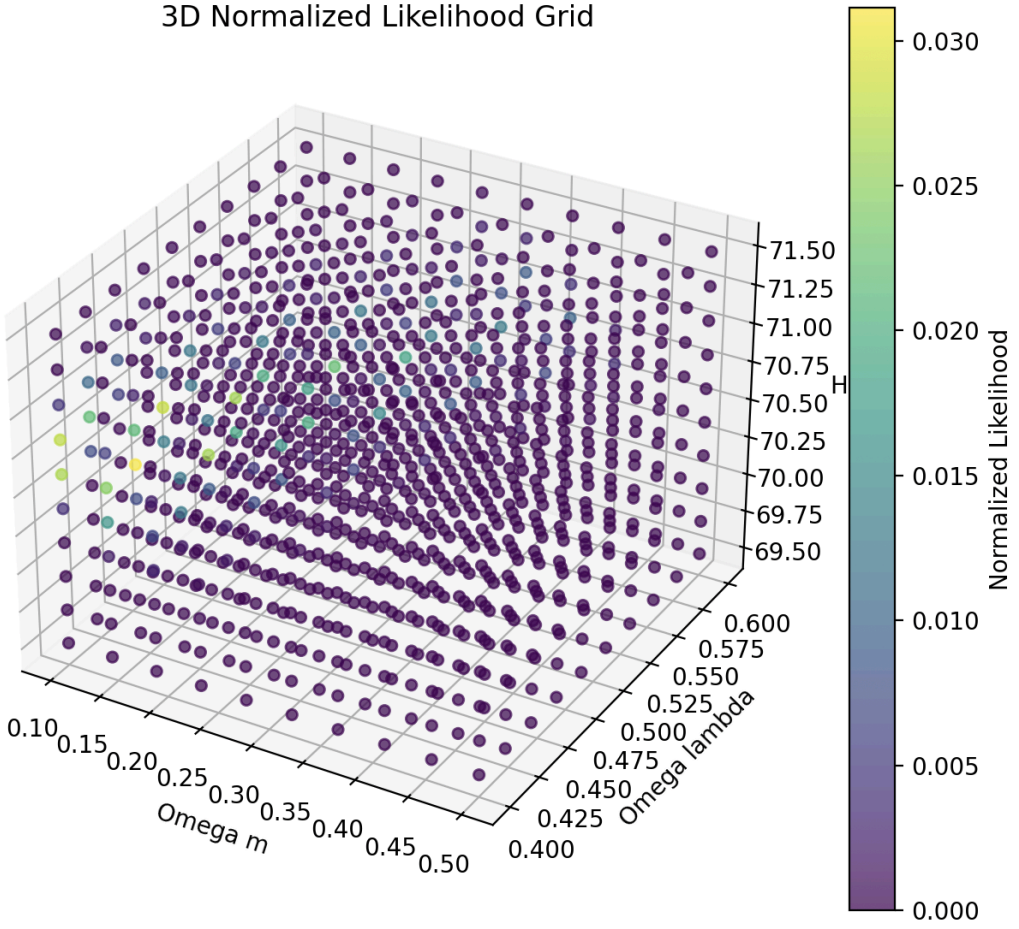
Task 4.1 Likelihood grid

First bullet point: 3D grid for a full model

To construct a three-dimensional likelihood grid for a full cosmological model from Unit 4, I developed the function ***compute_likelihood_grid***. This function initialises a 3D grid by iterating through 10 sample points for each parameter, computing the likelihood using the ***Likelihood*** class from ***unit4.py***, and normalising the results.

The function ***plot_3d_likelihood*** then visualises this grid in three dimensions using Matplotlib's ***scatter*** function. Each point in the grid is color-coded based on its likelihood value, providing an intuitive representation of the likelihood landscape.



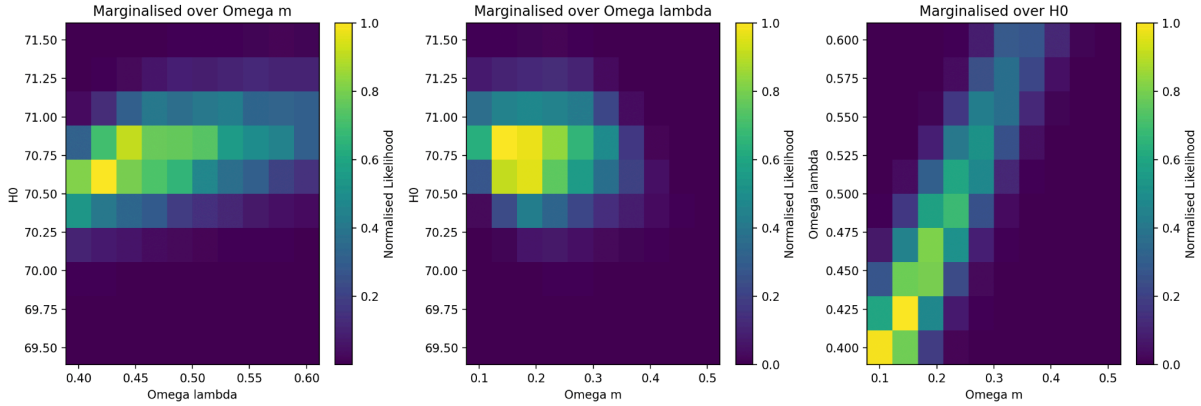
In this resulting grid, each point corresponds to a specific parameter set, with color intensity indicating the normalised likelihood value. Lighter colors represent higher likelihoods, while darker colors indicate lower likelihoods. This visualisation helps identify the most probable parameter combinations, showing regions of high likelihood in the parameter space.

Second bullet point: Three 2D & 1D grids

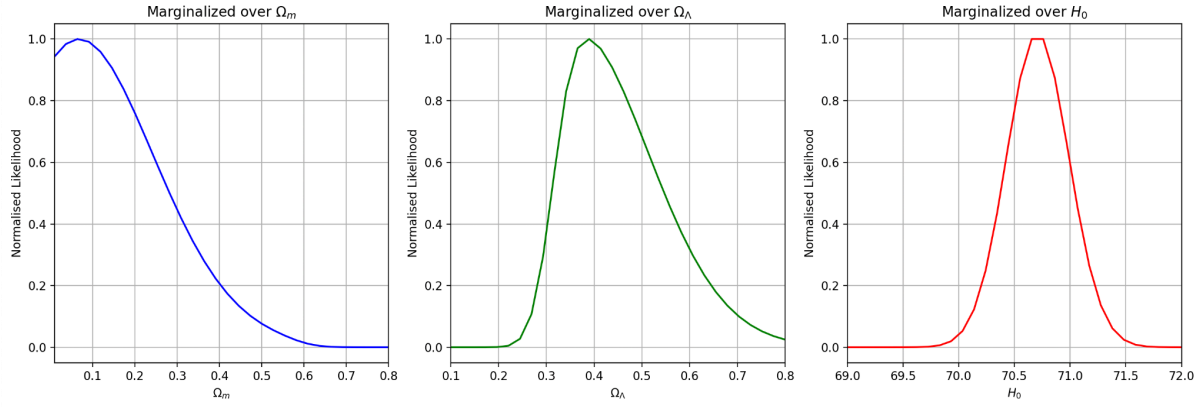
To construct 2D marginalised likelihood grids, I employed Simpson's rule for numerical integration via the functions:

- ***marginalize_over_omega_m***
- ***marginalize_over_omega_lambda***
- ***marginalize_over_h0***

Each function integrates over one parameter at a time, resulting in three 2D likelihood grids that retain the relationships between the remaining two parameters. For visualisation, I used Matplotlib's ***pcolormesh*** function, which explicitly associates x and y coordinates with the data points. Unlike ***imshow***, it does not flip the axes–the first axis (rows) is mapped to the x-axis, and the second axis (columns) is mapped to the y-axis, preserving the natural array structure. ***pcolormesh*** function treats the data as a grid of quadrilaterals, where each point defines a corner of a square, rather than its center.



The above grids represent the 2D marginalised likelihood distributions, where each subplot shows the likelihood distribution after integrating over one parameter. The three panels correspond to likelihoods marginalised over Ω_m, Ω_Λ, and H_0 respectively. Brighter regions indicate higher likelihoods, revealing the most probable regions in the two-dimensional parameter spaces.

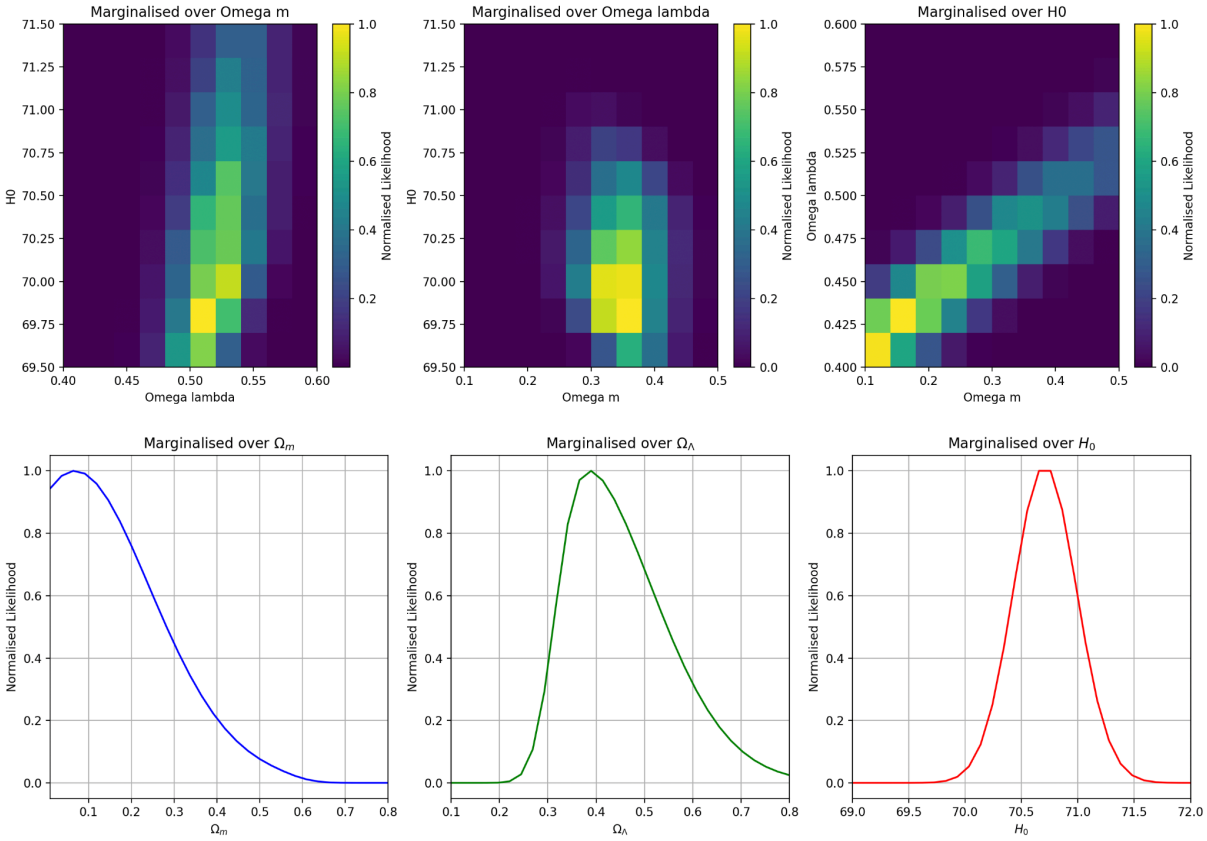


The above 1D marginalised likelihoods are obtained by integrating over two parameters. The three plotted curves represent the likelihood distributions for Ω_m, Ω_Λ, and H_0. The peak positions indicate the most probable values, while the curve spread reflects the uncertainty in each parameter estimate. Notably, the sharp peak in H_0 shows strong constraints on this

parameter, while the broader distributions for Ω_m and Ω_Λ indicate greater flexibility in their values.

Third bullet point: Plot 2D & 1D grids using the imshow function

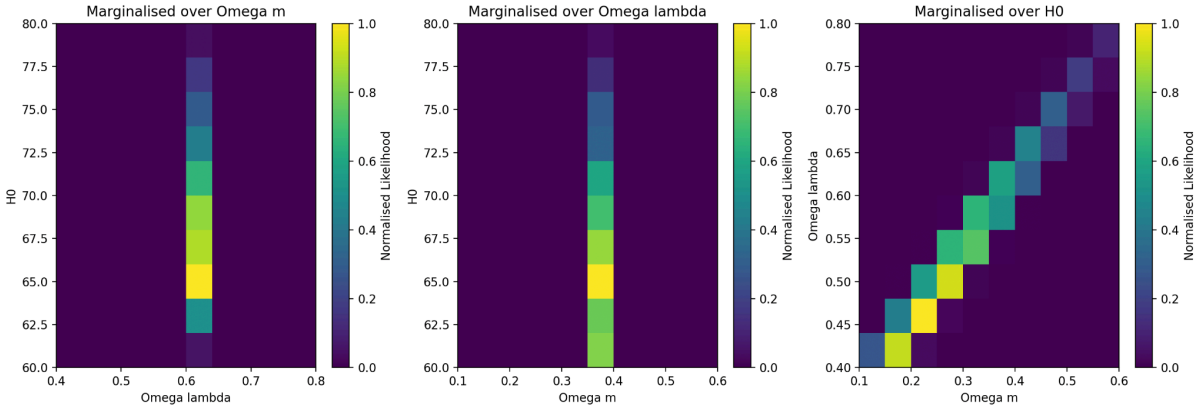
Next, I perform the same process as in the second bullet point but use Matplotlib's ***imshow*** function instead. Unlike ***pcolormesh***, ***imshow*** treats the data as a pixel-based image and directly displays a 2D array. It follows a (row, column) → (y, x) mapping, meaning that arr[i, j] is plotted at (j, i) instead of (i, j), flipping the axes. ***imshow*** assumes that each data point represents the center of a square pixel, and that all pixels are uniformly sized and evenly spaced.



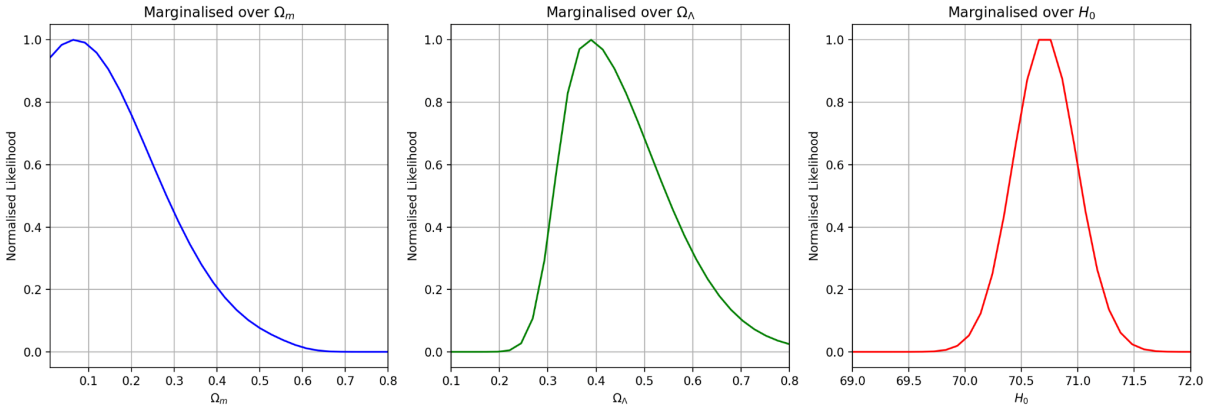
As seen in the above grids, the results using ***imshow*** are nearly identical to those obtained with ***pcolormesh***.

Fourth bullet point: Experiment with the ranges of the parameters

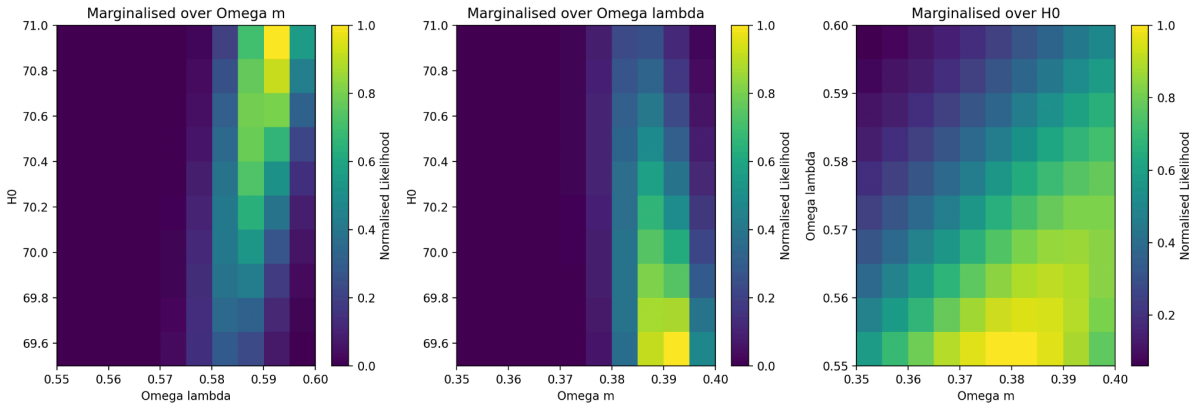
Wide parameter range: $\Omega_m = [0.1,\ 0.6]$, $\Omega_\Lambda = [0.4,\ 0.8]$, $H_0 = [60,\ 80]$



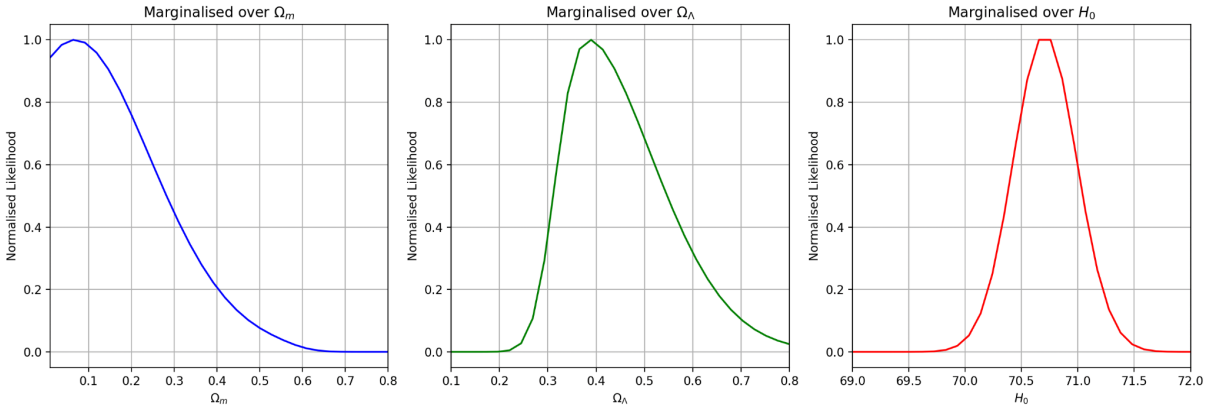
As seen in the above grids, the likelihood function appears as a sharp spike along that axis.



Wide parameter range: $\Omega_m = [0.35,\ 0.4]$, $\Omega_\Lambda = [0.55,\ 0.6]$, $H_0 = [69.5,\ 71.0]$

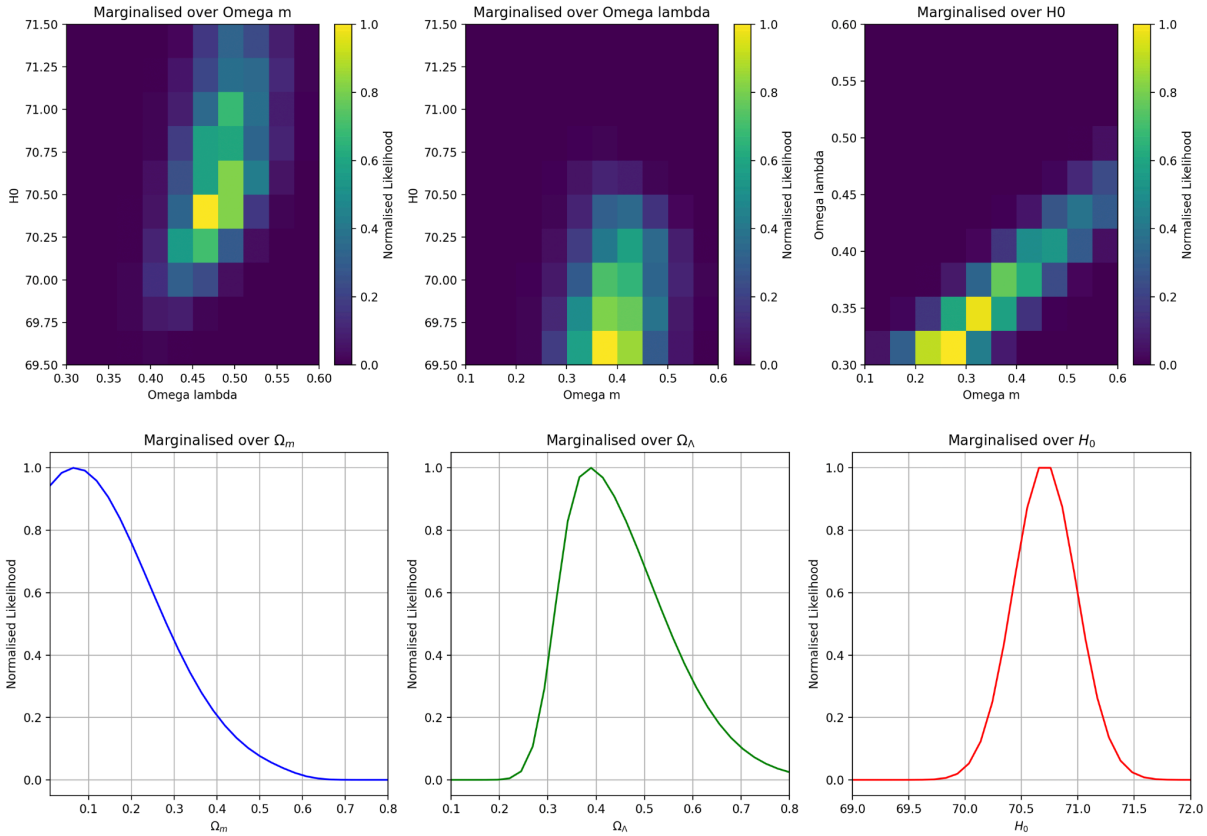


Conversely, when the parameter range is too narrow, the likelihood distribution becomes very broad and flat, as shown in the above grids.



Finding the optimal parameter range

By further investigating the most suitable parameter range, I found that the range $\Omega_m = [0.1,\ 0.6]$, $\Omega_\Lambda = [0.3,\ 0.6]$, $H_0 = [69.5,\ 71.5]$ provides the most well-defined likelihood distributions, as shown below. This aligns with my initial expectation from the 1D marginalized likelihood grids, where H_0 exhibits strong constraints, requiring a much narrower range than other parameters.

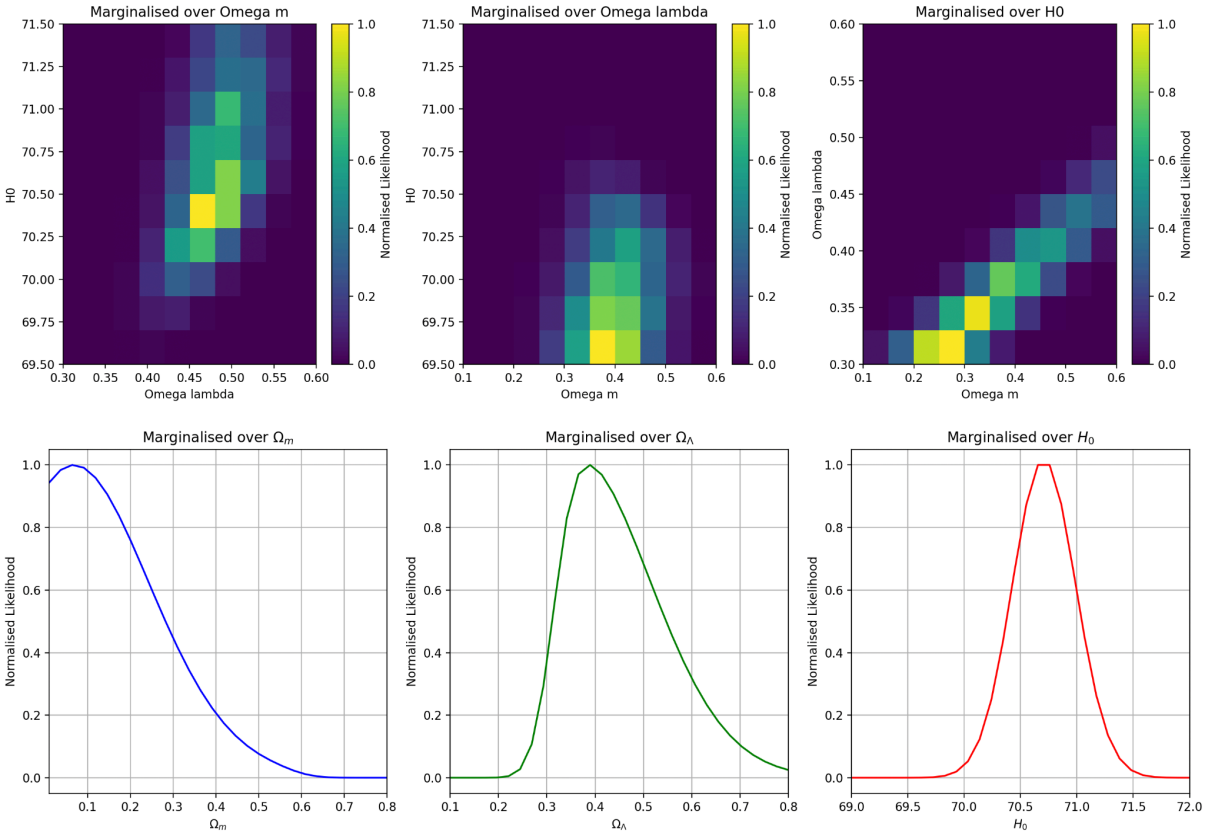


Fifth bullet point: Increased sample points lead to longer computation time

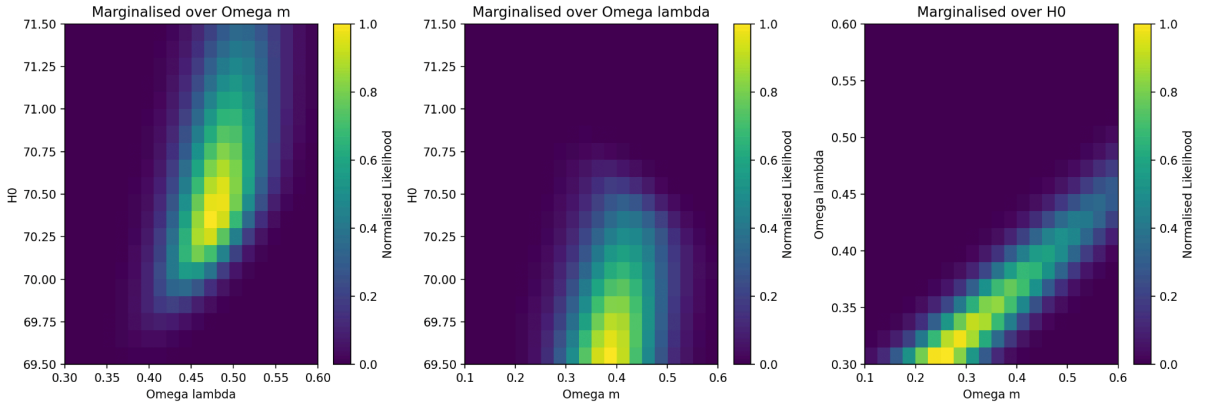
To examine the impact of sample size on computation time and likelihood grids, I tested five different numbers of sample points: 10, 20, 30, and 50, while keeping the same parameter range, $\Omega_m = [0.1,\ 0.6]$, $\Omega_\Lambda = [0.3,\ 0.6]$, $H_0 = [69.5,\ 71.5]$.

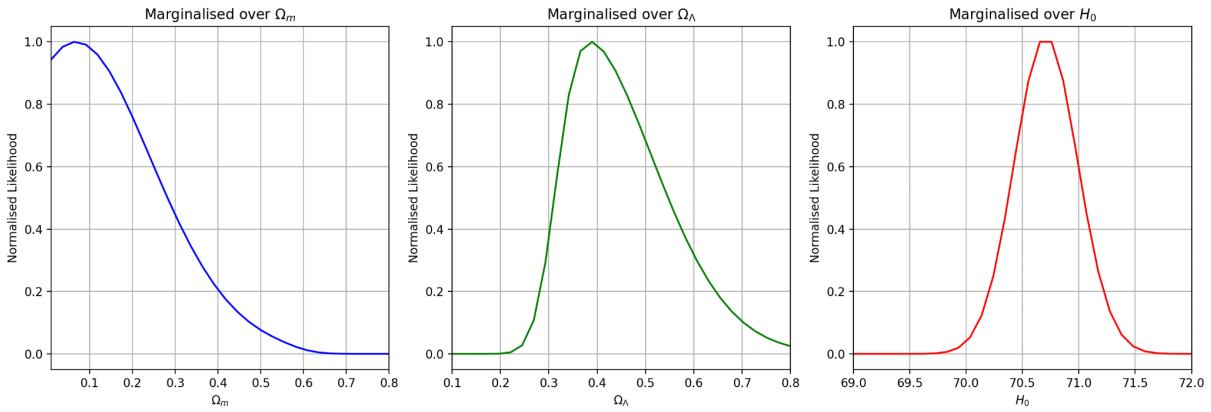
I measured the likelihood computation time using Python's time function, which records the elapsed wall-clock time between two points in the execution.

- 10 sample points: Computation time 2.25 seconds, producing the following grids:

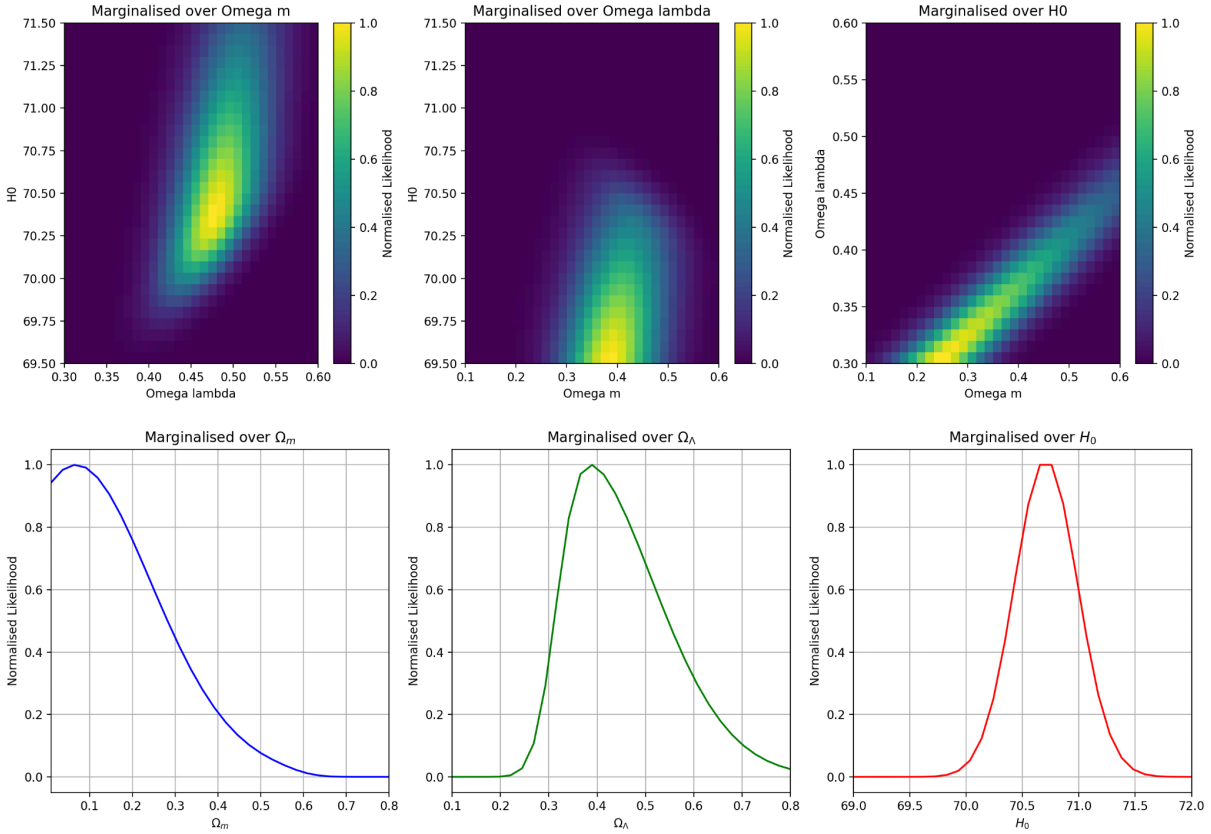


- 20 sample points: Computation time 16.82 seconds, producing more accurate and detailed grids at the cost of increased computation time.

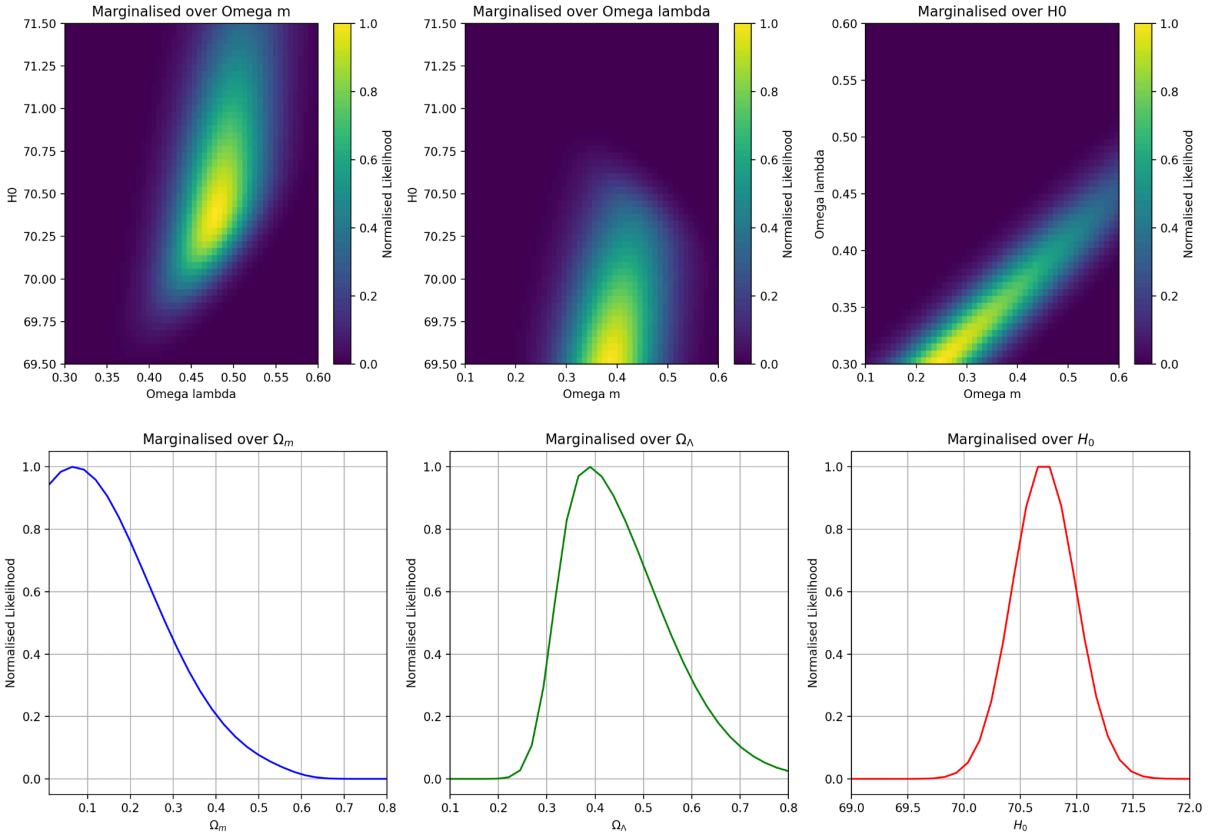




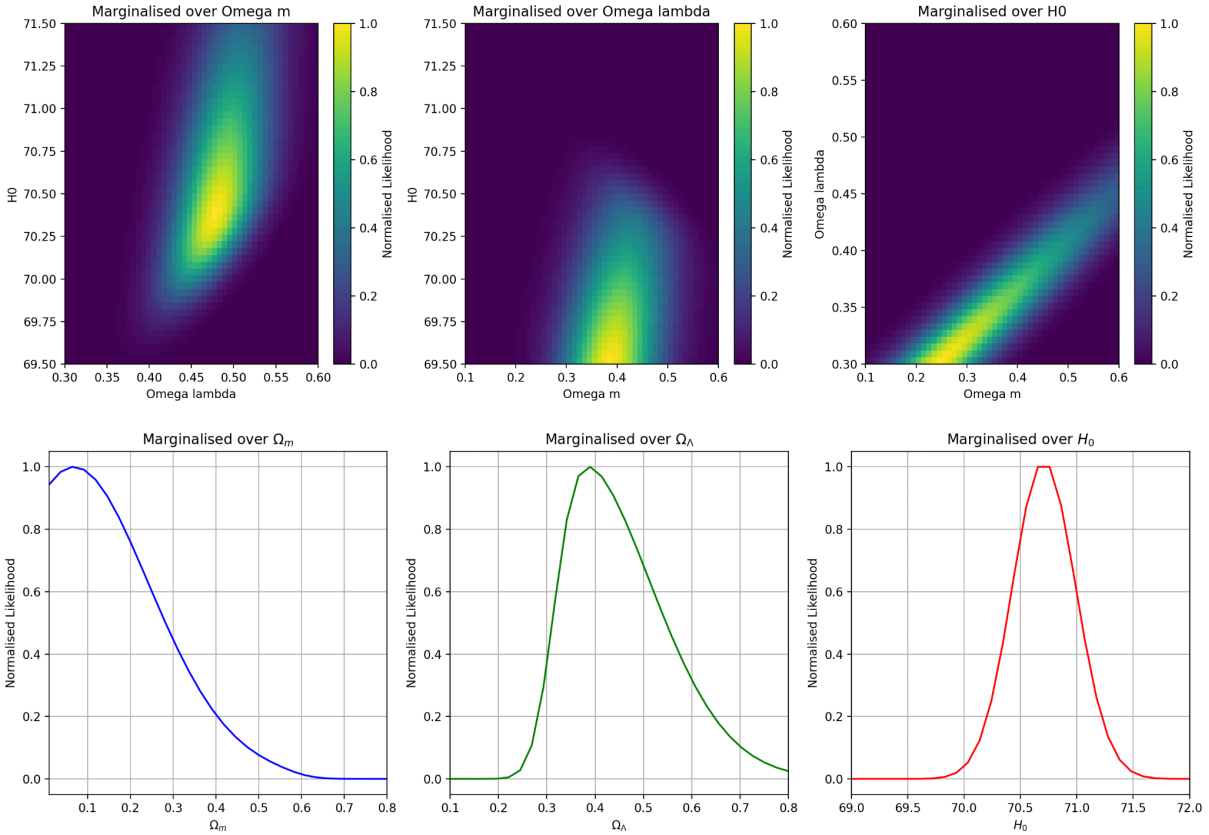
- 30 sample points: Computation time 56.25 seconds, further improving grid resolution while significantly increasing computation time.



- 40 sample points: Computation time 133.01 seconds, providing further improvements in grid resolution but with a significant increase in computation time.



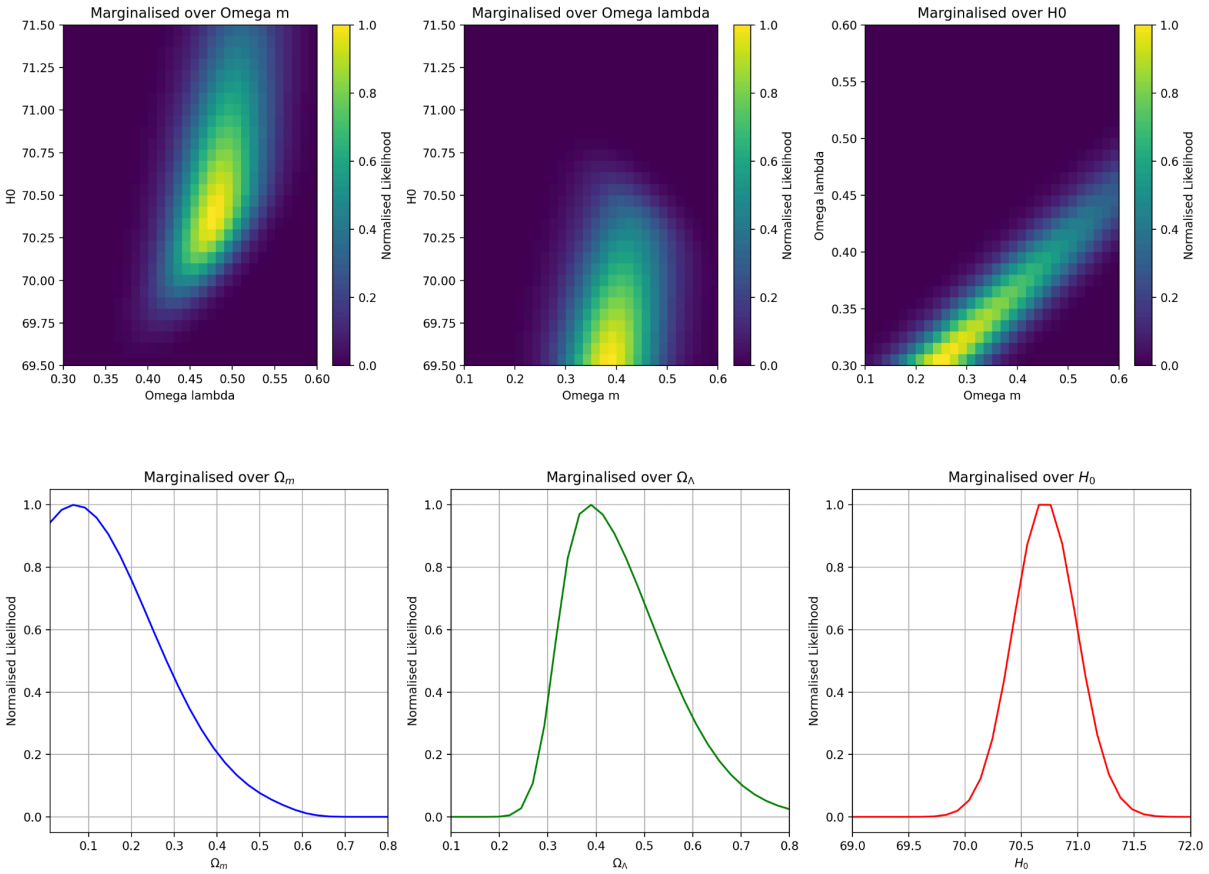
- 50 sample points: Computation time 266.47 seconds, enhancing grid resolution even further while doubling the computation time compared to 40 sample points.



These results demonstrate the trade-off between accuracy and computational efficiency—increasing sample points improves grid resolution but leads to an exponential increase in computation time.

Six bullet point: Conclusion

Throughout these bullet points, we have explored various combinations of parameter ranges and sample numbers. Considering the trade-off between likelihood grid resolution and computation time, 30 sample numbers with $\Omega_m = [0.1,\ 0.6]$, $\Omega_\Lambda = [0.3,\ 0.6]$, $H_0 = [69.5,\ 71.5]$, which takes 56.25 seconds, appears to be the optimal balance.



Task 4.2 Metropolis

Design a generic Metropolis class

The Metropolis class contains attributes including:

- ***likelihood_function***: A function computing the likelihood of a given parameter set.
- ***current_params***: An array storing the current values of Ω_m, Ω_Λ, and H_0.
- ***step_sizes***: Step sizes for proposing new parameter values.
- ***num_samples***: Number of MCMC samples to generate.

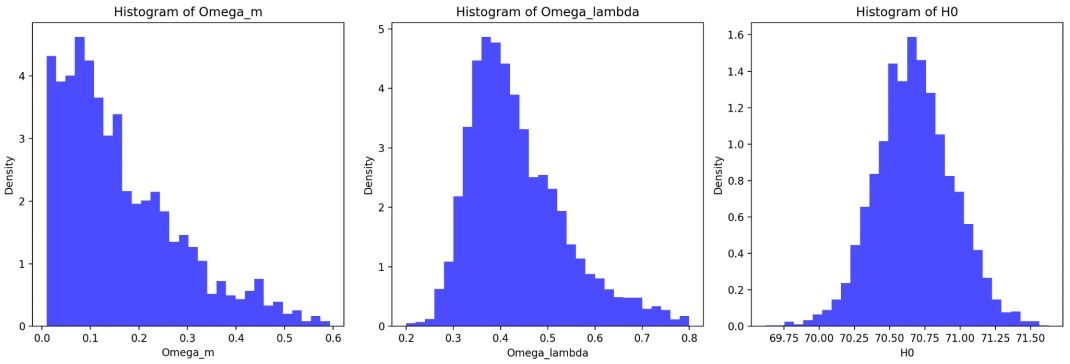
- ***burn_in***: The number of initial samples discarded to allow the chain to stabilise.

The Metropolis class contains methods including:

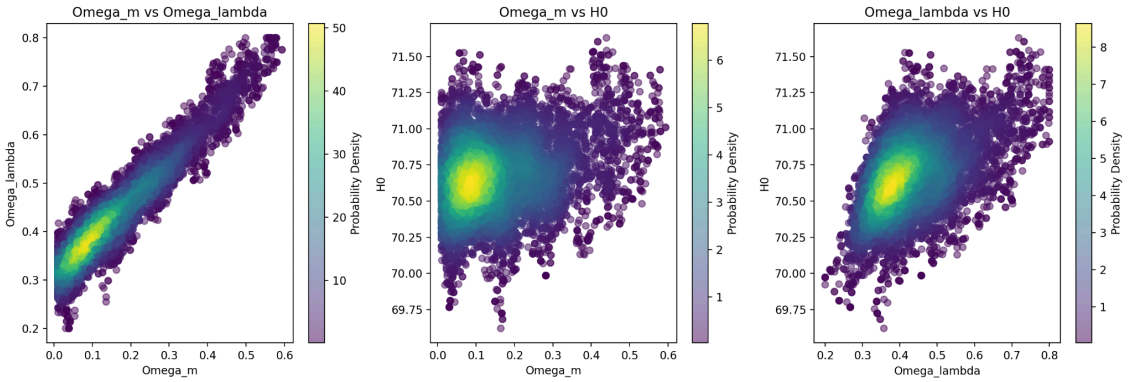
- ***propose_new_params()***: Generates a new set of parameters by adding a Gaussian random step to the current parameters while ensuring they remain within predefined bounds.
- ***run()***: Iterates through the MCMC chain, accepting or rejecting proposed moves based on the Metropolis criterion. The likelihood function is computed for both current and proposed parameters, and a Metropolis acceptance rule is applied.
- The algorithm discards an initial set of samples (**burn_in**) before recording the final results.

Use the Metropolis algorithm to analyse the supernova dataset

- **1D Histograms**: The marginal distributions of Ω_m, Ω_Λ, and H_0 are visualised using histograms. Each histogram shows the frequency density of the sampled values. 10,000 samples are used to plot the histograms.

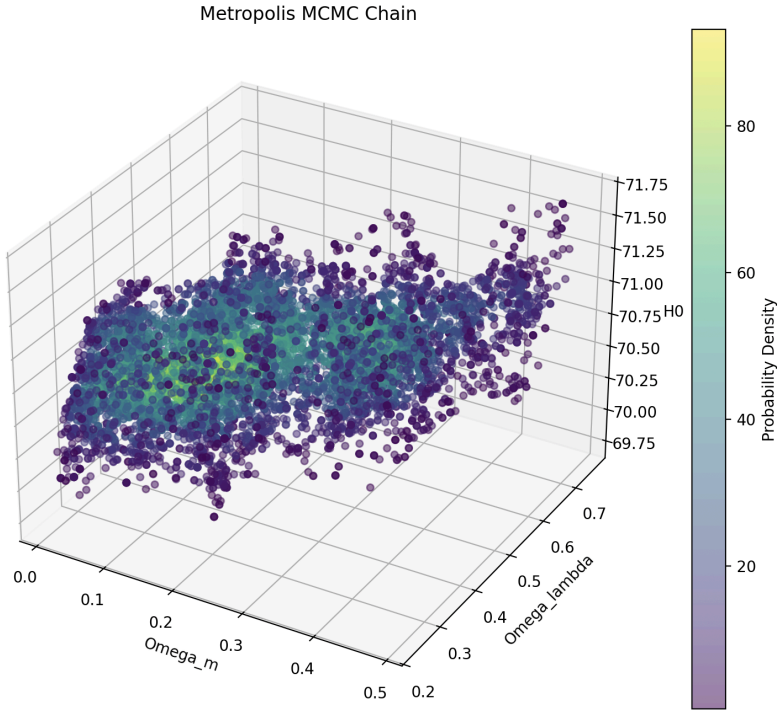


- **2D Scatter Density Plots**: The relationships between pairs of parameters (Ω_m vs Ω_Λ, Ω_Λ, vs H_0, Ω_m vs H_0) are shown using scatter plots, where color represents probability density. The density is computed using Kernel Density Estimation (KDE) via gaussian_kde.



- **3D Scatter Plot**: The full distribution in Ω_m – Ω_Λ–H_0 space is visualised with a 3D scatter plot, where the color of each point represents probability density computed

via gaussian_kde.



Compare Metropolis results with the likelihood grid

Both methods yield similar conclusions about the best-fit regions of parameter space.

- **1D Marginals**: Both the Metropolis histograms and the likelihood grid produce well-defined bell curves for the distributions of Ω_m, Ω_Λ, and H_0.
- **2D Probability Distributions**: The likelihood grid provides a more structured representation due to grid-based evaluation, whereas the Metropolis scatter plots show more dispersion but similar clustering.
- **3D Scatter Plot**: The likelihood grid systematically samples a structured probability landscape, while the Metropolis chain generates a cloud of samples where high-density regions align with the grid's probability peaks.

The likelihood grid provides an exact evaluation over a fixed parameter space, while the Metropolis algorithm provides a sample-based estimate of the posterior distribution, making it sensitive to initial conditions and step sizes.