🔭 Assignment: Measuring Cosmological Parameters Using Type Ia Supernovae

In this assignment, you'll analyze observational data from the Pantheon+SH0ES dataset of Type Ia supernovae to measure the Hubble constant H_0 and estimate the age of the universe. You will:

- Plot the Hubble diagram (distance modulus vs. redshift)
- Fit a cosmological model to derive H_0 and Ω_m
- Estimate the age of the universe
- Analyze residuals to assess the model
- Explore the effect of fixing Ω_m
- Compare low-z and high-z results

Let's get started!

📦 Getting Started: Setup and Libraries

Before we dive into the analysis, we need to import the necessary Python libraries:

- `numpy`, `pandas` — for numerical operations and data handling
- `matplotlib` — for plotting graphs
- `scipy.optimize.curve_fit` and `scipy.integrate.quad` — for fitting cosmological models and integrating equations
- `astropy.constants` and `astropy.units` — for physical constants and unit conversions

Make sure these libraries are installed in your environment. If not, you can install them using:

```
pip install numpy pandas matplotlib scipy astropy
```

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.integrate import quad
from astropy.constants import c
from astropy import units as u
```

📂 Load the Pantheon+SH0ES Dataset

We now load the observational supernova data from the Pantheon+SH0ES sample. This dataset includes calibrated distance moduli μ, redshifts corrected for various effects, and

uncertainties.

Instructions:

- Make sure the data file is downloaded from Pantheon dataset and available locally.
- We use `delim_whitespace=True` because the file is space-delimited rather than comma-separated.
- Commented rows (starting with `#`) are automatically skipped.

We will extract:

- `zHD`: Hubble diagram redshift
- `MU_SH0ES`: Distance modulus using SH0ES calibration
- `MU_SH0ES_ERR_DIAG`: Associated uncertainty

More detailed column names and the meanings can be referred here:

> Finally, we include a combined file of all the fitted parameters for each SN, before and after light-curve cuts are applied. This is in the format of a .FITRES file and has all the meta-information listed above along with the fitted SALT2 parameters. We show a screenshot of the release in Figure 7. Here, we give brief descriptions of each column. **CID** – name of SN. **CIDint** – counter of SNe in the sample. **IDSURVEY** – ID of the survey. **TYPE** – whether SN Ia or not – all SNe in this sample are SNe Ia. **FIELD** – if observed in a particular field. **CUTFLAG_SNANA** – any bits in light-curve fit flagged. **ERRFLAG_FIT** – flag in fit. **zHEL** – heliocentric redshift. **zHELERR** – heliocentric redshift error. **zCMB** – CMB redshift. **zCMBERR** – CMB redshift error. **zHD** – Hubble Diagram redshift. **zHDERR** – Hubble Diagram redshift error. **VPEC** – peculiar velocity. **VPECERR** – peculiar-velocity error. **MWEBV** – MW extinction. **HOST_LOGMASS** – mass of host. **HOST_LOGMASS_ERR** – error in mass of host. **HOST_sSFR** – sSFR of host. **HOST_sSFR_ERR** – error in sSFR of host. **PKMJDINI** – initial guess for PKMJD. **SNRMAX1** – First highest signal-to-noise ratio (SNR) of light curve. **SNRMAX2** – Second highest SNR of light curve. **SNRMAX3** – Third highest SNR of light curve. **PKMJD** – Fitted PKMJD. **PKMJDERR** –

In [2]:
```
# Local file path
file_path = "D:/programs/ISA/Project 2/Pantheon+SH0ES.dat"

# Load the file
df = pd.read_csv(file_path, delim_whitespace=True, comment='#')

# See structure
df.head()
```

```
C:\Users\Ansh\AppData\Local\Temp\ipykernel_3160\943745630.py:5: FutureWarning: The
'delim_whitespace' keyword in pd.read_csv is deprecated and will be removed in a fut
ure version. Use ``sep='\s+'`` instead
  df = pd.read_csv(file_path, delim_whitespace=True, comment='#')
```

Out[2]:

	CID	IDSURVEY	zHD	zHDERR	zCMB	zCMBERR	zHEL	zHELERR	m
0	2011fe	51	0.00122	0.00084	0.00122	0.00002	0.00082	0.00002	9
1	2011fe	56	0.00122	0.00084	0.00122	0.00002	0.00082	0.00002	9
2	2012cg	51	0.00256	0.00084	0.00256	0.00002	0.00144	0.00002	1
3	2012cg	56	0.00256	0.00084	0.00256	0.00002	0.00144	0.00002	1
4	1994DRichmond	50	0.00299	0.00084	0.00299	0.00004	0.00187	0.00004	1

5 rows × 47 columns

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1701 entries, 0 to 1700
Data columns (total 47 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   CID                   1701 non-null   object 
 1   IDSURVEY              1701 non-null   int64  
 2   zHD                   1701 non-null   float64
 3   zHDERR                1701 non-null   float64
 4   zCMB                  1701 non-null   float64
 5   zCMBERR               1701 non-null   float64
 6   zHEL                  1701 non-null   float64
 7   zHELERR               1701 non-null   float64
 8   m_b_corr              1701 non-null   float64
 9   m_b_corr_err_DIAG     1701 non-null   float64
 10  MU_SH0ES              1701 non-null   float64
 11  MU_SH0ES_ERR_DIAG     1701 non-null   float64
 12  CEPH_DIST             1701 non-null   float64
 13  IS_CALIBRATOR         1701 non-null   int64  
 14  USED_IN_SH0ES_HF      1701 non-null   int64  
 15  c                     1701 non-null   float64
 16  cERR                  1701 non-null   float64
 17  x1                    1701 non-null   float64
 18  x1ERR                 1701 non-null   float64
 19  mB                    1701 non-null   float64
 20  mBERR                 1701 non-null   float64
 21  x0                    1701 non-null   float64
 22  x0ERR                 1701 non-null   float64
 23  COV_x1_c              1701 non-null   float64
 24  COV_x1_x0             1701 non-null   float64
 25  COV_c_x0              1701 non-null   float64
 26  RA                    1701 non-null   float64
 27  DEC                   1701 non-null   float64
 28  HOST_RA               1701 non-null   int64  
 29  HOST_DEC              1701 non-null   int64  
 30  HOST_ANGSEP           1701 non-null   float64
 31  VPEC                  1701 non-null   float64
 32  VPECERR               1701 non-null   int64  
 33  MWEBV                 1701 non-null   float64
 34  HOST_LOGMASS          1701 non-null   float64
 35  HOST_LOGMASS_ERR      1701 non-null   float64
 36  PKMJD                 1701 non-null   float64
 37  PKMJDERR              1701 non-null   float64
 38  NDOF                  1701 non-null   int64  
 39  FITCHI2               1701 non-null   float64
 40  FITPROB               1701 non-null   float64
 41  m_b_corr_err_RAW      1701 non-null   float64
 42  m_b_corr_err_VPEC     1701 non-null   float64
 43  biasCor_m_b           1701 non-null   float64
 44  biasCorErr_m_b        1701 non-null   float64
 45  biasCor_m_b_COVSCALE  1701 non-null   float64
 46  biasCor_m_b_COVADD    1701 non-null   float64
dtypes: float64(39), int64(7), object(1)
memory usage: 624.7+ KB
```

📓 Preview Dataset Columns

Before diving into the analysis, let's take a quick look at the column names in the dataset. This helps us verify the data loaded correctly and identify the relevant columns we'll use for cosmological modeling.

In [4]:
```
df.columns
df.isnull().sum()
```

Out[4]:
```
CID                    0
IDSURVEY               0
zHD                    0
zHDERR                 0
zCMB                   0
zCMBERR                0
zHEL                   0
zHELERR                0
m_b_corr               0
m_b_corr_err_DIAG      0
MU_SH0ES               0
MU_SH0ES_ERR_DIAG      0
CEPH_DIST              0
IS_CALIBRATOR          0
USED_IN_SH0ES_HF       0
c                      0
cERR                   0
x1                     0
x1ERR                  0
mB                     0
mBERR                  0
x0                     0
x0ERR                  0
COV_x1_c               0
COV_x1_x0              0
COV_c_x0               0
RA                     0
DEC                    0
HOST_RA                0
HOST_DEC               0
HOST_ANGSEP            0
VPEC                   0
VPECERR                0
MWEBV                  0
HOST_LOGMASS           0
HOST_LOGMASS_ERR       0
PKMJD                  0
PKMJDERR               0
NDOF                   0
FITCHI2                0
FITPROB                0
m_b_corr_err_RAW       0
m_b_corr_err_VPEC      0
biasCor_m_b            0
biasCorErr_m_b         0
biasCor_m_b_COVSCALE   0
biasCor_m_b_COVADD     0
dtype: int64
```

🧹 Clean and Extract Relevant Data

To ensure reliable fitting, we remove any rows that have missing values in key columns:

- `zHD`: redshift for the Hubble diagram

- `MU_SH0ES` : distance modulus
- `MU_SH0ES_ERR_DIAG` : uncertainty in the distance modulus

We then extract these cleaned columns as NumPy arrays to prepare for analysis and modeling.

In [5]:
```
# Filter for entries with usable data based on the required columns
data = pd.read_csv(file_path, delim_whitespace=True, comment='#')

# Select the required columns
zHD = data['zHD'].values
mu_obs = data['MU_SH0ES'].values
mu_err = data['MU_SH0ES_ERR_DIAG'].values
required_columns = ['zHD', 'MU_SH0ES', 'MU_SH0ES_ERR_DIAG']
extracted_data = data[required_columns].copy()

# Filter for entries with usable data (no NaNs in required columns)
extracted_data.dropna(subset=required_columns, inplace=True)

print("Data cleaning and extraction complete. First 5 rows of extracted data:")
print(extracted_data.head())
print("\nData information:")
print(extracted_data.info())

extracted_data.to_csv('cleaned_pantheon_data.csv', index=False)
print("\nCleaned and extracted data saved to 'cleaned_pantheon_data.csv'")
```

```
C:\Users\Ansh\AppData\Local\Temp\ipykernel_3160\699438651.py:2: FutureWarning: The
'delim_whitespace' keyword in pd.read_csv is deprecated and will be removed in a fut
ure version. Use ``sep='\s+'`` instead
  data = pd.read_csv(file_path, delim_whitespace=True, comment='#')
Data cleaning and extraction complete. First 5 rows of extracted data:
       zHD  MU_SH0ES  MU_SH0ES_ERR_DIAG
0  0.00122   28.9987           1.516450
1  0.00122   29.0559           1.517470
2  0.00256   30.7233           0.782372
3  0.00256   30.7449           0.799068
4  0.00299   30.7757           0.881212

Data information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1701 entries, 0 to 1700
Data columns (total 3 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   zHD                1701 non-null   float64
 1   MU_SH0ES           1701 non-null   float64
 2   MU_SH0ES_ERR_DIAG  1701 non-null   float64
dtypes: float64(3)
memory usage: 40.0 KB
None

Cleaned and extracted data saved to 'cleaned_pantheon_data.csv'
```

📈 Plot the Hubble Diagram

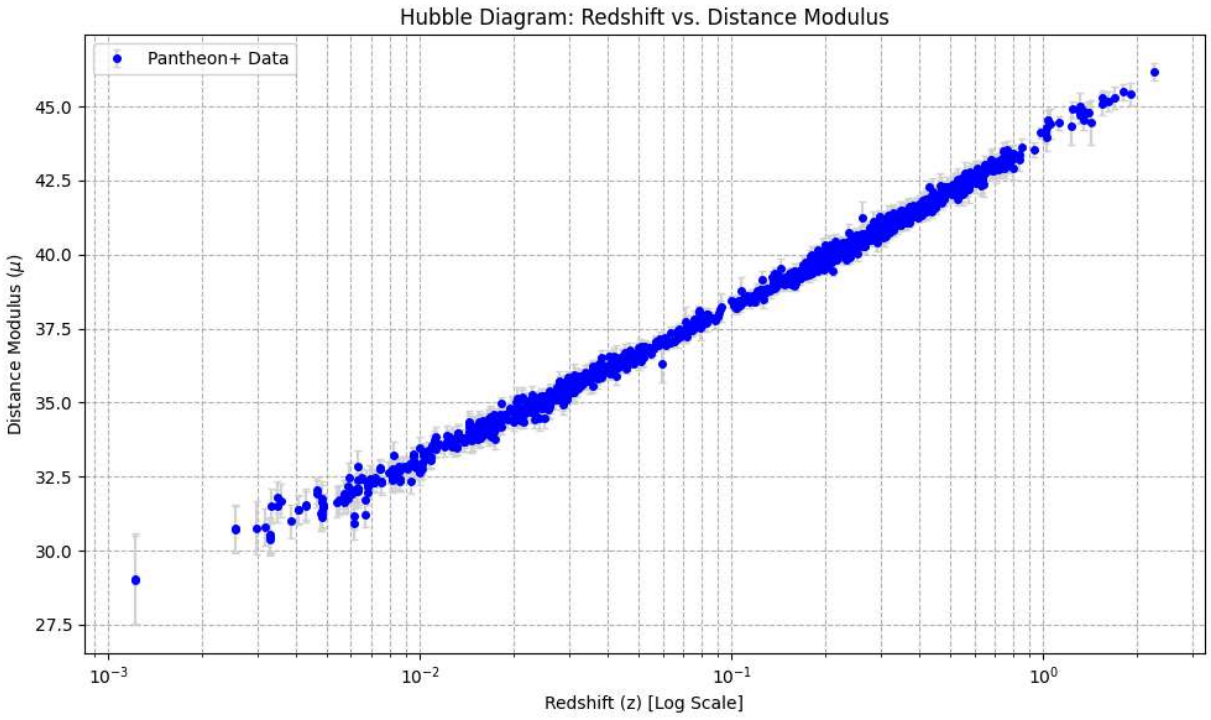
Let's visualize the relationship between redshift z and distance modulus μ, known as the Hubble diagram. This plot is a cornerstone of observational cosmology—it allows us to compare supernova observations with theoretical predictions based on different cosmological models.

We use a logarithmic scale on the redshift axis to clearly display both nearby and distant supernovae.

In [6]:
```
# Write a code to plot the distance modulus and the redshift (x-axis), label them a

#Try using log scale in x-axis
plt.figure(figsize=(10, 6))
plt.errorbar(extracted_data['zHD'], extracted_data['MU_SH0ES'],
             yerr=extracted_data['MU_SH0ES_ERR_DIAG'],
             fmt='o', color='blue', ecolor='lightgray', capsize=2, markersize=4, la
plt.xscale('log')
plt.xlabel('Redshift (z) [Log Scale]')
plt.ylabel('Distance Modulus ($\\mu$)')
plt.title('Hubble Diagram: Redshift vs. Distance Modulus')
plt.grid(True, which="both", ls="--", c='0.7')
plt.legend()
plt.tight_layout()
plt.savefig('hubble_diagram.png')
print("\nHubble Diagram plotted and saved as 'hubble_diagram.png'")
```

```
Hubble Diagram plotted and saved as 'hubble_diagram.png'
```



🧮 Define the Cosmological Model

We now define the theoretical framework based on the flat $\Lambda\mathrm{CDM}$ model (read about the model in wikipedia if needed). This involves:

- The dimensionless Hubble parameter:

$$E(z) = \sqrt{\Omega_m(1+z)^3 + (1-\Omega_m)}$$

- The distance modulus is:

$$\mu(z) = 5\log_{10}(d_L/\mathrm{Mpc}) + 25$$

- And the corresponding luminosity distance :

$$d_L(z) = (1+z)\cdot\frac{c}{H_0}\int_0^z \frac{dz'}{E(z')}$$

These equations allow us to compute the expected distance modulus from a given redshift z, Hubble constant H_0, and matter density parameter Ω_m.

In [7]:
```python
# Define the E(z) for flat LCDM
def E(z, Omega_m):
    """
    E(z) for a flat LCDM cosmology.
    Omega_m: Matter density parameter at present day.
    """
    Omega_lambda = 1.0 - Omega_m # For a flat universe
    return np.sqrt(Omega_m * (1 + z)**3 + Omega_lambda)

# Luminosity distance in Mpc
def luminosity_distance(z, H0, Omega_m):
    """
    Luminosity distance in Mpc.
    z: Redshift
    H0: Hubble constant in km/s/Mpc
    Omega_m: Matter density parameter
    """
    # Integrate 1/E(z') from 0 to z
    integral, _ = quad(lambda zp: 1.0 / E(zp, Omega_m), 0, z)
    dL = (c / H0) * (1 + z) * integral # dL in Mpc
    return dL

def distance_modulus(z, H0, Omega_m):
    """
    Calculates the distance modulus from the luminosity distance.
    mu = 5 log10(DL) + 25
    where DL is in Mpc.
    """
    DL = np.array([luminosity_distance(zi, H0, Omega_m) for zi in z])
    return 5 * np.log10(DL) + 25
```

In [8]:
```python
def dL_integrand(z_prime, omega_m):
    # For a flat Lambda-CDM model (Omega_k = 0), Omega_Lambda = 1 - Omega_m
```

```
    omega_lambda = 1.0 - omega_m
    return 1.0 / np.sqrt(omega_m * (1.0 + z_prime)**3 + omega_lambda)

def luminosity_distance(z, H0, omega_m):
    # Use Astropy's speed of light with units
    c_kms = c.to(u.km / u.s).value  # proper usage of astropy constant

    def E(z, omega_m):
        return np.sqrt(omega_m * (1 + z)**3 + (1 - omega_m))

    def dL_integrand(zp, omega_m):
        return 1.0 / E(zp, omega_m)

    integral, _ = quad(dL_integrand, 0, z, args=(omega_m,))
    dL = (c_kms / H0) * (1 + z) * integral  # Luminosity distance in Mpc
    return dL

def cosmological_model(z, H0, omega_m):
    # Handle z=0 to avoid division by zero or log of zero issues for dL_Mpc
    # For z=0, the distance modulus should be effectively 0 or undefined depending
    # For practical fitting, we usually only fit for z > 0
    if isinstance(z, np.ndarray):
        mu = np.zeros_like(z, dtype=float)
        non_zero_z_indices = z > 0
        dL_Mpc_values = np.array([luminosity_distance(zi, H0, omega_m) for zi in z[
        mu[non_zero_z_indices] = 5 * np.log10(dL_Mpc_values) + 25
        # Set mu to a large value or handle for z <= 0 if they exist in data and ca
        # For typical supernova data, z will always be > 0.
        return mu
    else: # For a single z value
        if z <= 0:
            return 0 # Or handle as per your assignment's requirement for z=0
        dL_Mpc = luminosity_distance(z, H0, omega_m)
        mu = 5 * np.log10(dL_Mpc) + 25
        return mu
```

🔧 Fit the Model to Supernova Data

We now perform a non-linear least squares fit to the supernova data using our theoretical model for $\mu(z)$. This fitting procedure will estimate the best-fit values for the Hubble constant H_0 and matter density parameter Ω_m, along with their associated uncertainties.

We'll use:

- `curve_fit` from `scipy.optimize` for the fitting.
- The observed distance modulus (\mu), redshift (z), and measurement errors.

The initial guess is:

- $H_0 = 70\,\mathrm{km/s/Mpc}$
- $\Omega_m = 0.3$

In [9]:
```
try:
    data_sn = pd.read_csv('Pantheon+SH0ES.dat', delim_whitespace=True, comment='#')
    zHD = data_sn['zHD'].values
    MU_SH0ES = data_sn['MU_SH0ES'].values
    MU_SH0ES_ERR_DIAG = data_sn['MU_SH0ES_ERR_DIAG'].values
except FileNotFoundError:
    print("Error: 'Pantheon+SH0ES.dat' not found. Please ensure the data file is do
    # Exit or handle the error appropriately
```

```
C:\Users\Ansh\AppData\Local\Temp\ipykernel_3160\4078616039.py:2: FutureWarning: The
'delim_whitespace' keyword in pd.read_csv is deprecated and will be removed in a fut
ure version. Use ``sep='\s+'`` instead
  data_sn = pd.read_csv('Pantheon+SH0ES.dat', delim_whitespace=True, comment='#')
```

In [10]:
```
print(c)
```

```
Name   = Speed of light in vacuum
Value  = 299792458.0
Uncertainty  = 0.0
Unit  = m / s
Reference = CODATA 2018
```

In [11]:
```
# Initial guesses for H0 (km/s/Mpc) and Omega_m
# Common initial values are H0 around 70 and Omega_m around 0.3
initial_guesses = [70.0, 0.3]

try:
    # Perform the curve fit
    # p_opt will contain the optimized parameters [H0, Omega_m]
    # pcov will contain the covariance matrix of the parameters
    p_opt, pcov = curve_fit(
        f=cosmological_model,
        xdata=zHD,
        ydata=MU_SH0ES,
        p0=initial_guesses,
        sigma=MU_SH0ES_ERR_DIAG, # Use uncertainties for weighted fitting
        absolute_sigma=True     # Treat sigma as absolute errors
    )

    # Extract the fitted parameters
    H0_fit = p_opt[0]
    Omega_m_fit = p_opt[1]

    # Calculate the uncertainties (standard deviations) from the covariance matrix
    perr = np.sqrt(np.diag(pcov))
    H0_err = perr[0]
    Omega_m_err = perr[1]

    print(f"Fitted H₀: {H0_fit:.2f} ± {H0_err:.2f} km/s/Mpc")
    print(f"Fitted Ωm: {Omega_m_fit:.3f} ± {Omega_m_err:.3f}")

except RuntimeError as e:
    print(f"Error during curve fitting: {e}")
    print("This might be due to issues with initial guesses, data, or model definit
```

```
Fitted H₀: 72.97 ± 0.26 km/s/Mpc
Fitted Ωm: 0.351 ± 0.019
```

In [12]:
```python
import sys
print(sys.executable)
print(sys.path)
```

```
c:\Users\Ansh\AppData\Local\Programs\Python\Python312\python.exe
['c:\\Users\\Ansh\\AppData\\Local\\Programs\\Python\\Python312\\python312.zip',
'c:\\Users\\Ansh\\AppData\\Local\\Programs\\Python\\Python312\\DLLs', 'c:\\Users\\An
sh\\AppData\\Local\\Programs\\Python\\Python312\\Lib', 'c:\\Users\\Ansh\\AppData\\Lo
cal\\Programs\\Python\\Python312', '', 'C:\\Users\\Ansh\\AppData\\Roaming\\Python\\P
ython312\\site-packages', 'C:\\Users\\Ansh\\AppData\\Roaming\\Python\\Python312\\sit
e-packages\\win32', 'C:\\Users\\Ansh\\AppData\\Roaming\\Python\\Python312\\site-pack
ages\\win32\\lib', 'C:\\Users\\Ansh\\AppData\\Roaming\\Python\\Python312\\site-packa
ges\\Pythonwin', 'c:\\Users\\Ansh\\AppData\\Local\\Programs\\Python\\Python312\\Lib
\\site-packages']
```

⏳ Estimate the Age of the Universe

Now that we have the best-fit values of H_0 and Ω_m, we can estimate the age of the universe. This is done by integrating the inverse of the Hubble parameter over redshift:

$$t_0 = \int_0^\infty \frac{1}{(1+z)H(z)}\,dz$$

We convert H_0 to SI units and express the result in gigayears (Gyr). This provides an independent check on our cosmological model by comparing the estimated age to values from other probes like Planck CMB measurements.

In [13]:
```python
# Function to calculate the age of the universe
def age_of_universe(H0, Omega_m, Omega_lambda):
    # Convert H0 to inverse time units (s^-1)
    H0_s = H0 * 1000 * u.m / u.s / (1e6 * u.parsec).to(u.m) # km/s/Mpc to s^-1
    H0_s_value = H0_s.value

    # Integral for the age of the universe
    def integral_age(a, Om, Ol):
        return 1 / (a * np.sqrt(Om * a**(-3) + Ol))

    # Perform the integral from a=0 to a=1 (z=infinity to z=0)
    Omega_lambda = 1 - Omega_m # Assuming flat universe for this calculation
    t_H, _ = quad(integral_age, 0, 1, args=(Omega_m, Omega_lambda))

    age_seconds = t_H / H0_s_value
    age_gyr = (age_seconds * u.s).to(u.Gyr).value
    return age_gyr

# Assume Omega_m = 0.3 for age calculation (as per handout question 3)
Omega_m_fixed_for_age = 0.3
Omega_lambda_fixed_for_age = 1 - Omega_m_fixed_for_age

age_full_data_fixed_omega_m = age_of_universe(H0_fit, Omega_m_fixed_for_age, Omega_
```

```
print(f"Estimated age of the Universe (with H0 from full dataset, Omega_m=0.3): {ag

# Example for a different Omega_m value (e.g., 0.2)
Omega_m_alternative = 0.2
Omega_lambda_alternative = 1 - Omega_m_alternative
age_alternative_omega_m = age_of_universe(H0_fit, Omega_m_alternative, Omega_lambda
print(f"Estimated age of the Universe (with H0 from full dataset, Omega_m=0.2): {ag
```

```
Estimated age of the Universe (with H0 from full dataset, Omega_m=0.3): 12.92 Gyr
Estimated age of the Universe (with H0 from full dataset, Omega_m=0.2): 14.42 Gyr
```

📊 Analyze Residuals

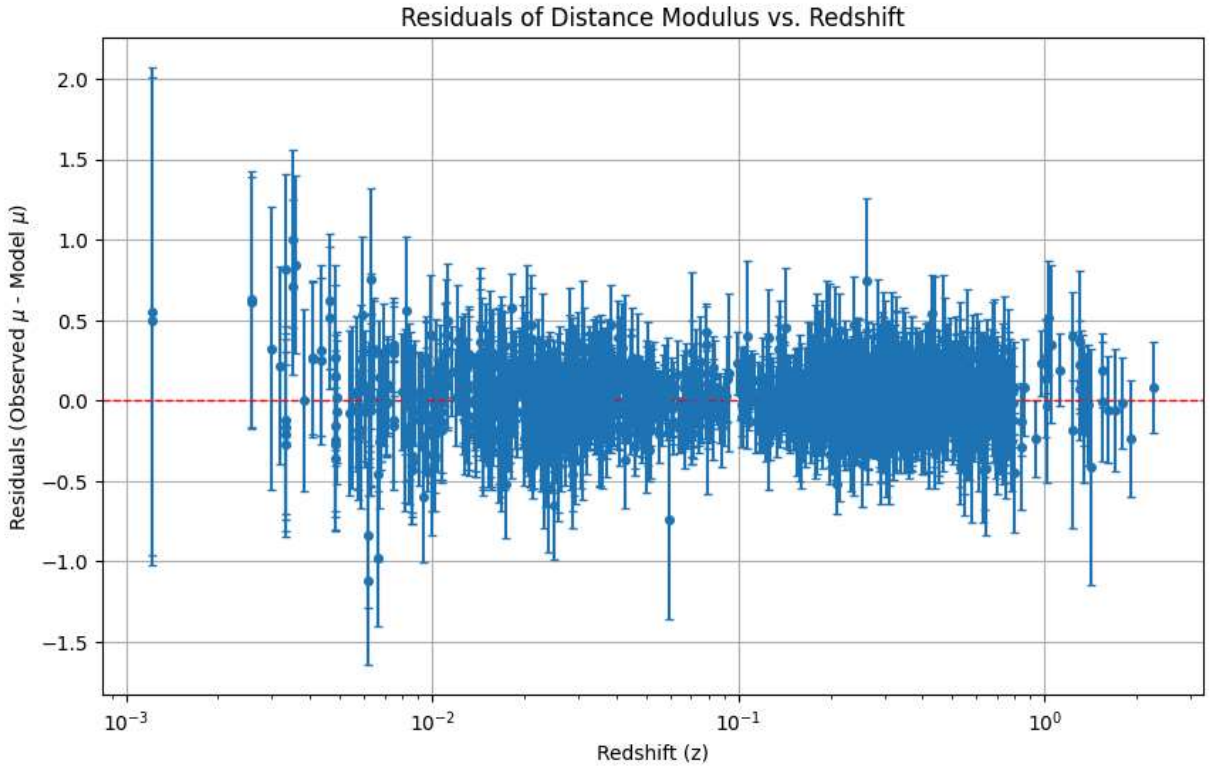
To evaluate how well our cosmological model fits the data, we compute the residuals:

$$\text{Residual} = \mu_{\text{obs}} - \mu_{\text{model}}$$

Plotting these residuals against redshift helps identify any systematic trends, biases, or outliers. A good model fit should show residuals scattered randomly around zero without any significant structure.

In [14]:
```
# Calculate residuals
mu_predicted = distance_modulus(zHD, H0_fit, Omega_m_fit)
residuals = mu_obs - mu_predicted

# Plotting the residuals (Plot 3)
plt.figure(figsize=(10, 6))
plt.errorbar(zHD, residuals, yerr=mu_err, fmt='o', markersize=4, capsize=2)
plt.axhline(0, color='red', linestyle='--', linewidth=1)
plt.xlabel('Redshift (z)')
plt.ylabel('Residuals (Observed $\\mu$ - Model $\\mu$)')
plt.title('Residuals of Distance Modulus vs. Redshift')
plt.xscale('log')
plt.grid(True)
plt.show()
```



🔧 Fit with Fixed Matter Density

To reduce parameter degeneracy, let's fix $\Omega_m = 0.3$ and fit only for the Hubble constant H_0.

In [15]:
```
def distance_modulus_fixed_omega_m(z, H0):
    return distance_modulus(z, H0, 0.3)

# Try fitting with this fixed value
params_fixed_omega_m, covariance_fixed_omega_m = curve_fit(distance_modulus_fixed_o

H0_fixed_omega_m = params_fixed_omega_m[0]
H0_err_fixed_omega_m = np.sqrt(np.diag(covariance_fixed_omega_m))[0]

print(f"\nBest-fit H0 (with fixed Omega_m=0.3): {H0_fixed_omega_m:.2f} ± {H0_err_fi
```

```
Best-fit H0 (with fixed Omega_m=0.3): 73.53 ± 0.17 km/s/Mpc
```

🔍 Compare Low-z and High-z Subsamples

Finally, we examine whether the inferred value of H_0 changes with redshift by splitting the dataset into:

- **Low-z** supernovae ($z < 0.1$)
- **High-z** supernovae ($z \geq 0.1$)//

We then fit each subset separately (keeping $\Omega_m = 0.3$) to explore any potential tension or trend with redshift.

In [16]:
```
z_split = 0.1

# Split the data into low-z and high-z subsamples
low_z_data = data[data['zHD'] < z_split]
high_z_data = data[data['zHD'] >= z_split]

zHD_low_z = low_z_data['zHD'].values
mu_obs_low_z = low_z_data['MU_SH0ES'].values
mu_err_low_z = low_z_data['MU_SH0ES_ERR_DIAG'].values

zHD_high_z = high_z_data['zHD'].values
mu_obs_high_z = high_z_data['MU_SH0ES'].values
mu_err_high_z = high_z_data['MU_SH0ES_ERR_DIAG'].values

# Fit low-z subsample (keeping Omega_m = 0.3)
# H0_low will store the array of parameters from curve_fit (just H0 in this case)
H0_low, covariance_low = curve_fit(distance_modulus_fixed_omega_m, zHD_low_z, mu_ob

# Fit high-z subsample (keeping Omega_m = 0.3)
# H0_high will store the array of parameters from curve_fit (just H0 in this case)
H0_high, covariance_high = curve_fit(distance_modulus_fixed_omega_m, zHD_high_z, mu

print(f"Low-z (z < {z_split}): H₀ = {H0_low[0]:.2f} km/s/Mpc")
print(f"High-z (z ≥ {z_split}): H₀ = {H0_high[0]:.2f} km/s/Mpc")
```

```
Low-z (z < 0.1): H₀ = 73.01 km/s/Mpc
High-z (z ≥ 0.1): H₀ = 73.85 km/s/Mpc
```

You can check your results and potential reasons for different values from accepted constant using this paper by authors of the Pantheon+ dataset

You can find more about the dataset in the paper too