Posteriors
])

thinned_w_estimates = np.array([
    fixed_w,                          # Fiducial Value
    most_likely_w1,                   # MLE Value
    thinned_mean_mean_value_w,        # Raw Samples
    thinned_conv_vals[1],             # Polynomial Averaging
    likely_thinned_w_all_chains,      # Likelihood Indexing
    thinned_w_estimate_all_chains     # Posteriors
])

percent_error_omega_m = np.array([
    0,                                                                  # Fiducial Value
    0,                                                                  # MLE Value
    likely_raw_sample_error_thinned_unthinned_diff_all_chains_Omega_M,  # Raw Samples
    likely_error_thinned_unthinned_diff_all_chains[0],                  # Polynomial Averaging
    likely_omega_m_error_thinned_unthinned_diff_all_chains,             # Likelihood Indexing
    posterior_likely_omega_m_error_thinned_unthinned_diff_all_chains    # Posteriors
])

percent_error_w = np.array([
    0,                                                          # Fiducial Value
    0,                                                          # MLE Value
    likely_raw_sample_error_thinned_unthinned_diff_all_chains_w,    # Raw Samples
    likely_error_thinned_unthinned_diff_all_chains[1],          # Polynomial Averaging
    likely_w_error_thinned_unthinned_diff_all_chains,           # Likelihood Indexing
    posterior_likely_w_error_thinned_unthinned_diff_all_chains      # Posteriors
])

# Thinned
df_thinned_estimates = pd.DataFrame({
    "Method": ["Fiducial Value", "MLE Value", "Raw Samples", "Polynomial Averaging", "Likelihood Indexing", "Pos
    "Thinned Omega_M Est.": thinned_omega_m_estimates,
    "Thinned Omega_M Uncert.": thinned_Omega_M_estimation_stds,
    "Thinned w Est.": thinned_w_estimates,
    "Thinned w Uncert.": thinned_w_estimation_stds,
    "% " + "Error Thinned/Unthinned Omega_M": percent_error_omega_m,
    "% " + "Error Thinned/Unthinned w": percent_error_w
})

# Formatting
df_thinned_estimates_sorted = df_thinned_estimates.sort_values(by="Thinned Omega_M Est.", ascending=False).rese

for col in df_thinned_estimates_sorted.columns:
    if col != "Method":
        if "%" in col:
            df_thinned_estimates_sorted[col] = df_thinned_estimates_sorted[col].apply(lambda x: f'{x:.3f} %')
        else:
            df_thinned_estimates_sorted[col] = df_thinned_estimates_sorted[col].apply(lambda x: f'{x:.3f}')

html1 = df_thinned_estimates_sorted.to_html(index=False)

display(HTML(html1))
```

Method	Thinned Omega_M Est.	Thinned Omega_M Uncert.	Thinned w Est.	Thinned w Uncert.	% Error Thinned/Unthinned Omega_M	% Error Thinned/Unthinned w
Polynomial Averaging	0.299	0.013	-0.985	0.029	0.884 %	-1.539 %
MLE Value	0.284	0.013	-1.009	0.030	0.000 %	0.000 %
Posteriors	0.278	0.055	-1.019	0.144	0.213 %	-0.083 %
Fiducial Value	0.276	0.015	-0.985	0.074	0.000 %	0.000 %
Likelihood Indexing	0.270	0.000	-1.000	0.000	13.024 %	-3.985 %
Raw Samples	0.260	0.016	-0.950	0.080	0.057 %	-0.008 %

In [195…
```
# Unthinned Autocorrelation Plot

data_ac = np.vstack((chain_1_Omega_M_values, chain_1_w_values)).T
df_autocorr = pd.DataFrame(data=data_ac, columns=["$\\Omega_M$", 'w'])

fig = plt.figure(figsize = (14, 8))
plt.title("Autocorrelation Plot")
plt.xlabel("Autocorrelation")
plt.ylabel("Lag")
plt.ylim(-0.25, 1)
autocorrelation_plot(df_autocorr[:])
plt.show()
```

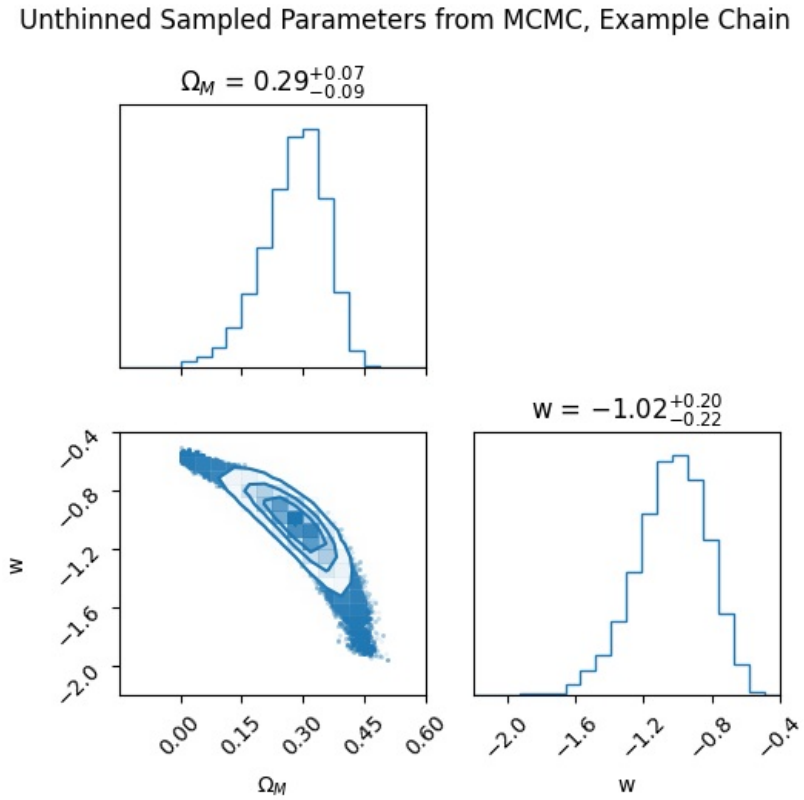


In [196…
```
# Thinned Autocorrelation Plot

thinned_data_ac = np.vstack((thinned_chain_1_Omega_M_values, thinned_chain_1_w_values)).T
df_autocorr = pd.DataFrame(data=thinned_data_ac, columns=["$\\Omega_M$", 'w'])

fig = plt.figure(figsize = (14, 8))
plt.title("Autocorrelation Plot")
plt.xlabel("Autocorrelation")
plt.ylabel("Lag")
plt.ylim(-0.25, 1)
autocorrelation_plot(df_autocorr[:])
plt.show()

# If the autocorrelation plot goes to 0 at max iteration, does that mean that the # of samples is too low?
```



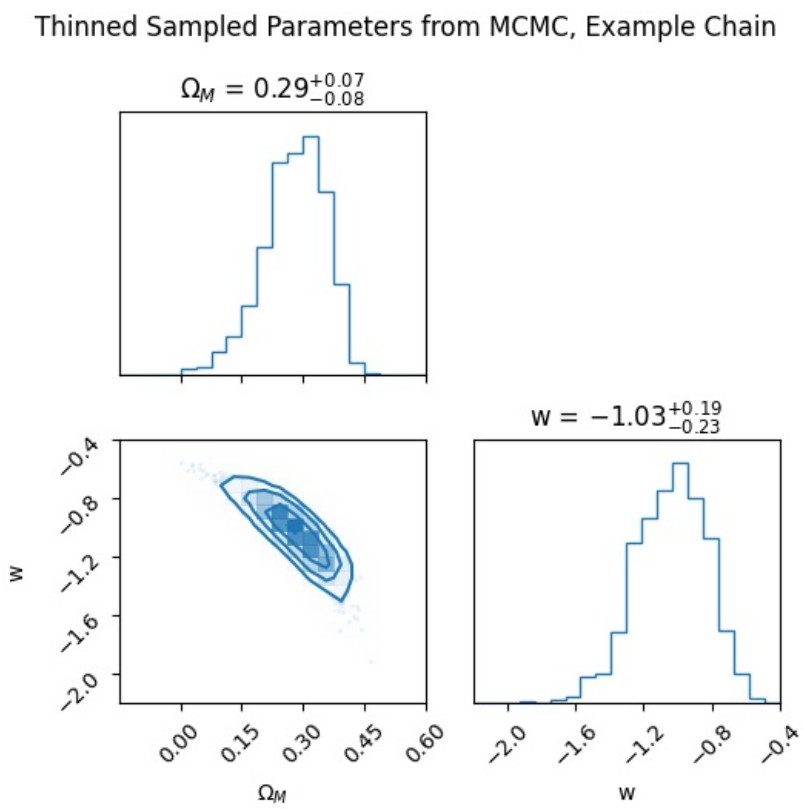
In [203…
```
# Unthinned Corner Plot Sample Chain

data_corner = np.vstack((chain_1_Omega_M_values, chain_1_w_values)).T
fig = corner.corner(data_corner, color='C0', smooth=0.7, labels=['$\\Omega_M$', 'w'], range=[[-0.15, 0.6], [-2.1
fig.suptitle("Unthinned Sampled Parameters from MCMC, Example Chain")
plt.tight_layout()
plt.show()
```

```
<Figure size 1400x800 with 0 Axes>
```



In [198…
```
# Thinned Corner Plot Sample Chain

thinned_data_corner = np.vstack((thinned_chain_1_Omega_M_values, thinned_chain_1_w_values)).T
fig = corner.corner(thinned_data_corner, color='C0', smooth=0.7, labels=['$\\Omega_M$', 'w'], range=[[-0.15, 0.6
fig.suptitle("Thinned Sampled Parameters from MCMC, Example Chain")
plt.tight_layout()
plt.show()
```

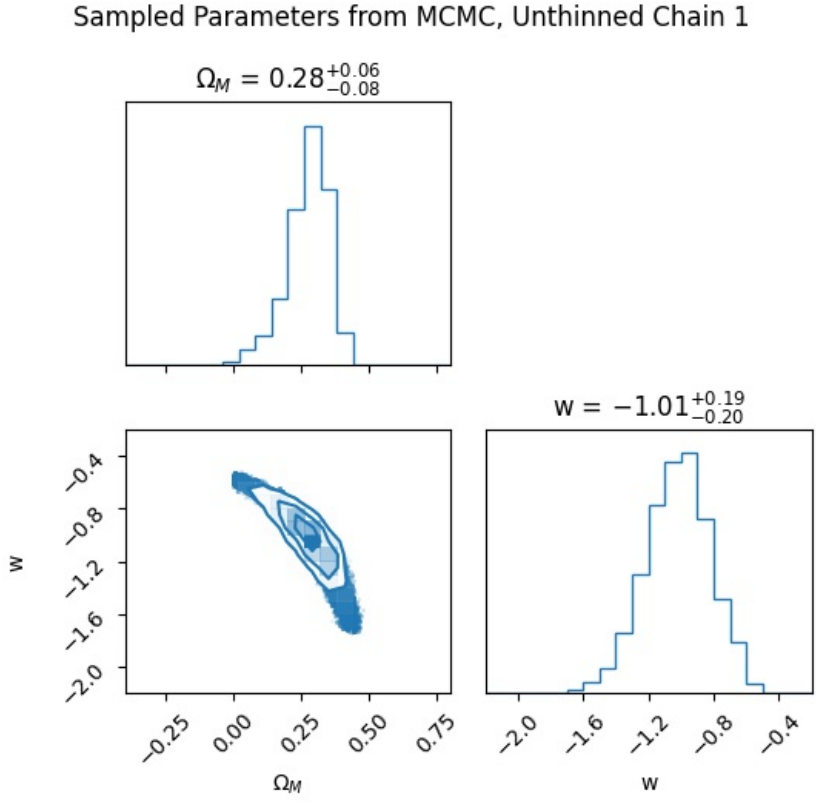
```
<Figure size 1400x800 with 0 Axes>
```



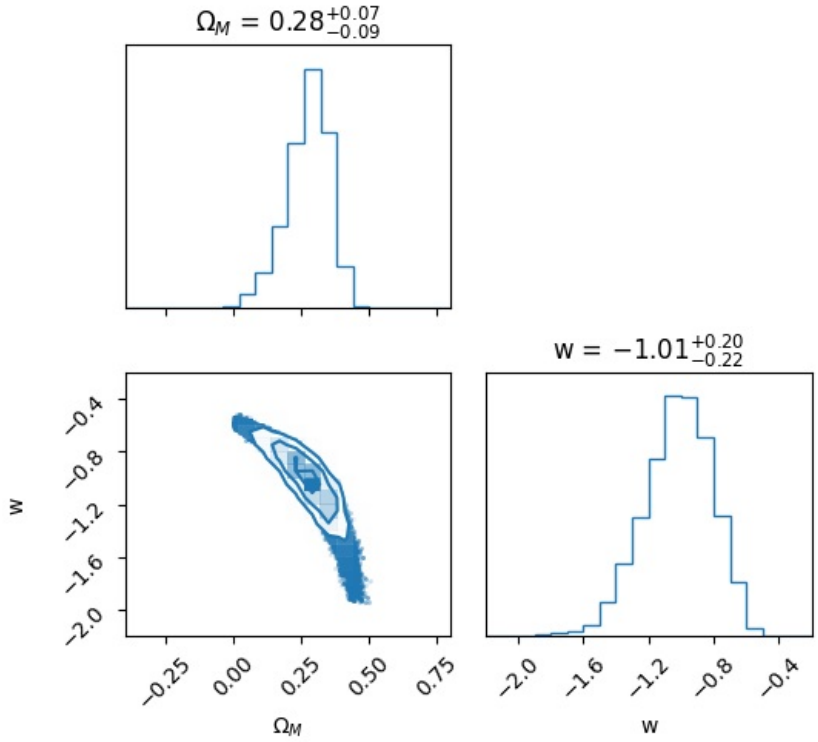
In [199…

```
# Unthinned Corner Plot All Chains

for i in range(num_chains):
    chain_Omega_M_values = all_chains_Omega_M[i, :]
    chain_w_values = all_chains_w[i, :]
    data_corner = np.vstack((chain_Omega_M_values, chain_w_values)).T

    figure = corner.corner(data_corner, color='C0', smooth=0.5, labels=['$\\Omega_M$', 'w'],
                           range=[[-0.4, 0.8], [-2.2, -0.2]], show_titles=True,
                           hist_kwargs={'density': True})

    figure.suptitle(f"Sampled Parameters from MCMC, Unthinned Chain {i + 1}")
    plt.tight_layout()
    plt.show()
```

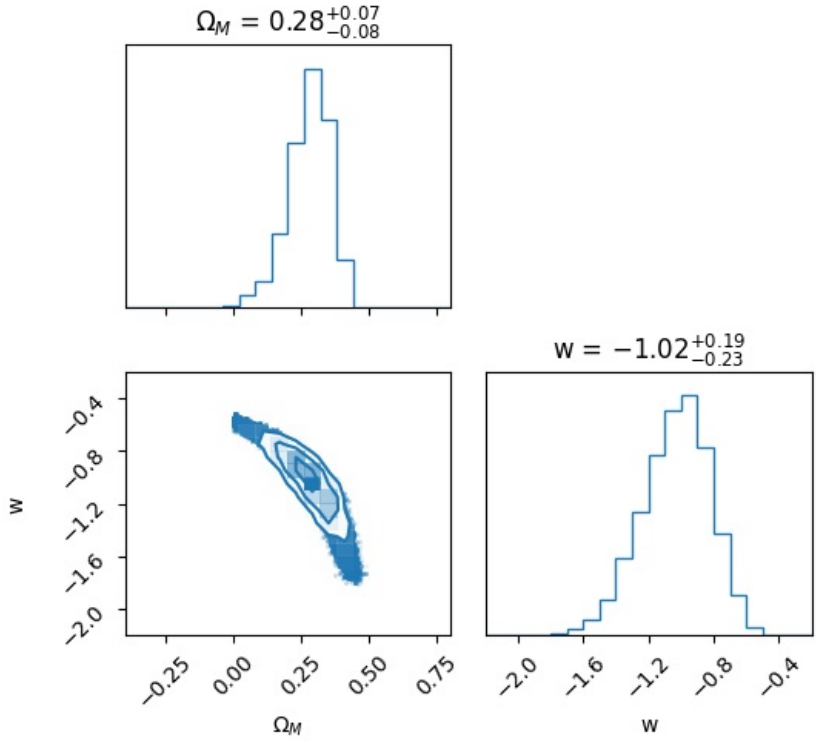
```
<Figure size 1400x800 with 0 Axes>
```



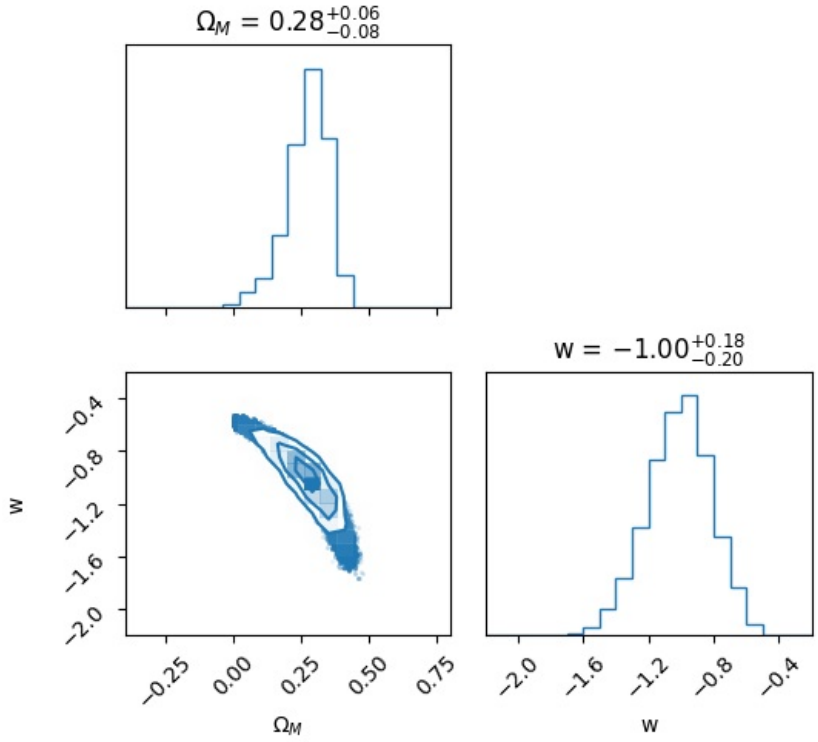
Sampled Parameters from MCMC, Unthinned Chain 2



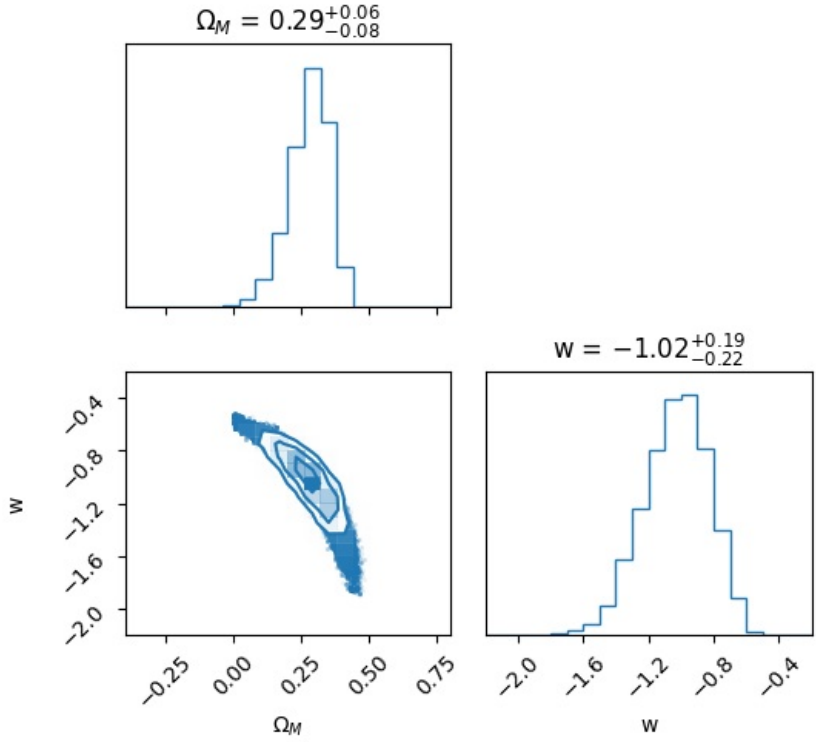
Sampled Parameters from MCMC, Unthinned Chain 3

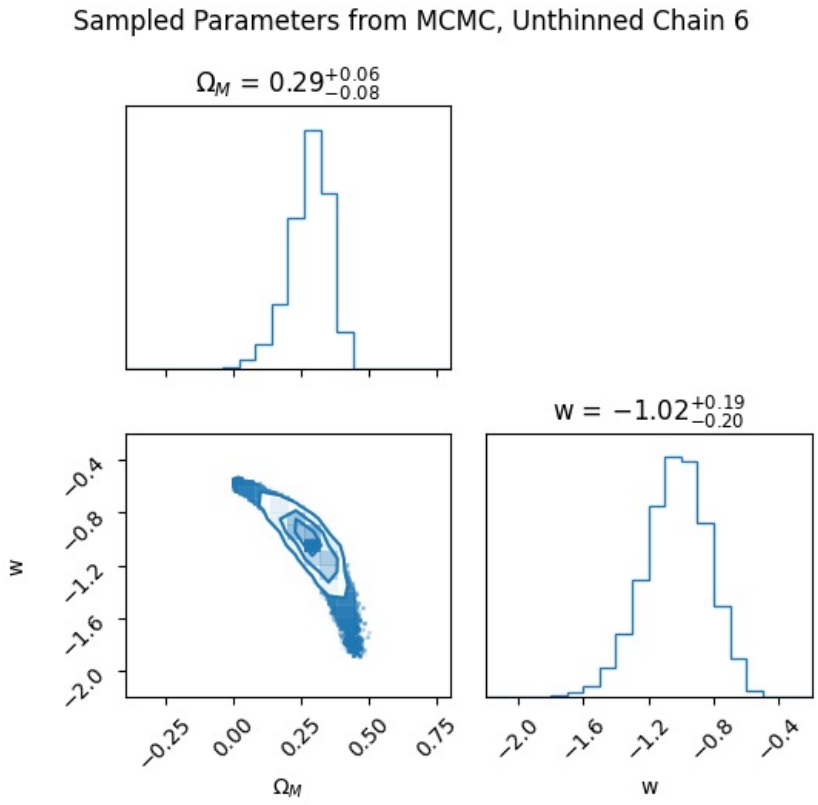


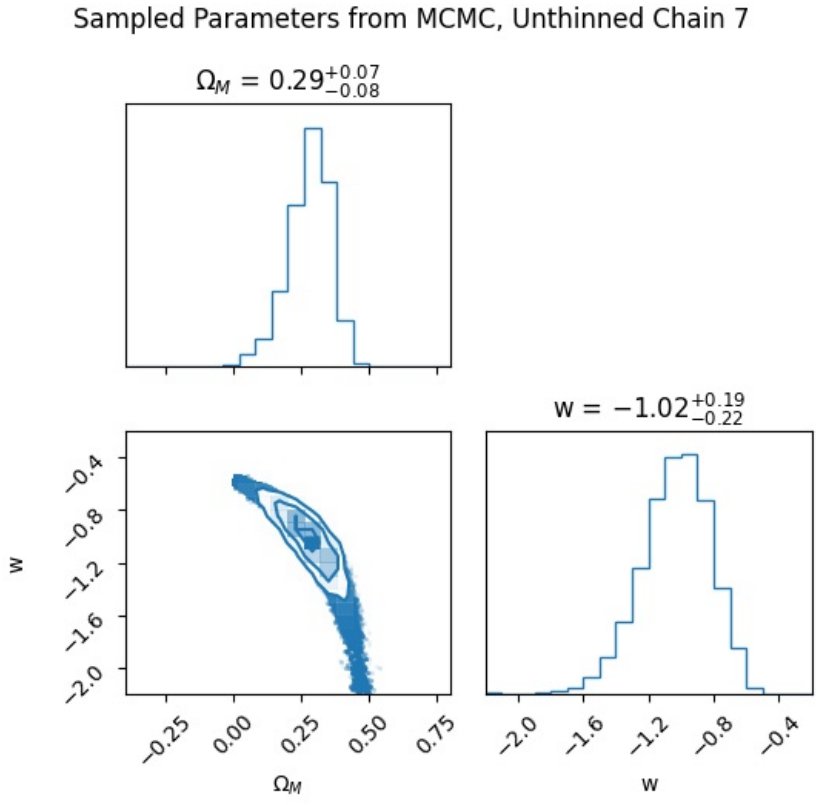
Sampled Parameters from MCMC, Unthinned Chain 4



Sampled Parameters from MCMC, Unthinned Chain 5





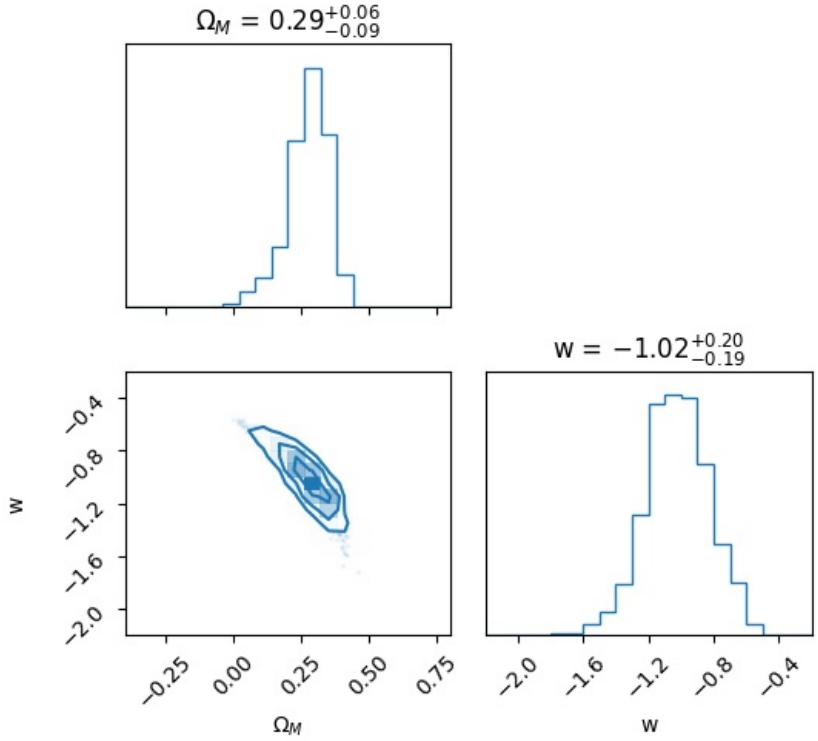


In [200…

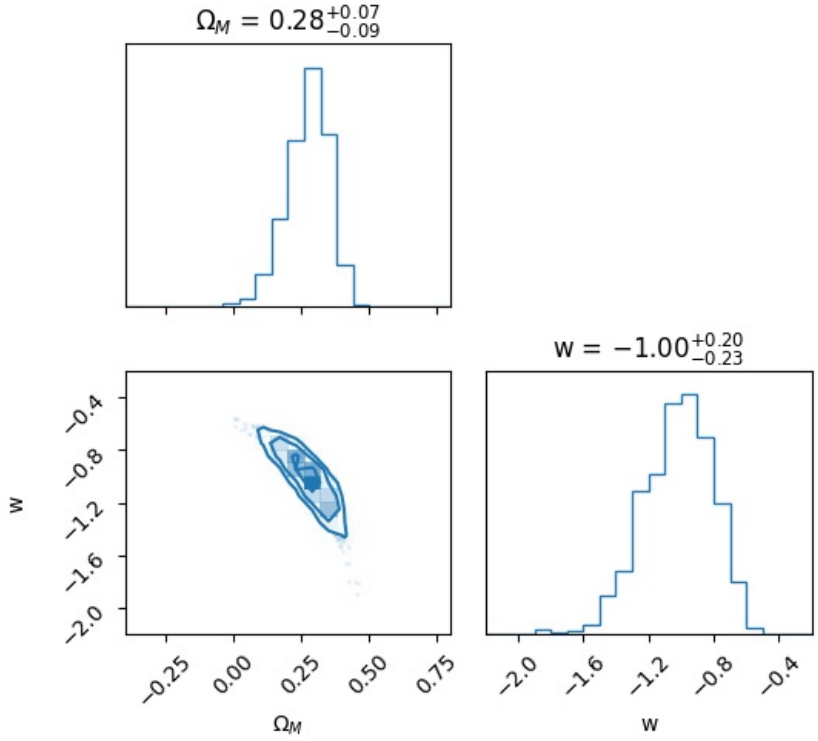
```python
# Thinned Corner Plot All Chains

for i in range(num_chains):
    thinned_chain_Omega_M_values = thinned_all_chains_Omega_M_values[i, :]
    thinned_chain_w_values = thinned_all_chains_w_values[i, :]
    thinned_data_corner = np.vstack((thinned_chain_Omega_M_values, thinned_chain_w_values)).T

    figure = corner.corner(thinned_data_corner, color='C0', smooth=0.5, labels=['$\\Omega_M$', 'w'],
                           range=[[-0.4, 0.8], [-2.2, -0.2]], show_titles=True,
                           hist_kwargs={'density': True})

    figure.suptitle(f"Sampled Parameters from MCMC, Thinned Chain {i + 1}")
    plt.tight_layout()
    plt.show()
```

```
<Figure size 1400x800 with 0 Axes>
```

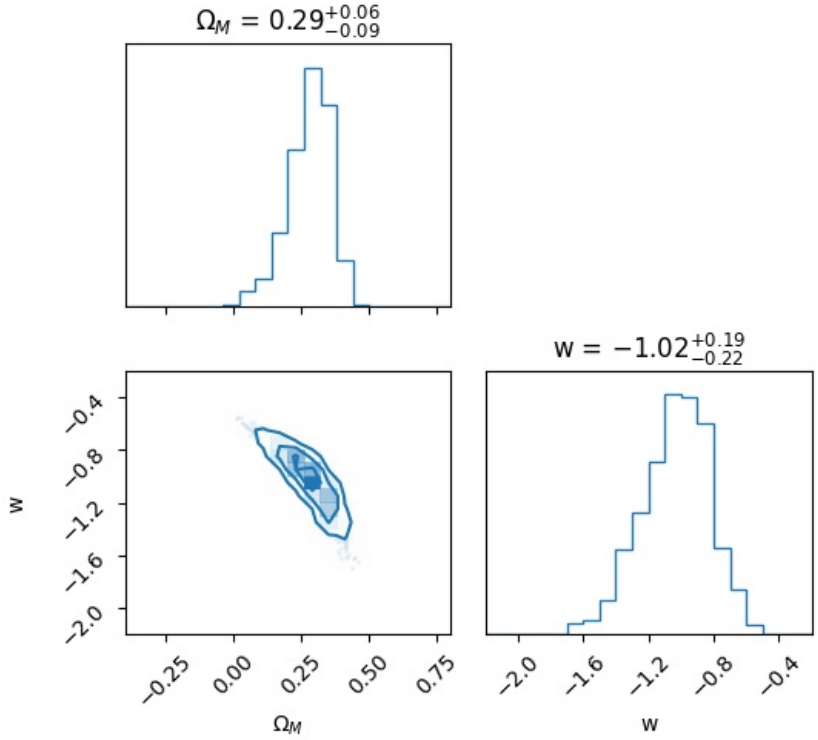
Sampled Parameters from MCMC, Thinned Chain 1



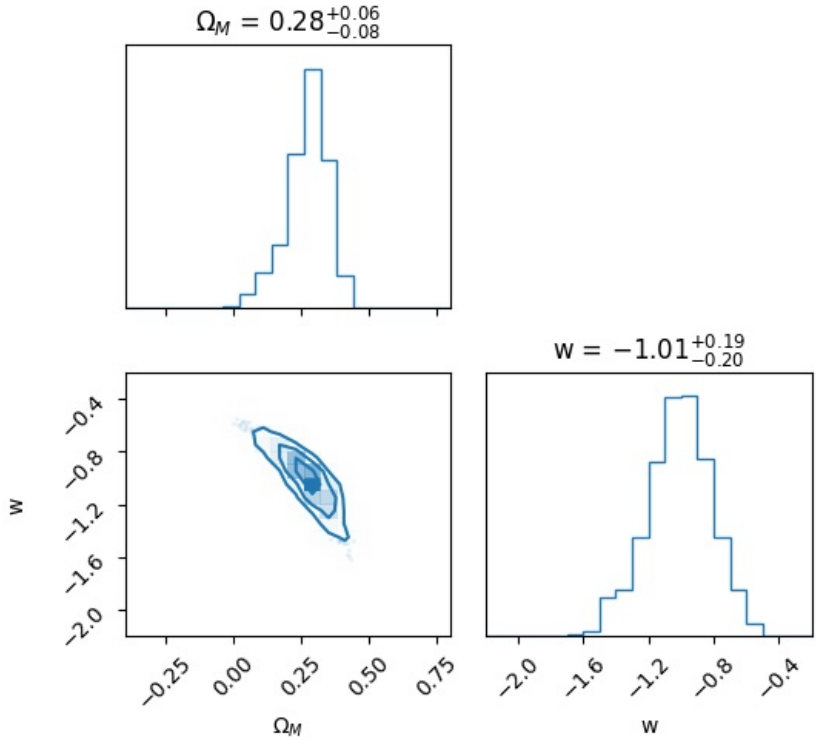
Sampled Parameters from MCMC, Thinned Chain 2



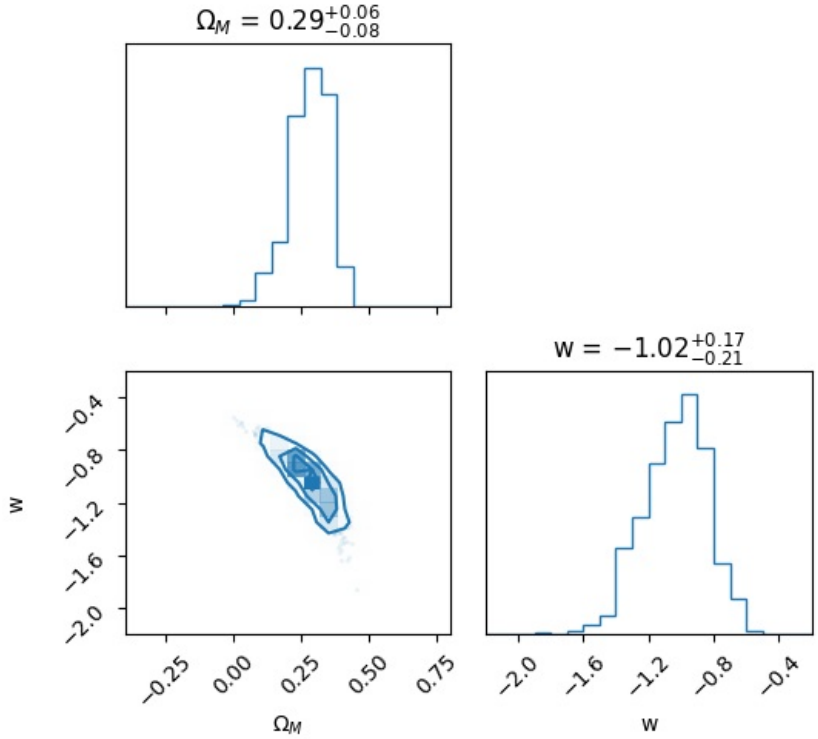
Sampled Parameters from MCMC, Thinned Chain 3



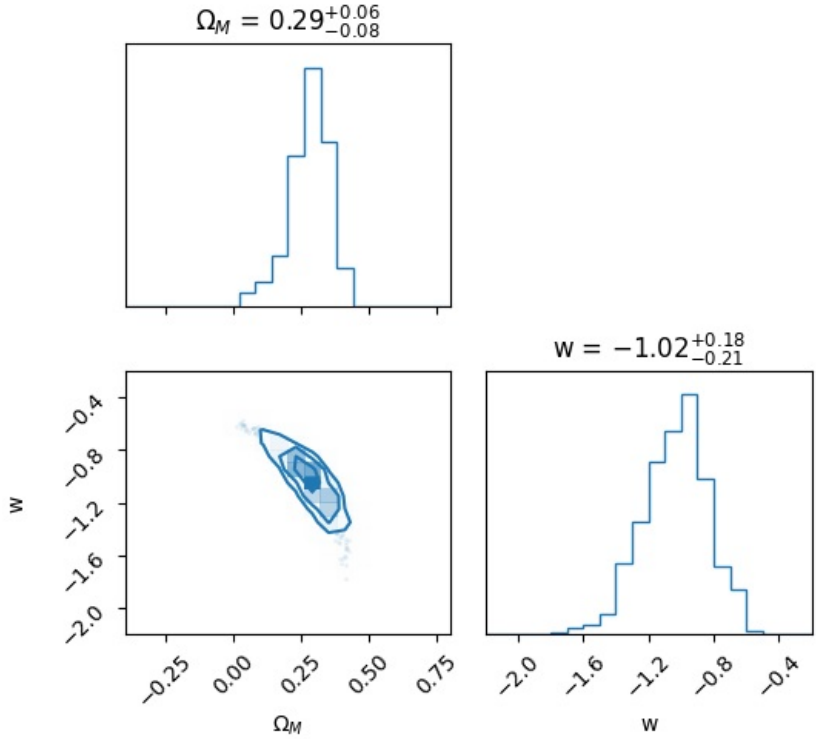
Sampled Parameters from MCMC, Thinned Chain 4

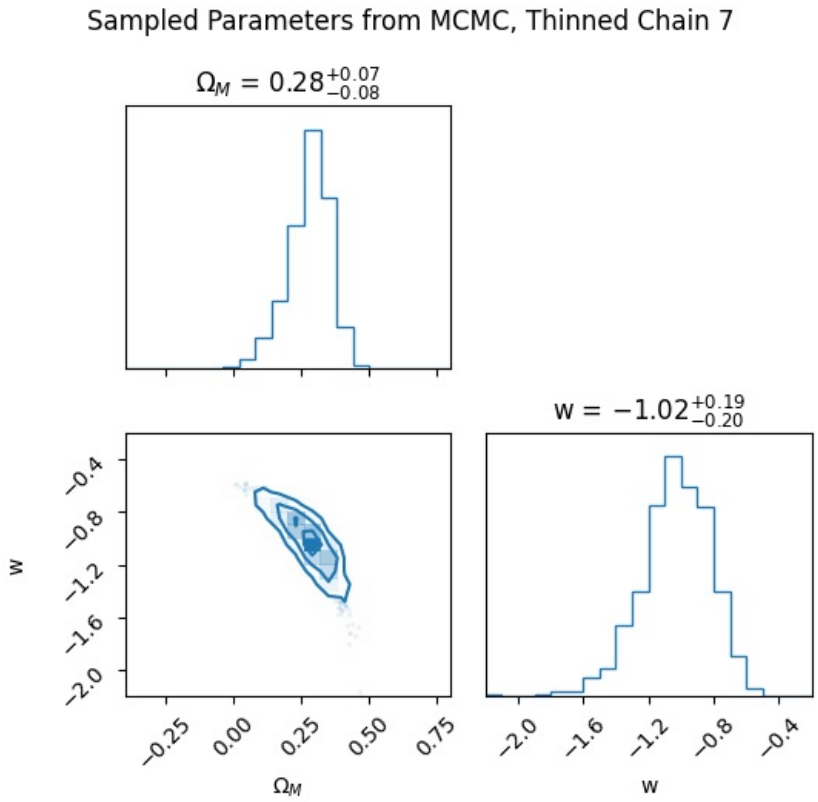


Sampled Parameters from MCMC, Thinned Chain 5



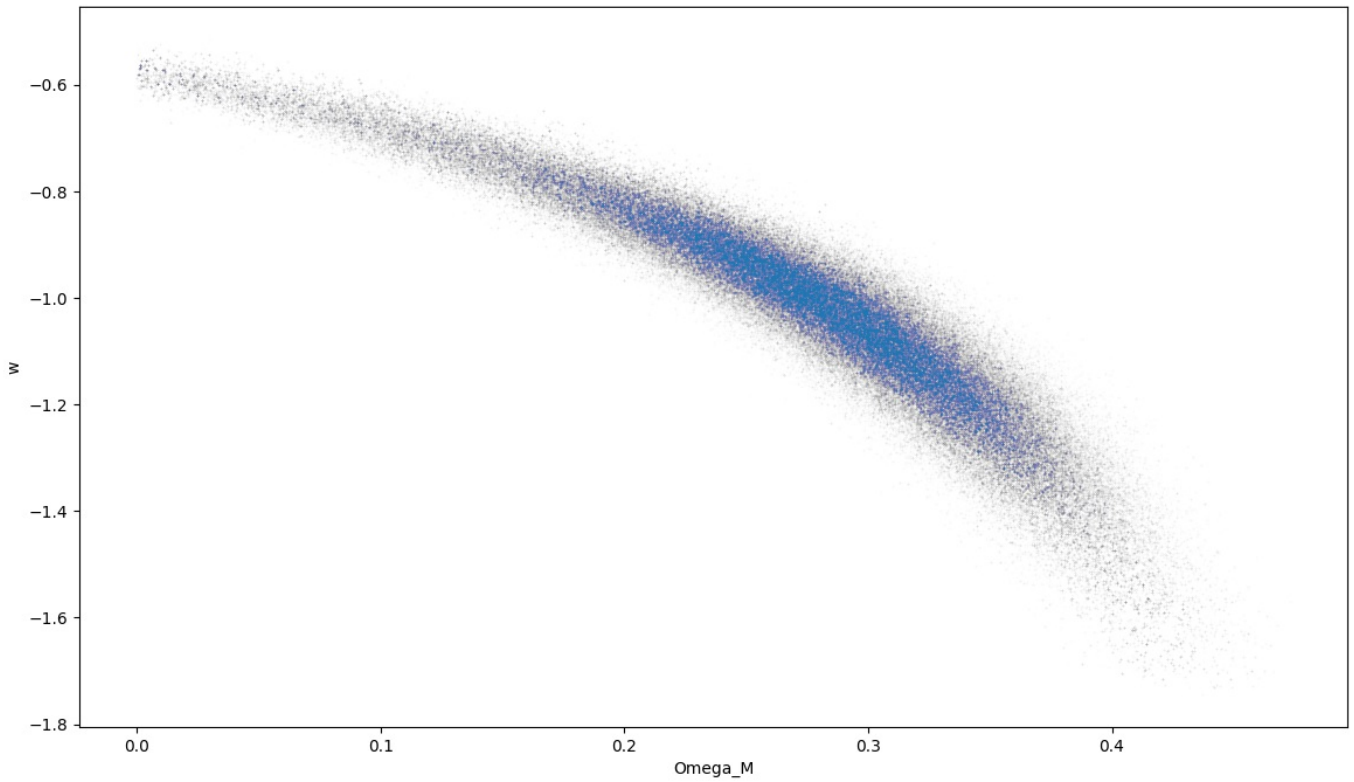
Sampled Parameters from MCMC, Thinned Chain 6





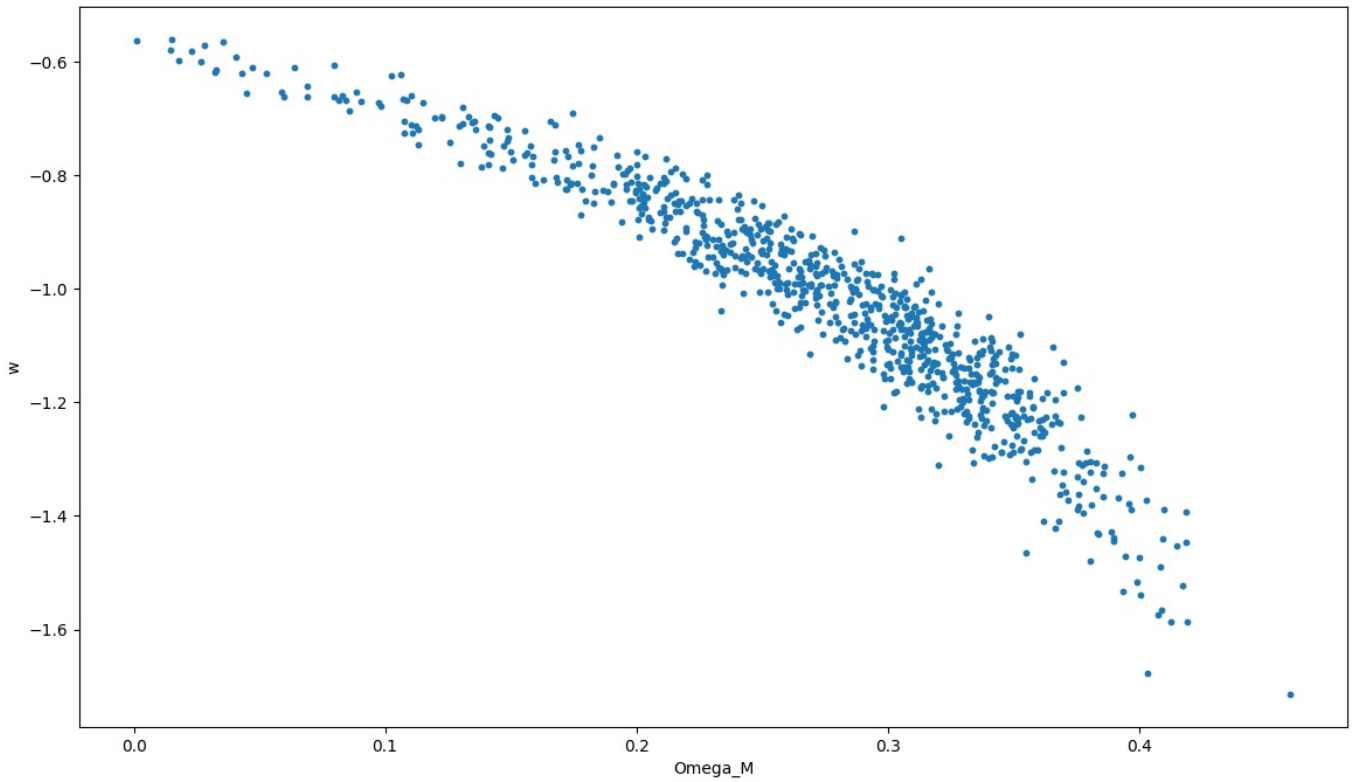
In [208…

```
# Unthinned Scatter Plot All Chains

fig = plt.figure(figsize = (14, 8))
plt.scatter(all_chains_Omega_M[0,:], all_chains_w[0,:], s = 0.1, alpha = 0.015)
plt.xlabel('Omega_M')
plt.ylabel('w')
plt.show()
```



In [209…

```
# Thinned Scatter Plot All Chains

fig = plt.figure(figsize = (14, 8))
plt.scatter(thinned_all_chains_Omega_M_values[0,:], thinned_all_chains_w_values[0,:], s = 10, alpha = 1)
plt.xlabel('Omega_M')
plt.ylabel('w')
plt.show()
```



In [116…

```
# Model
# with pm.Model() as model:
#     Omega_M = pm.Uniform('Omega_M', lower=0, upper=1)
#     Omega_DE = 1 - Omega_M

#     step = pm.Metropolis()

#     trace = pm.sample(50000, step=step, initvals = {'Omega_M': 0.28}, return_inferencedata=False, tune=5000, 


# Plot

# fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
```

```
# az.plot_trace(trace, var_names=['Omega_M', 'w'], ax=axes)
# plt.show()

# Posterior predictive checks
# az.plot_posterior(data=trace,var_names=["Omega_M"],figsize=(16,4))
# plt.show()

# Summary of findings
# summary = az.summary(trace, round_to=2)
# print(summary)


# Omega_M_samples, w_samples, M_n_samples = parameter_arrays(samples, step_manual, start_Omega_M, start_w, star
# print(Omega_M_samples[990:])
# print(w_samples[990:])
# print(M_n_samples[990:])

# def parameter_arrays(samples, step, new_Omega_M, new_w, new_M_n):
#     Omega_M_vals = []
#     w_vals = []
#     M_n_vals = []

#     for _ in range(samples):
#         # Propose new values
#         new_Omega_M = new_Omega_M + np.random.normal(scale=step)
#         new_w = new_w + np.random.normal(scale=step)
#         new_M_n = new_M_n + np.random.normal(scale=step)

#         Omega_M_vals = np.append(Omega_M_vals, new_Omega_M)
#         w_vals = np.append(w_vals, new_w)
#         M_n_vals = np.append(M_n_vals, new_M_n)

#     return Omega_M_vals, w_vals, M_n_vals
```

In [747…

```
# Select a proposal function that defines how to step across data points (decides on what grounds to reject/acc
# Pick a starting point, use proposal function to move to the next point/stay at current point
# Repeat until model convergence

# Burn-in stage - Decide how many of the initially selected data point acceptances/rejections to omit from infe
# PYMC example - trace3 = pm.sample(#samples = N, step=[H388],tune=5000,chains=1) (tune out first 5000 values)
# If a different posterior analysis is observed after running the model, use a larger burn-in period, or change
# Chain should start at expected peak of model landscape (i.e start = pm.find_MAP(fmin=scipy.optimize.fmin_powe

# Ensure that the model is efficient, and doesn't accept/reject values disproportionately
# Possibility of reparameterizing model (i.e multiplying/dividing one parameter by the other, and leaving the o

# Ensure "mixing" of chain
# Example - thinning the chain by removing every N value (N > 10 not necessary)
# Example - running multiple chain, comparing convergence between each one, and determining the general cause b
```

In [748…

```
# FIM Case 2

def fisher_matrix_2(z, m_err, Omega_M, Omega_DE, M_n, Omega_K):
    # Define parameters
    params = {'Omega_M': Omega_M, 'Omega_DE': Omega_DE, 'w': w, 'M_n': M_n, 'Omega_K': Omega_K}
    vary_params_flat = ['Omega_M', 'Omega_DE', 'Omega_K', 'M_n']
    param_keys = list(vary_params_flat)
    n_params = len(param_keys)
    fisher = np.zeros((n_params, n_params))
    h = 1e-5

    for i in range(n_params):
        for j in range(n_params):
            base_params = params.copy()

            # Modify parameters i and j
            base_params[param_keys[i]] += h
            m_th1 = m_th(z, **base_params)

            base_params[param_keys[i]] -= 2 * h
            m_th2 = m_th(z, **base_params)

            # Reset to original
            base_params[param_keys[i]] += h

            # Central difference for partial derivatives
            dm_dpi = (m_th1 - m_th2) / (2 * h)

            if i == j:  # Diagonal elements, derivative wrt same parameter
                fisher[i, j] = np.sum((dm_dpi**2) / (m_err**2))
            else:  # Off-diagonal elements, mixed partial derivatives
                base_params[param_keys[j]] += h
                m_th1 = m_th(z, **base_params)
```

```
                base_params[param_keys[j]] -= 2 * h
                m_th2 = m_th(z, **base_params)

                dm_dpj = (m_th1 - m_th2) / (2 * h)
                fisher[i, j] = np.sum((dm_dpi * dm_dpj) / (m_err**2))

    return fisher

F_flat = fisher_matrix_2(z, m_err, Omega_M, Omega_DE, M_n, Omega_K)

print("Fisher Matrix (Flat Universe):")
print(F_flat)
```

```
Fisher Matrix (Flat Universe):
[[ 38622.96362906  25691.94522167  41583.98284445 -26881.4705564 ]
 [ 25691.94522196  18692.43960121  28994.78991589 -18959.0305462 ]
 [ 41583.98284445  28994.78991557  45888.27340232 -29838.92600466]
 [-26881.4705564  -18959.0305463  -29838.92600466  19434.0772081 ]]
```