🔭 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
```

```
# 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
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
```

📂 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:

```
# Local file path
import pandas as pd

data = pd.read_csv('Pantheon+SH0ES.dat', delim_whitespace=True, comment='#')

data.head()
```

```
# See structure
```

```
C:\Users\Asus\AppData\Local\Temp\ipykernel_19080\1921556846.py:4: FutureWarning: The 'delim_whitespace' keyword in pd.read_csv is de
  data = pd.read_csv('Pantheon+SH0ES.dat', delim_whitespace=True, comment='#')
```

	CID	IDSURVEY	zHD	zHDERR	zCMB	zCMBERR	zHEL	zHELERR	m_b_corr	m_b_corr_err_DIAG	...	PKMJDERR	NDOF	FI
0	2011fe	51	0.00122	0.00084	0.00122	0.00002	0.00082	0.00002	9.74571	1.516210	...	0.1071	36	2
1	2011fe	56	0.00122	0.00084	0.00122	0.00002	0.00082	0.00002	9.80286	1.517230	...	0.0579	101	8
2	2012cg	51	0.00256	0.00084	0.00256	0.00002	0.00144	0.00002	11.47030	0.781906	...	0.0278	165	23
3	2012cg	56	0.00256	0.00084	0.00256	0.00002	0.00144	0.00002	11.49190	0.798612	...	0.0667	55	10
4	1994DRichmond	50	0.00299	0.00084	0.00299	0.00004	0.00187	0.00004	11.52270	0.880798	...	0.0522	146	10

5 rows × 47 columns

🗒️ 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.

```
Start coding or generate with AI.
```

🖌️ 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.

```
# Filter for entries with usable data based on the required columns
z = data['zHD']                    # Redshift
mu = data['MU_SH0ES']              # Distance modulus
mu_err = data['MU_SH0ES_ERR_DIAG'] # Optional: uncertainties
```

📈 Plot the Hubble Diagram

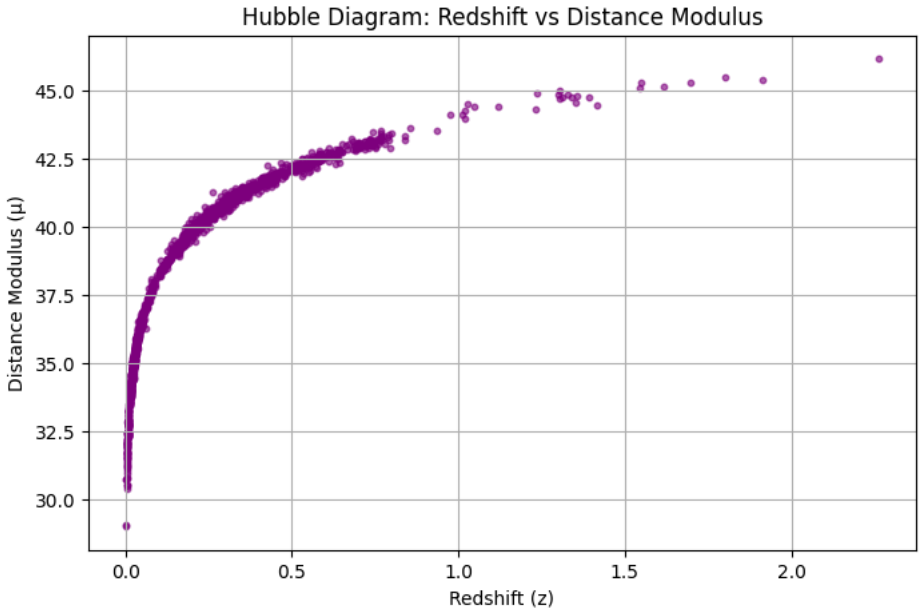
Let's visualize the relationship between redshift z and distance modulus μ, known as the Hubble diagram. This plot is a cornerstone of observational cosmology—it allows us to compare supernova observations with theoretical predictions based on different cosmological models.

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

```
# Write a code to plot the distance modulus and the redshift (x-axis), label them accordingly.
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))
plt.scatter(z, mu, s=10, color='purple', alpha=0.6)
plt.xlabel('Redshift (z)')
plt.ylabel('Distance Modulus (μ)')
plt.title('Hubble Diagram: Redshift vs Distance Modulus')
plt.grid(True)
plt.show()

#Try using log scale in x-axis
```



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.

```
# Define the E(z) for flat LCDM
def E(z, Omega_m):
    return np.sqrt(Omega_m * (1 + z)**3 + (1 - Omega_m))


# Luminosity distance in Mpc, try using scipy quad to integrate.
from scipy.integrate import quad
def luminosity_distance(z, H0, Omega_m):
    c = 3e5  # km/s
    integral = np.array([quad(lambda z_prime: 1.0 / E(z_prime, Omega_m), 0, zi)[0] for zi in z])
    dL = (1 + z) * c / H0 * integral
    return dL


# Theoretical distance modulus, use above function inside mu_theory to compute luminosity distance
def mu_theory(z, H0, Omega_m):
     dL = luminosity_distance(z, H0, Omega_m)
     return 5 * np.log10(dL) + 25
```

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:

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

```
# Initial guess: H0 = 70, Omega_m = 0.3
from scipy.optimize import curve_fit
import numpy as np
p0 = [70, 0.3]

# Write a code for fitting and taking error out of the parameters
z_fit = z[z>0]
mu_fit = mu[ z>0]

popt, pcov = curve_fit(mu_theory, z_fit, mu_fit, p0=p0)

H0_fit, Omega_m_fit = popt
H0_err, Omega_m_err = np.sqrt(np.diag(pcov))  # 1-sigma errors

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

```
Fitted H0 = 72.66 ± 0.20 km/s/Mpc
Fitted Omega_m = 0.357 ± 0.014
```

⏳ 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.

```
# Write the function for age of the universe as above
from scipy.integrate import quad

def age_of_universe(H0, Omega_m):
    # Define H(z) based on flat Lambda-CDM
    def E_inv(z):
        return 1.0 / ((1 + z) * np.sqrt(Omega_m * (1 + z)**3 + (1 - Omega_m)))

    # Integrate from z = 0 to z = 1000 (approx infinity)
    integral, _ = quad(E_inv, 0, 1000)

    # Convert H0 to 1/sec
    H0_SI = H0 * 1000 / (3.086e22)  # km/s/Mpc → 1/s

    # Age in seconds → convert to Gyr
    age_sec = integral / H0_SI
    age_years = age_sec / (60 * 60 * 24 * 365.25)
    age_gyr = age_years / 1e9

    return age_gyr


t0 = age_of_universe(H0_fit, Omega_m_fit)
print(f"Estimated age of Universe: {t0:.2f} Gyr")
```

```
Estimated age of Universe: 12.35 Gyr
```

📊 Analyze Residuals

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

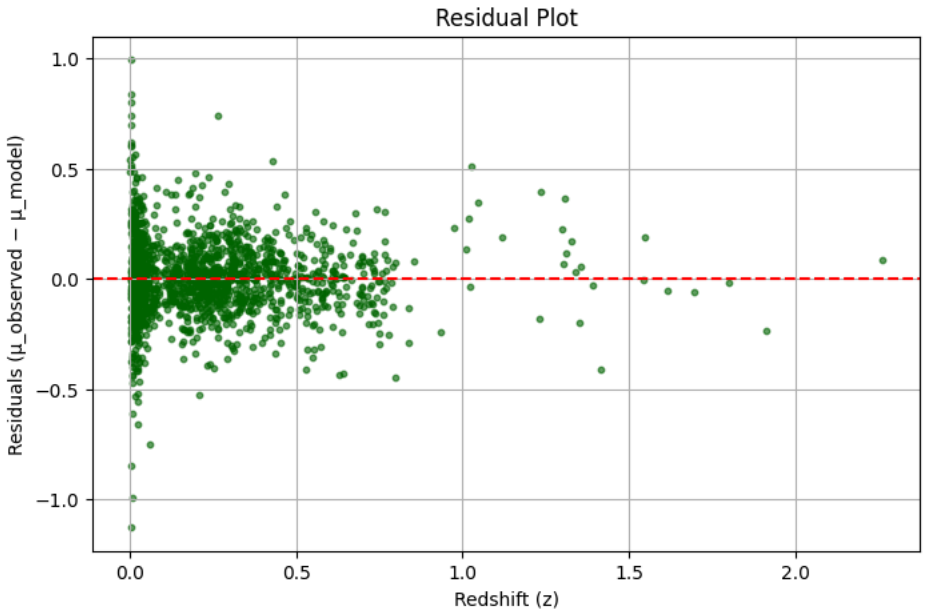
$$\text{Residual} = \mu_{\text{obs}} - \mu_{\text{model}}$$

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.

```
# Write the code to find residual by computing mu_theory and then plot
# Get predicted μ values from your model
mu_pred = mu_theory(z_fit, H0_fit, Omega_m_fit)
```

```
# Calculate residuals
residuals = mu_fit - mu_pred

# Plot the residuals
plt.figure(figsize=(8, 5))
plt.scatter(z_fit, residuals, s=10, color='darkgreen', alpha=0.6)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Redshift (z)')
plt.ylabel('Residuals (μ_observed − μ_model)')
plt.title('Residual Plot')
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.

```
def mu_fixed_Om(z, H0):
    return mu_theory(z, H0, Omega_m=0.3)

# Try fitting with this fixed value

from scipy.optimize import curve_fit

# Use z > 0 only
z_nonzero = z[z > 0]
mu_nonzero = mu[z > 0]

# Fit only for H0 (Omega_m fixed at 0.3)
H0_fixed, pcov = curve_fit(mu_fixed_Om, z_nonzero, mu_nonzero, p0=[70])
H0_fit_val = H0_fixed[0]
H0_err_val = np.sqrt(np.diag(pcov))[0]

print(f"Fitted H₀ with fixed Ωₘ = 0.3: {H0_fit_val:.2f} ± {H0_err_val:.2f} km/s/Mpc")
```

```
Fitted H₀ with fixed Ωₘ = 0.3: 73.28 ± 0.14 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.

```
# Split the data for the three columns and do the fitting again and see
```

```
from scipy.optimize import curve_fit

# Define redshift split value
z_split = 0.1

# LOW-Z data (z < 0.1)
z_low = z[z < z_split]
mu_low = mu[z < z_split]
H0_low, _ = curve_fit(mu_fixed_Om, z_low, mu_low, p0=[70])

# HIGH-Z data (z >= 0.1)
z_high = z[z >= z_split]
mu_high = mu[z >= z_split]
H0_high, _ = curve_fit(mu_fixed_Om, z_high, mu_high, 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 = 72.80 km/s/Mpc
High-z (z ≥ 0.1): H₀ = 73.65 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