Name: Kritish Nagyal, B.Tech Electronics and Communication (Avionics) Engineering

Assignment: Measuring Cosmological Parameters Using Type Ia Supernovae

In this assignment, we'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. We 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

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

In [5]:
```
#Load necessary libraries.
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.optimize import curve_fit
from astropy import units as u
from astropy.constants import c
from scipy.integrate import quad
```

In [6]:
```
#load the data set.
a = pd.read_csv('Pantheon+SH0ES.dat', delim_whitespace=True)
a
```

Out[6]:

	CID	IDSURVEY	zHD	zHDERR	zCMB	zCMBERR	zHEL	zHELERR	m_b_corr	m_b_corr_err_DIAG	...	Pl
0	2011fe	51	0.00122	0.00084	0.00122	0.00002	0.00082	0.00002	9.74571	1.516210	...	
1	2011fe	56	0.00122	0.00084	0.00122	0.00002	0.00082	0.00002	9.80286	1.517230	...	
2	2012cg	51	0.00256	0.00084	0.00256	0.00002	0.00144	0.00002	11.47030	0.781906	...	
3	2012cg	56	0.00256	0.00084	0.00256	0.00002	0.00144	0.00002	11.49190	0.798612	...	
4	1994DRichmond	50	0.00299	0.00084	0.00299	0.00004	0.00187	0.00004	11.52270	0.880798	...	
...	...	...	...	...	...	...	...	...	...	...	...	
1696	rutledge	106	1.61505	0.00545	1.61499	0.00500	1.61399	0.00500	25.90650	0.331927	...	
1697	geta	106	1.69706	0.04006	1.69702	0.04000	1.70000	0.04000	26.03330	0.379521	...	
1698	stone	106	1.80119	0.02014	1.80111	0.02000	1.80000	0.02000	26.23350	0.280685	...	
1699	wilson	106	1.91165	0.00263	1.91160	0.00100	1.91401	0.00100	26.17030	0.357624	...	
1700	colfax	106	2.26137	0.02018	2.26130	0.02000	2.26000	0.02000	26.92980	0.280011	...	

1701 rows × 47 columns

In [9]:
```
#View the top entries in the data set.
print(a.head())
```

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

   zHELERR  m_b_corr  m_b_corr_err_DIAG  ...  PKMJDERR  NDOF   FITCHI2  \
0  0.00002   9.74571           1.516210  ...    0.1071    36   26.8859
1  0.00002   9.80286           1.517230  ...    0.0579   101   88.3064
2  0.00002  11.47030           0.781906  ...    0.0278   165  233.5000
3  0.00002  11.49190           0.798612  ...    0.0667    55  100.1220
4  0.00004  11.52270           0.880798  ...    0.0522   146  109.8390

    FITPROB  m_b_corr_err_RAW  m_b_corr_err_VPEC  biasCor_m_b  biasCorErr_m_b  \
0  0.864470            0.0991             1.4960       0.0381           0.005
1  0.812220            0.0971             1.4960      -0.0252           0.003
2  0.000358            0.0399             0.7134       0.0545           0.019
3  0.000193            0.0931             0.7134       0.0622           0.028
4  0.988740            0.0567             0.6110       0.0650           0.009

   biasCor_m_b_COVSCALE  biasCor_m_b_COVADD
0                   1.0               0.003
1                   1.0               0.004
2                   1.0               0.036
3                   1.0               0.040
4                   1.0               0.006

[5 rows x 47 columns]
```

In [11]:
```python
#checking for any null values in our data set
a.isnull()
```

Out[11]:

	CID	IDSURVEY	zHD	zHDERR	zCMB	zCMBERR	zHEL	zHELERR	m_b_corr	m_b_corr_err_DIAG	...	PKMJDERR	NDO
0	False	False	False	False	False	False	False	False	False	False	...	False	Fals
1	False	False	False	False	False	False	False	False	False	False	...	False	Fals
2	False	False	False	False	False	False	False	False	False	False	...	False	Fals
3	False	False	False	False	False	False	False	False	False	False	...	False	Fals
4	False	False	False	False	False	False	False	False	False	False	...	False	Fals
...	...	...	...	...	...	...	...	...	...	...	...	...	.
1696	False	False	False	False	False	False	False	False	False	False	...	False	Fals
1697	False	False	False	False	False	False	False	False	False	False	...	False	Fals
1698	False	False	False	False	False	False	False	False	False	False	...	False	Fals
1699	False	False	False	False	False	False	False	False	False	False	...	False	Fals
1700	False	False	False	False	False	False	False	False	False	False	...	False	Fals

1701 rows × 47 columns

In [13]:
```python
#Counting the no. of null values in our data set
a.isnull().sum()
```

Out[13]:
```
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
```

Since we have no null values in our data set so no need to drop any row.

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 [17]:
```
columns = ['zHD', 'MU_SH0ES', 'MU_SH0ES_ERR_DIAG']
a_clean = a[columns].dropna()
a_clean #New data set with on three parameers - zHD, MU_SH0ES, MU_SH0ES_ERR_DIAG
```

Out[17]:

	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
...	...	...	...
1696	1.61505	45.1595	0.333024
1697	1.69706	45.2863	0.380480
1698	1.80119	45.4865	0.281981
1699	1.91165	45.4233	0.358642
1700	2.26137	46.1828	0.281309

1701 rows × 3 columns

In [19]:

```
#Creating individual numpy arrarys for each of these 3 parameters.
HD = a_clean['zHD'].to_numpy()
MU = a_clean['MU_SH0ES'].to_numpy()
MU_err = a_clean['MU_SH0ES_ERR_DIAG'].to_numpy()
```

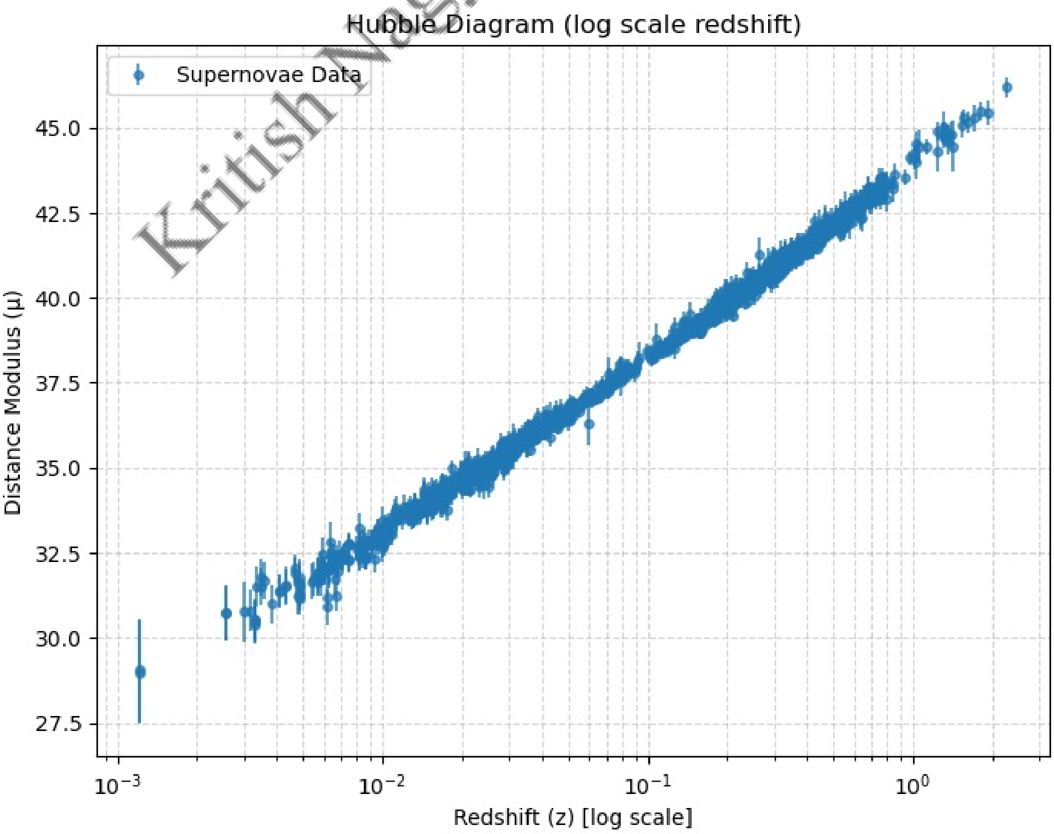
Hubble Diagram

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 [42]:

```
plt.figure(figsize=(8, 6))
plt.errorbar(HD, MU, yerr=MU_err, fmt='o', markersize=4, label='Supernovae Data', alpha=0.7)

plt.xscale('log')  # x-axis (redshift) to logarithmic scale
plt.xlabel("Redshift (z) [log scale]")
plt.ylabel("Distance Modulus (μ)")
plt.title("Hubble Diagram (log scale redshift)")
plt.grid(True, which='both', linestyle='--', alpha=0.5)
plt.legend()
plt.show()
```



Define the Cosmological Model

We now define the theoretical framework based on the flat ΛCDM 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 :

  $d_L(z) = (1 + z) \cdot cH_0 \int_{z0} dz' E(z')$

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 [45]:

```
#E(z) for a flat ΛCDM universe
def E(z, Omega_m):
    return np.sqrt(Omega_m * (1 + z)**3 + (1 - Omega_m))

#Luminosity distance in Mpc
def luminosity_distance(z, H0, Omega_m):
    H0 = H0 * u.km / u.s / u.Mpc  # Convert H0 to proper units
    integrand = lambda z_prime: 1.0 / np.sqrt(Omega_m * (1 + z_prime)**3 + (1 - Omega_m))
    integral, _ = quad(integrand, 0, z)
    d_L = (c / H0) * (1 + z) * integral
    return d_L.to(u.Mpc)

# 3. Theoretical distance modulus
def mu_theory(z_array, H0, Omega_m):
    return np.array([
        5 * np.log10(luminosity_distance(z, H0, Omega_m).value) + 25
        for z in z_array
    ])
```

Fit the Model to Supernova Data

We now perform a non-linear least squares fit to the supernova data using our theoretical model for μ(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:

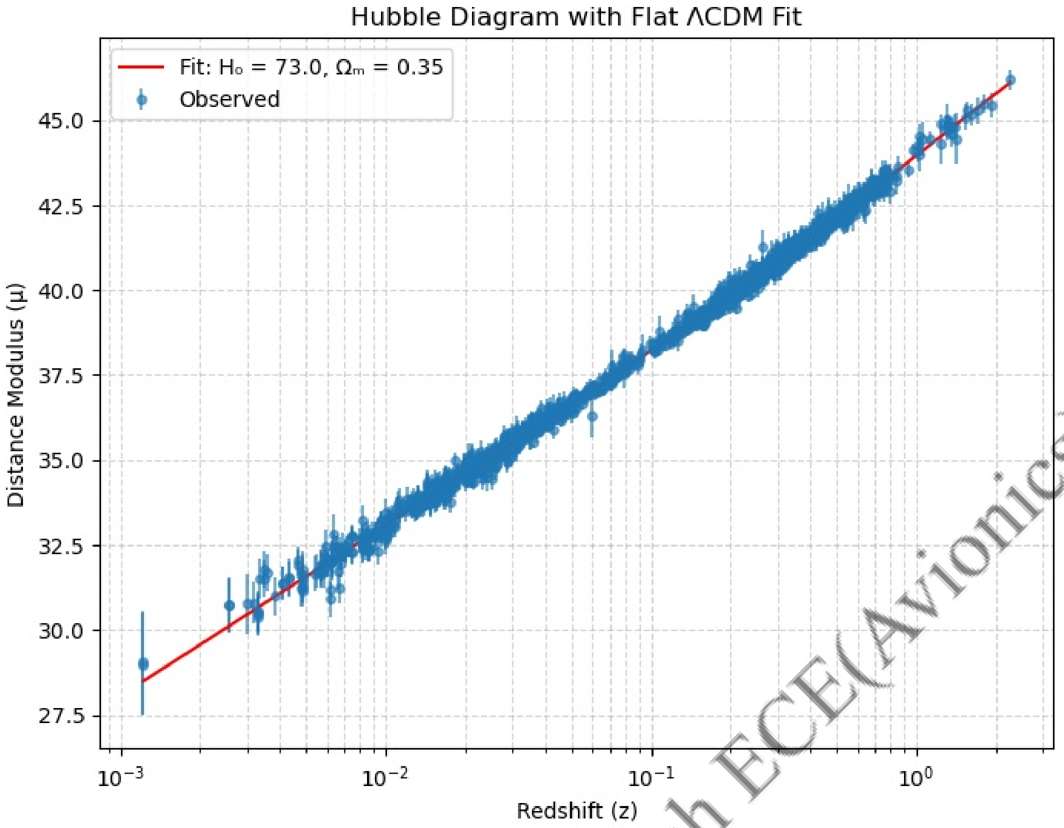
- H_0 = 70km/s/Mpc
- Ω_m = 0.3

In [48]:

```
# Initial guess: H0 = 70, Omega_m = 0.3
# Non-linear least squares fit
initial = [70, 0.3]

popt, pcov = curve_fit(
    mu_theory, HD, MU,
    sigma=MU_err, absolute_sigma=True,
    p0=[70, 0.3]
)

H0_fit, Omega_m_fit = popt
H0_err, Omega_m_err = np.sqrt(np.diag(pcov))

print(f"Best-fit H0: {H0_fit:.2f} ± {H0_err:.2f} km/s/Mpc")
print(f"Best-fit Omega_m: {Omega_m_fit:.3f} ± {Omega_m_err:.3f}")

# Step 3: Plot
z_plot = np.logspace(np.log10(np.min(HD[HD > 0])), np.log10(np.max(HD)), 100)
mu_model = mu_theory(z_plot, H0_fit, Omega_m_fit)

plt.figure(figsize=(8, 6))
plt.errorbar(HD, MU, yerr=MU_err, fmt='o', label='Observed', alpha=0.6, markersize=4)
plt.plot(z_plot, mu_model, 'r-', label=f'Fit: H₀ = {H0_fit:.1f}, Ωₘ = {Omega_m_fit:.2f}')
plt.xscale('log')
plt.xlabel('Redshift (z)')
```

```python
plt.ylabel('Distance Modulus (μ)')
plt.title('Hubble Diagram with Flat ΛCDM Fit')
plt.grid(True, which='both', linestyle='--', alpha=0.5)
plt.legend()
plt.show()
```

```
Best-fit H0: 72.97 ± 0.26 km/s/Mpc
Best-fit Omega_m: 0.351 ± 0.019
```



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:

$t_0 = \int_{\infty 0}^{1(1+z)H(z)} dz$

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 [50]:
```python
# Best-fit parameters
H0_fit = 70.0       #Hubble constant in km/s/Mpc
Omega_m_fit = 0.3  #Matter density
H0_si = (H0_fit * u.km / u.s / u.Mpc).to(1 / u.s) #Convert H0 to 1/s

# Define the integrand function for the age of the universe
def age_integrand(z, Omega_m):
    return 1.0 / ((1 + z) * np.sqrt(Omega_m * (1 + z)**3 + (1 - Omega_m)))

integral, _ = quad(age_integrand, 0, 1e5, args=(Omega_m_fit)) #Perform numerical integration from z = 0 to z = .
t0_seconds = integral / H0_si.value  #age of universe in seconds
t0_gyr = (t0_seconds * u.s).to(u.Gyr) #age of universe in gyr

print(f"Estimated age of the universe: {t0_gyr:.2f}")
```

```
Estimated age of the universe: 13.47 Gyr
```

Analyze Residuals

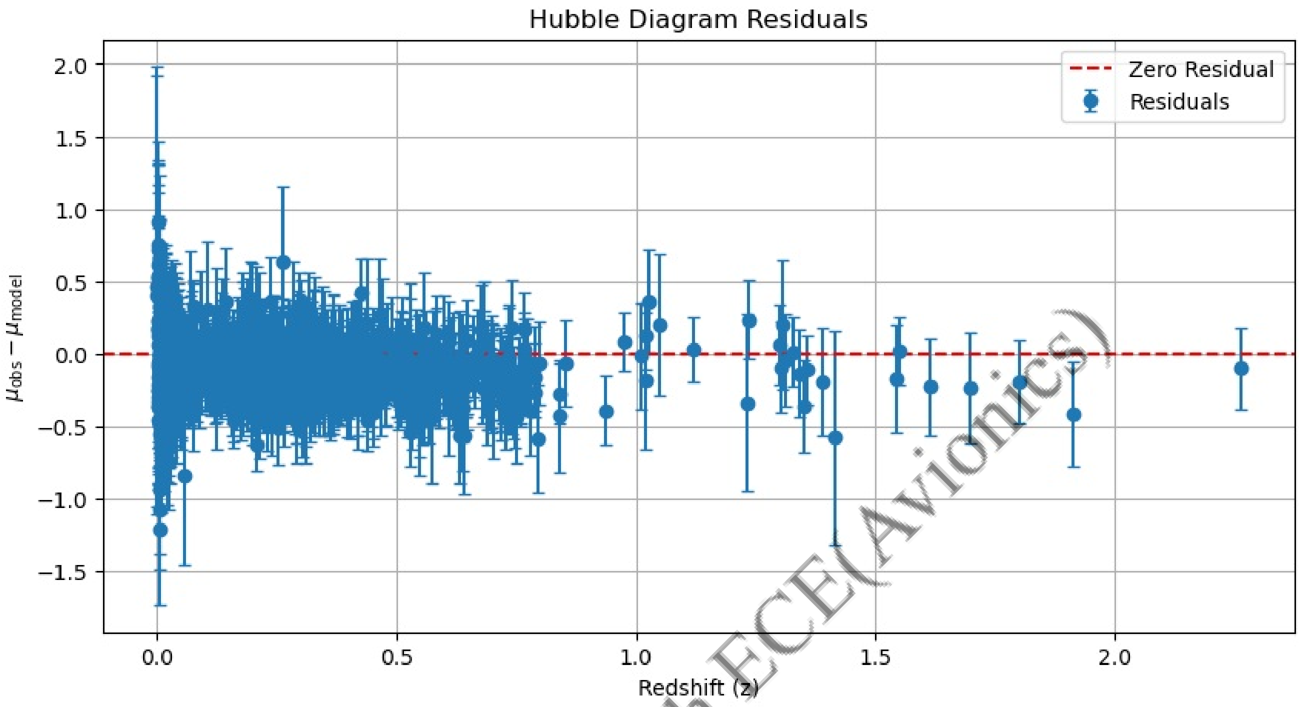
To evaluate how well our cosmological model fits the data, we compute the residuals:

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

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 [52]:
```python
#H0_fit and Omega_m_fit have already ben computed using curve_fit
mu_model = mu_theory(HD, H0_fit, Omega_m_fit) #Compute model predictions
```

```
residuals = MU - mu_model #Compute residuals
# Plotting residuals vs. redshift
plt.figure(figsize=(10, 5))
plt.errorbar(HD, residuals, yerr=MU_err, fmt='o', capsize=3, label='Residuals')
plt.axhline(0, color='red', linestyle='--', label='Zero Residual')
plt.xlabel('Redshift (z)')
plt.ylabel(r'$\mu_{\mathrm{obs}} - \mu_{\mathrm{model}}$')
plt.title('Hubble Diagram Residuals')
plt.grid(True)
plt.legend()
plt.show()
```



Fit with Fixed Matter Density

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

In [54]:
```
#Define the new model with fixed Omega_m = 0.3
def mu_fixed_Om(z, H0):
    return mu_theory(z, H0, Omega_m=0.3)

H0_initial_guess = [70] #curve fitting for H0 only
H0_fit_only, H0_cov = curve_fit(
    mu_fixed_Om, HD, MU,
    sigma=MU_err, absolute_sigma=True,
    p0=H0_initial_guess
)
#extracting fitted value and uncertainty
H0_best = H0_fit_only[0]
H0_err = np.sqrt(np.diag(H0_cov))[0]
print(f"Best-fit H0 (Omega_m = 0.3 fixed): {H0_best:.2f} ± {H0_err:.2f} km/s/Mpc")
```

```
Best-fit H0 (Omega_m = 0.3 fixed): 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 ≥ 0.1)

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

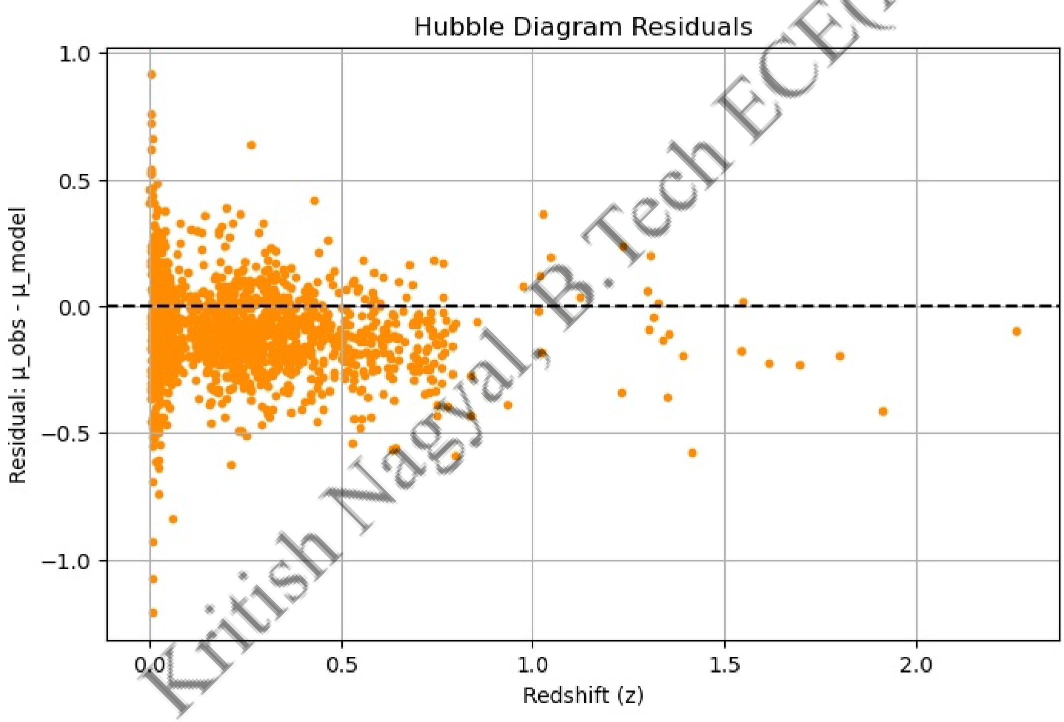
In [56]:
```
# Split point
z_split = 0.1
#Low-z subset
HD_low = HD[HD < z_split]
MU_low = MU[HD < z_split]
MU_err_low = MU_err[HD < z_split]
#High-z subset
HD_high = HD[HD >= z_split]
MU_high = MU[HD >= z_split]
```

```python
MU_err_high = MU_err[HD >= z_split]
#Fit for low-z
H0_low, H0_low_cov = curve_fit(
    mu_fixed_Om, HD_low, MU_low,
    sigma=MU_err_low, absolute_sigma=True,
    p0=[70]
)
#Fit for high-z
H0_high, H0_high_cov = curve_fit(
    mu_fixed_Om, HD_high, MU_high,
    sigma=MU_err_high, absolute_sigma=True,
    p0=[70]
)
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
```

In [59]:

```python
import numpy as np
import matplotlib.pyplot as plt

# Example: z, mu_obs (observed), mu_model (model predictions)
residuals = MU - mu_model

plt.figure(figsize=(8, 5))
plt.scatter(HD, residuals, color='darkorange', s=10)
plt.axhline(0, color='black', linestyle='--')
plt.xlabel("Redshift (z)")
plt.ylabel("Residual: µ_obs - µ_model")
plt.title("Hubble Diagram Residuals")
plt.grid(True)
plt.show()
```



In [ ]: