proj

May 5, 2023

1 Constrain Cosmological Parameters using Supernovae Data

This notebook uses supernovae data, specifically, the data of the distance modulus vs. redshift to fit equations and constrain cosmological constants.

Here is an overview of our methods: 1) Fetch supernova data

2) Visualize data and get the data where redshift is small

3) Fit data with small redshift to the following equation and use the fitting results to calculate Hubble Constant H_0 and cosmological constant q_0.

$m - M = 43.17 - 5log_{10}(\frac{H_0}{70}) + 5log_{10}z + 1.086(1 - q_0)z$ —— (Eq. 1)

To better understand our data, we also try fitting our data backwards in the following steps, i.e., how well different q_0 fits our data.

4) Visualize the difference between our data fitted model and some hypothesized models.

5) Try different cosmological constants on our data and calculate the loss for each.

6) Use Cosmolopy package to more accurately calculate luminosity distance and thus the distance modulus.

[155]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import statsmodels.api as sm
```

1.1 1. Load Supernovae Data

[156]:
```
sn_data = pd.read_csv("./data.txt", sep="\t", header=0)
sn_data.head()
```

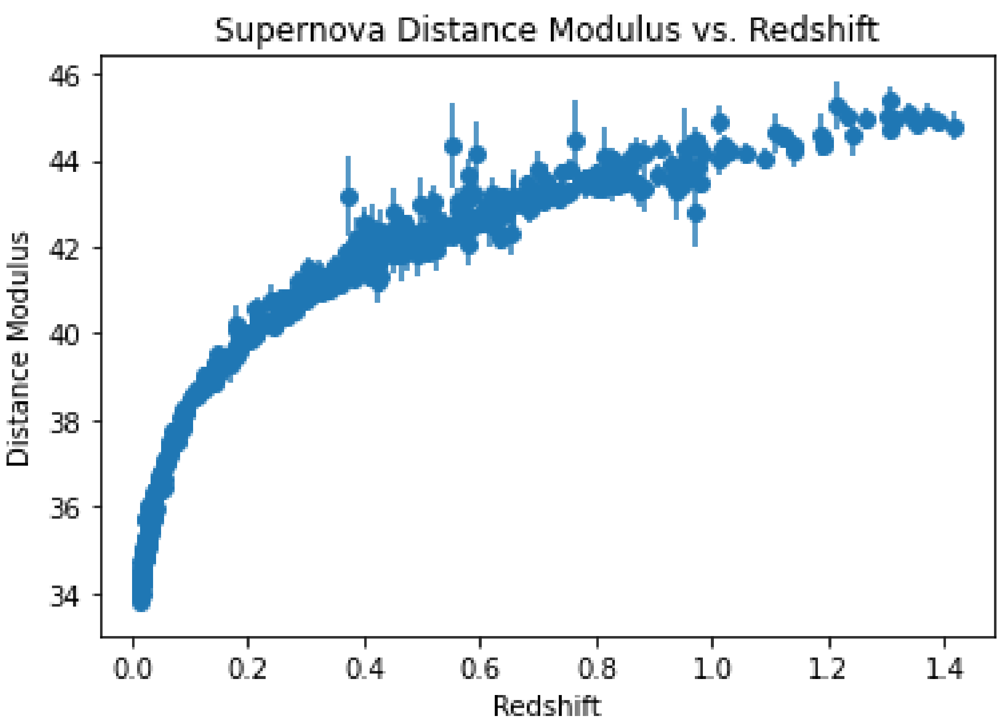
[156]:
```
  supernova name  redshift  distance modulus  distance modulus error  \
0         1993ah  0.028488         35.346583                0.223906
1         1993ag  0.050043         36.682368                0.166829
2          1993o  0.052926         36.817691                0.155756
3          1993b  0.070086         37.446737                0.158467
4         1992bs  0.062668         37.483409                0.156099

   probability
```

```
0      0.128419
1      0.128419
2      0.128419
3      0.128419
4      0.128419
```

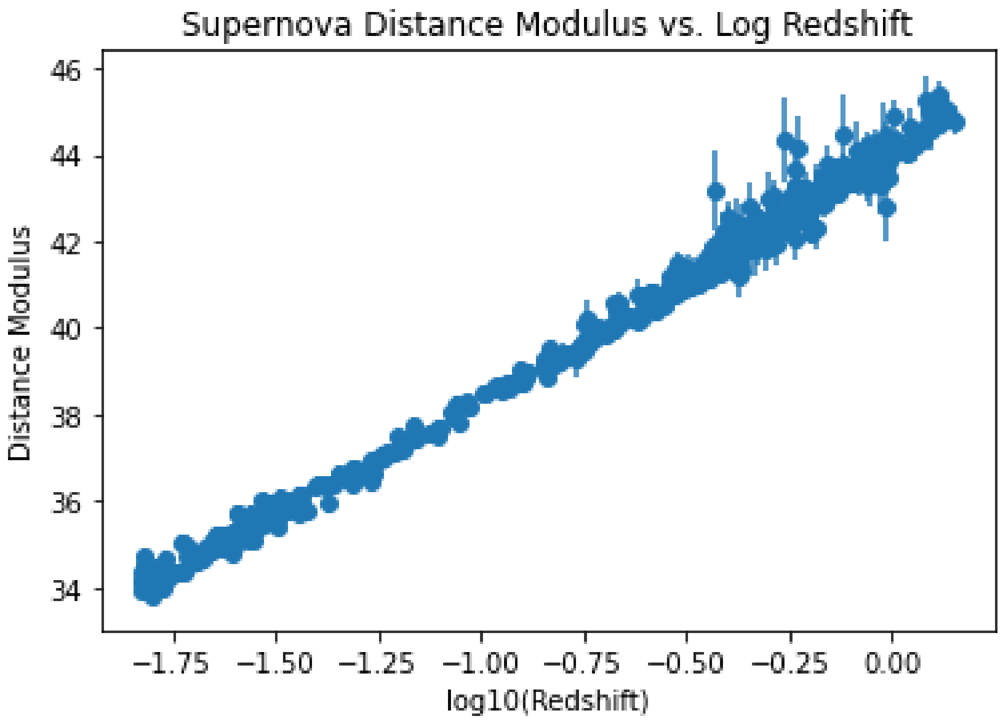
1.2 2. Visualize Data

We observe a log relationship between distance modulus and redshift.

[157]:
```
plt.errorbar(sn_data["redshift"], sn_data["distance modulus"], yerr =␣
↪sn_data["distance modulus error"], xerr=None, fmt='o')
plt.xlabel('Redshift')
plt.ylabel('Distance Modulus')
plt.title('Supernova Distance Modulus vs. Redshift')
plt.show()
```



[158]:
```
plt.errorbar(np.log10(sn_data["redshift"]), sn_data["distance modulus"], yerr =␣
↪sn_data["distance modulus error"], xerr=None, fmt='o')
plt.xlabel('log10(Redshift)')
plt.ylabel('Distance Modulus')
plt.title('Supernova Distance Modulus vs. Log Redshift')
plt.show()
```



Now we can extract two dataset: `small_z_data` and `medium_z_data`.

`small_z_data` contains the supernova data that has a very small redshift(z«1), and `medium_z_data` contains the data with relatively bigger redshift but still smaller than 0.2(z<0.2).

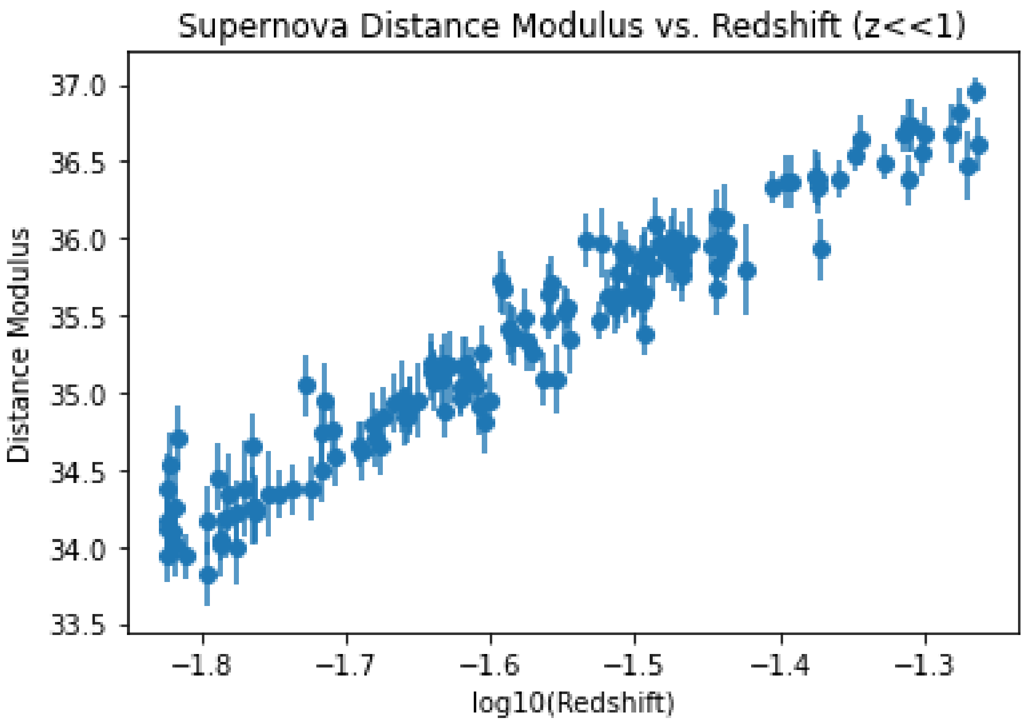
We will introduce the use of these two datasets later.

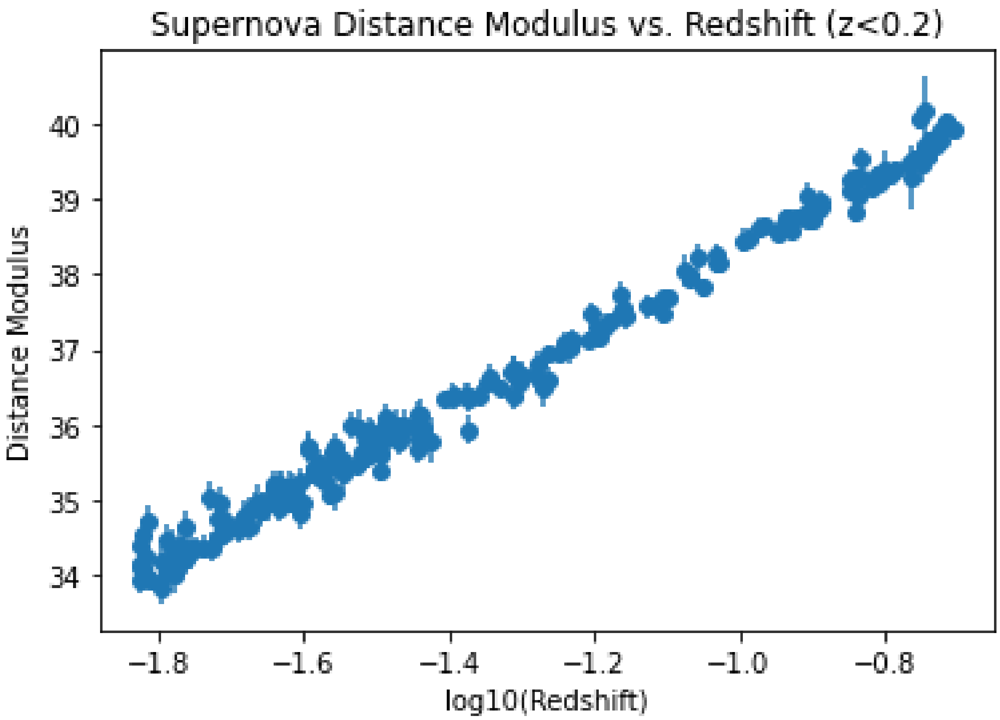
[159]:
```
small_z_data = sn_data[sn_data["redshift"] < 0.055]
medium_z_data = sn_data[sn_data["redshift"] < 0.2]
print("Number of points in small_z_data: ", len(small_z_data), "\nNumber of␣
↪points in medium_z_data: ", len(medium_z_data))
```

```
Number of points in small_z_data:  146
Number of points in medium_z_data:  230
```

[160]:
```
plt.errorbar(np.log10(small_z_data["redshift"]), small_z_data["distance␣
↪modulus"], yerr = small_z_data["distance modulus error"], xerr=None, fmt='o')
plt.xlabel('log10(Redshift)')
plt.ylabel('Distance Modulus')
plt.title('Supernova Distance Modulus vs. Redshift (z<<1)')
plt.show()

plt.errorbar(np.log10(medium_z_data["redshift"]), medium_z_data["distance␣
↪modulus"], yerr = medium_z_data["distance modulus error"], xerr=None,␣
↪fmt='o')
```

```
plt.xlabel('log10(Redshift)')
plt.ylabel('Distance Modulus')
plt.title('Supernova Distance Modulus vs. Redshift (z<0.2)')
plt.show()
```





1.3 3. Fit Data and Calculate Hubble Constant and Cosmological Constant

Equation 1: $m - M = 43.17 - 5log_{10}(\frac{H_0}{70}) + 5log_{10}z + 1.086(1 - q_0)z$

Observe that when z«1, we can neglect the last term $1.086(1 - q_0)z$. The distance modulus and $log_{10}z$ are linear. So we can use the data when z«1 to fit a linear regression model. The fitted intercept can be used to calculate Hubble constant H_0.

To take into account the error bars, we use weighted least squares where the weights are calculated from distance modulus error.

[161]:
```
X = np.array(np.log10(small_z_data["redshift"])).reshape(-1, 1)
X = sm.add_constant(X) # add column of 1 for intercept
y = np.array(small_z_data["distance modulus"])

# Weights for weighted least squares
weights = 1 / np.array(small_z_data["distance modulus error"]) ** 2

H0_model = sm.WLS(y, X, weights=weights).fit()
H0_model.params
```

[161]:
```
array([43.26060668,  5.03219119])
```

[162]:
```
H0_model.summary()
```

[162]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            WLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.951
Model:                            WLS   Adj. R-squared:                  0.951
Method:                 Least Squares   F-statistic:                     2806.
Date:                Fri, 05 May 2023   Prob (F-statistic):           2.55e-96
Time:                        22:52:23   Log-Likelihood:                 43.463
No. Observations:                 146   AIC:                            -82.93
Df Residuals:                     144   BIC:                            -76.96
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         43.2606      0.147    293.727      0.000      42.969      43.552
x1             5.0322      0.095     52.973      0.000       4.844       5.220
==============================================================================
Omnibus:                        1.446   Durbin-Watson:                   1.654
Prob(Omnibus):                  0.485   Jarque-Bera (JB):                1.013
Skew:                           0.126   Prob(JB):                        0.603
Kurtosis:                       3.320   Cond. No.                         22.7
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

[163]:
```python
H0 = 70 * 10**((H0_model.params[0] - 43.17) / (-5.0))
print("H0:", H0)
```

```
H0: 67.13928363809563
```

Our linear model fits the data well with R^2 close to 1. We calculate H_0 around 67.14, which is also consistent with other studies.

Now, we could use slightly bigger z to fit the last term $1.086(1-q_0)z$ to calculate q_0.

[164]:
```python
X = np.array((medium_z_data["redshift"])).reshape(-1, 1)
X = sm.add_constant(X)
y = np.array(medium_z_data["distance modulus"]) - 5*np.array(np.
 ↪log10(medium_z_data["redshift"])) - H0_model.params[0]

# Weights for weighted least squares
weights = 1 / np.array(medium_z_data["distance modulus error"]) ** 2

q0_model = sm.WLS(y, X, weights=weights).fit()
```

```
q0_model.params
```

[164]:
```
array([-0.09279572,  1.43659398])
```

[165]:
```
q0 = 1 - q0_model.params[1]/1.086
print("q0:", q0)
```

```
q0: -0.3228305557712643
```

We get $q_0 < 0$, indicating that we are in an accelarating outward universe.

We could also directly fit data with small redshift to the whole equation, by indicating the model as $m - M = b + a_0 log_{10} z + a_1 z$.

[166]:
```
X0 = np.array(np.log10(medium_z_data["redshift"])).reshape(-1, 1)
X1 = np.array((medium_z_data["redshift"])).reshape(-1, 1)
X = np.append(X0, X1, axis = 1)

X = sm.add_constant(X)
y = np.array(medium_z_data["distance modulus"])

# Weights for weighted least squares
weights = 1 / np.array(medium_z_data["distance modulus error"]) ** 2

model = sm.WLS(y, X, weights=weights).fit()
model.params
```

[166]:
```
array([43.1024494 ,  4.96133736,  1.66229566])
```

[167]:
```
H0 = 70 * 10**((model.params[0] - 43.17) / (-5.0))
print("H0:", H0)
q0 = 1 - model.params[2]/1.086
print("q0:", q0)
```

```
H0: 72.21179809082025
q0: -0.530658990003376
```

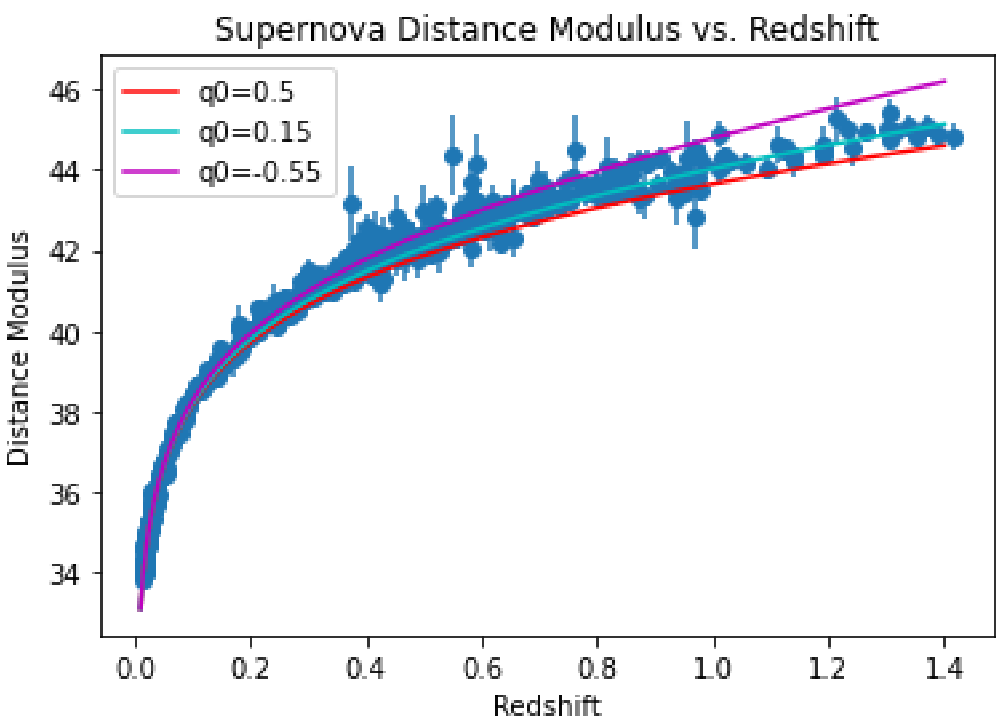
1.4 4. Compare Observational Results and Expected Results for three Model Universes

One universe is flat, and contains only matter ($\Omega_{m,0} = 1, q_0 = 0.5$). The second universe is negatively curved, and contains only matter ($\Omega_{m,0} = 0.3, q_0 = 0.15$). The third universe is flat, and contains both matter and a cosmological constant ($\Omega_{m,0} = 0.3, \Omega_{\Lambda,0} = 0.7, q_0 = -0.55$). The data are best fitted by the thrid model.

[168]:
```
def calculate_dm(q0, z_data):
    dm = model.params[0] + 5*np.array(np.log10(z_data)) + 1.086*(1-q0)*np.
 ↪array(z_data)
    return dm
```

[169]:

```
plt.errorbar(sn_data["redshift"], sn_data["distance modulus"], yerr =␣
↪sn_data["distance modulus error"], xerr=None, fmt='o', zorder=0)

z_data = np.linspace(10**(-2), 1.4, num=300)

plt.plot(z_data, calculate_dm(0.5, z_data), 'r', label="q0=0.5")
plt.plot(z_data, calculate_dm(0.15, z_data), 'c', label="q0=0.15")
plt.plot(z_data, calculate_dm(-0.55, z_data), 'm', label="q0=-0.55")


plt.xlabel('Redshift')
plt.ylabel('Distance Modulus')
plt.title('Supernova Distance Modulus vs. Redshift')
plt.legend()
plt.show()
```

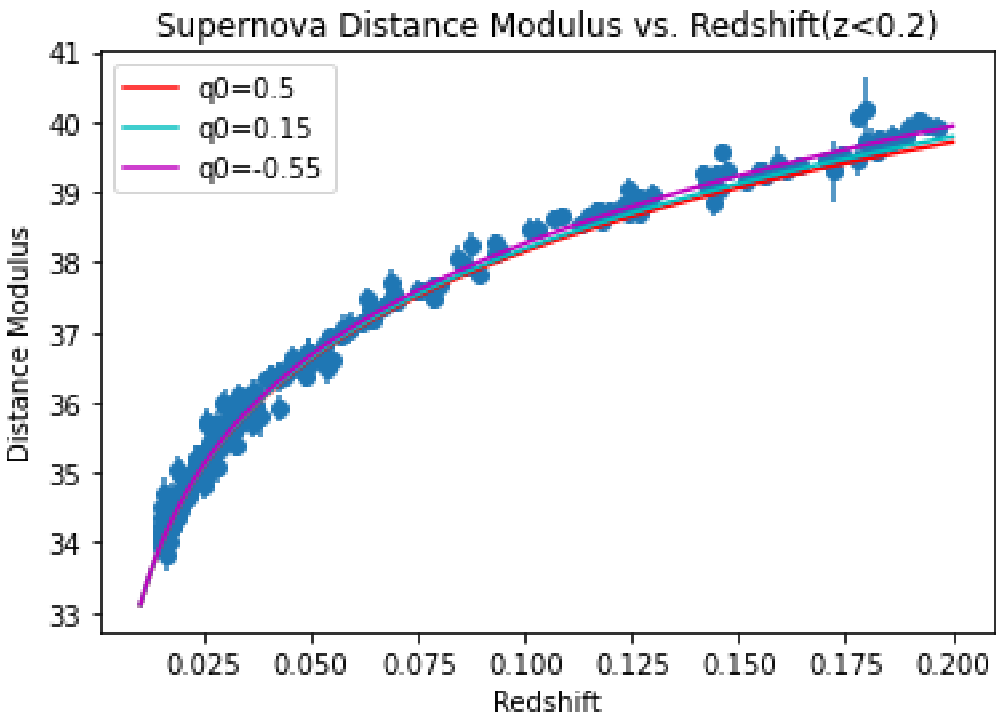


Comparing the expected results of three model universes and our observed results, we found that if using the whole dataset when z ranges from 0 to 1.4, the model of q0=0.15 seems to fit best. Yet, equation 1 is most accurate when redshift z is small, so let's fit these models to smaller redshift data.

[170]:

```
plt.errorbar(medium_z_data["redshift"], medium_z_data["distance modulus"], yerr␣
↪= medium_z_data["distance modulus error"], xerr=None, fmt='o', zorder=0)
```

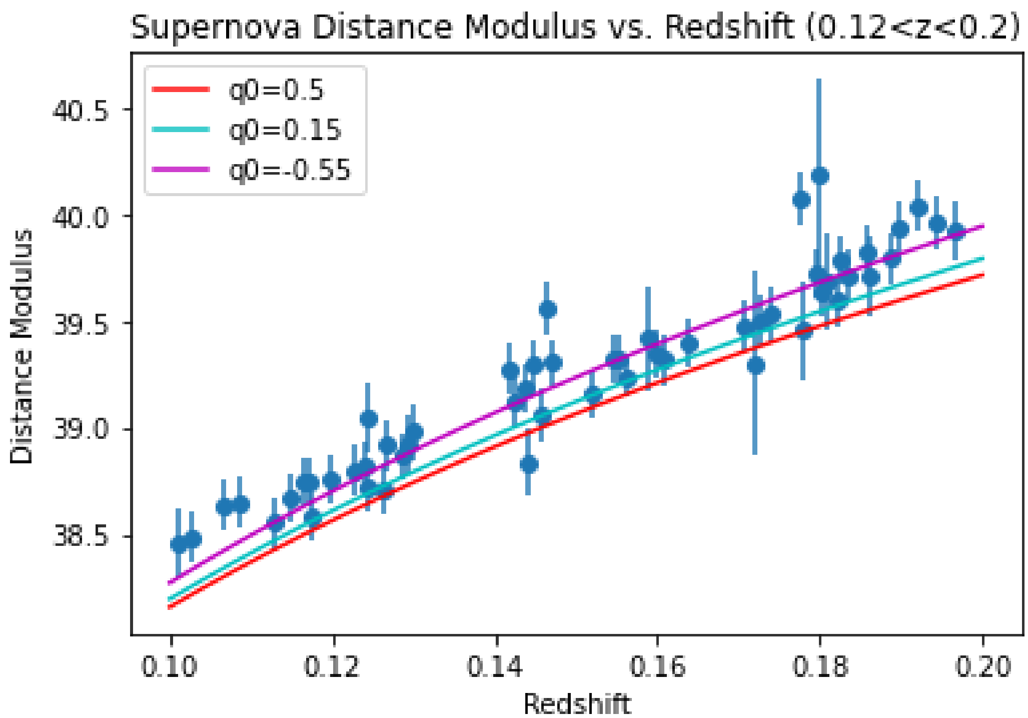
```
z_data = np.linspace(10**(-2), 0.2, num=300)

plt.plot(z_data, calculate_dm(0.5, z_data), 'r', label="q0=0.5")
plt.plot(z_data, calculate_dm(0.15, z_data), 'c', label="q0=0.15")
plt.plot(z_data, calculate_dm(-0.55, z_data), 'm', label="q0=-0.55")


plt.xlabel('Redshift')
plt.ylabel('Distance Modulus')
plt.title('Supernova Distance Modulus vs. Redshift(z<0.2)')
plt.legend()
plt.show()
```



We see that three lines are close when z is small, since the last term $1.086(1-q_0)z$ of Equation 1 is neglegible when z«1. At larger z, three lines deviates. We can zoom in the parts where three line deviates to observe better.

[171]:
```
medium_large_z_data = medium_z_data[medium_z_data["redshift"] > 0.1]
plt.errorbar(medium_large_z_data["redshift"], medium_large_z_data["distance␣
 ↪modulus"], yerr = medium_large_z_data["distance modulus error"], xerr=None,␣
 ↪fmt='o', zorder=0)
```

```
z_data = np.linspace(0.1, 0.2, num=200)

plt.plot(z_data, calculate_dm(0.5, z_data), 'r', label="q0=0.5")
plt.plot(z_data, calculate_dm(0.15, z_data), 'c', label="q0=0.15")
plt.plot(z_data, calculate_dm(-0.55, z_data), 'm', label="q0=-0.55")


plt.xlabel('Redshift')
plt.ylabel('Distance Modulus')
plt.title('Supernova Distance Modulus vs. Redshift (0.12<z<0.2)')
plt.legend()
plt.show()
```



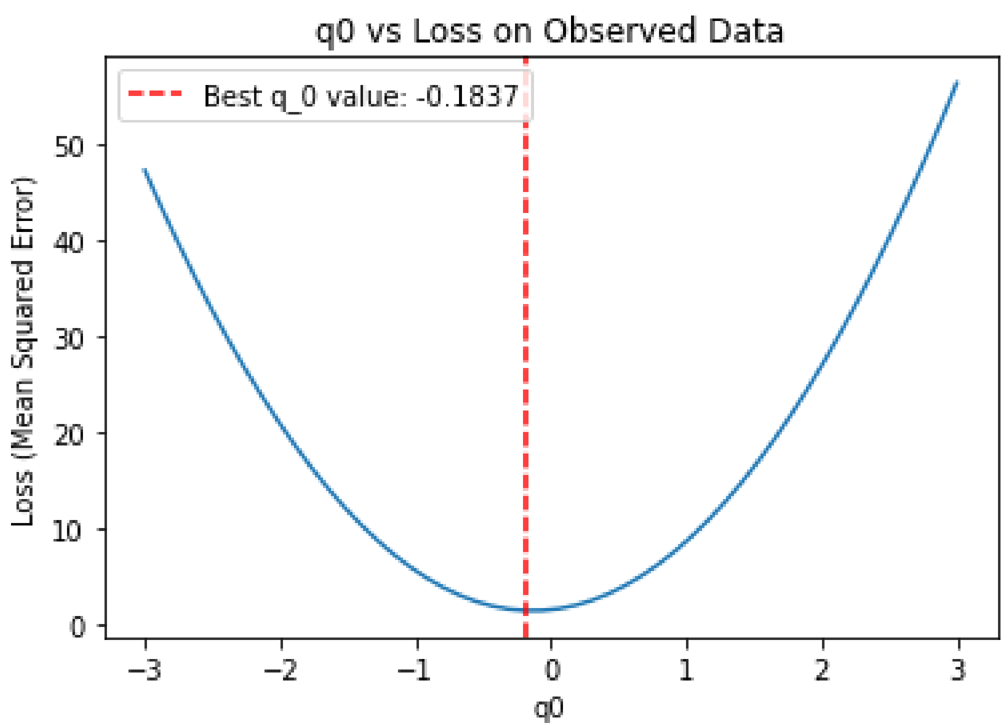
After zooming in, we observe that the third model, where $q_0 = -0.55$ best describes our data, which is consistent with our findings in Part 3.

1.5 5. Fitting with Different Cosmological Constants

[200]:
```
def weighted_mean_squared_error(predict, true, weights):
    return np.mean(weights * (predict - true) ** 2)
```

[201]:
```
losses = []
q0s = np.linspace(-3, 3, 50)
```

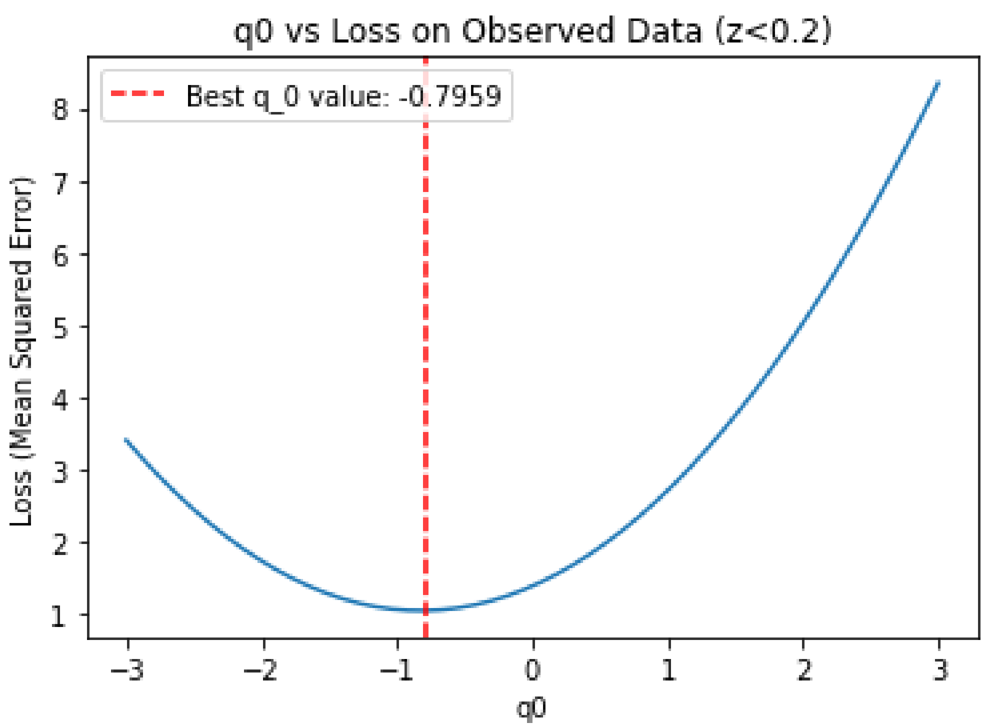
```
weights = 1 / np.array(sn_data["distance modulus error"]) ** 2

for q0 in q0s:
    prediction = [calculate_dm(q0, z) for z in sn_data["redshift"]]
    losses.append(weighted_mean_squared_error(prediction, sn_data["distance␣
↪modulus"], weights))

best_q0 = q0s[np.argmin(losses)]
plt.plot(q0s, losses)
plt.axvline(x=best_q0, color='r', linestyle="--", label=f"Best q_0 value:␣
↪{round(best_q0, 4)}")

plt.xlabel('q0')
plt.ylabel('Loss (Mean Squared Error)')
plt.title('q0 vs Loss on Observed Data')
plt.legend()
plt.show()
```



Here we found that q0=-0.1837 best fits our data if we use the whole dataset, but this is not an accurate measurement since our method is more accurate with small z_data, so let's try it now.

[202]:
```
losses = []
q0s = np.linspace(-3, 3, 50)
```

```
weights = 1 / np.array(medium_z_data["distance modulus error"]) ** 2

for q0 in q0s:
    prediction = [calculate_dm(q0, z) for z in medium_z_data["redshift"]]
    losses.append(weighted_mean_squared_error(prediction,␣
 ↪medium_z_data["distance modulus"], weights))

best_q0 = q0s[np.argmin(losses)]
plt.plot(q0s, losses)
plt.axvline(x=best_q0, color='r', linestyle="--", label=f"Best q_0 value:␣
 ↪{round(best_q0, 4)}")

plt.xlabel('q0')
plt.ylabel('Loss (Mean Squared Error)')
plt.title('q0 vs Loss on Observed Data (z<0.2)')
plt.legend()
plt.show()
```



1.6 6. Using Cosmolopy

Now, let's try different cosmological constants(values of Ω) and see what value fits our data best. We could use the cosmolopy package to calculate lumiosity distance accurately and then use that to calculate distance modulus.

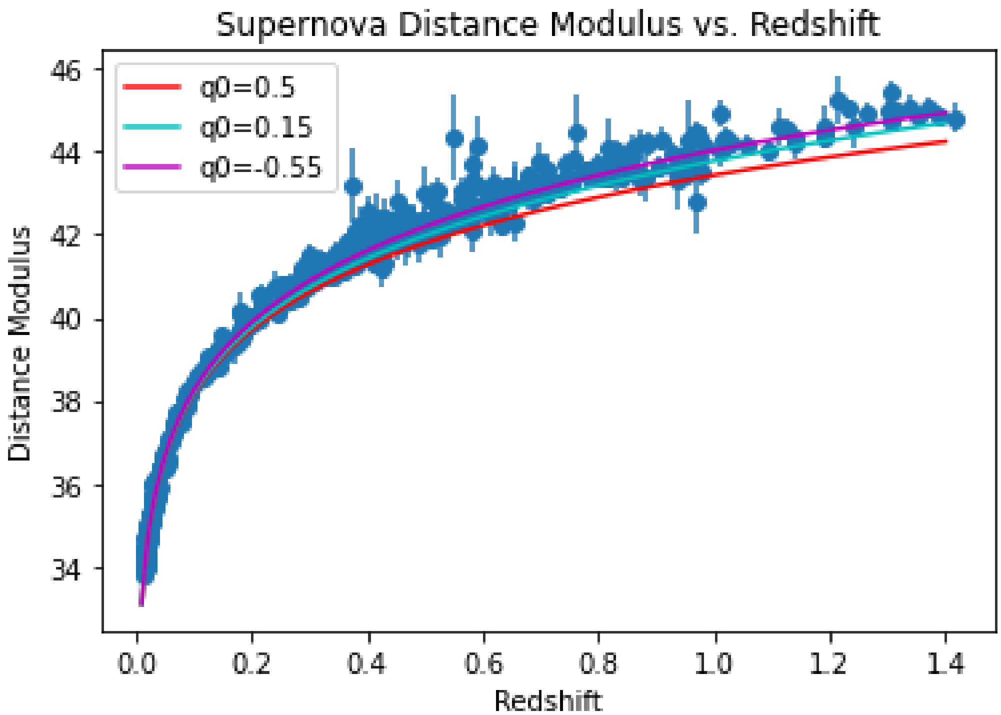
[203]:
```
import cosmolopy.distance as cd
import cosmolopy.constants as cc
```

[204]:
```
def calculate_dm_from_cosmolopy(omega_M, omega_lambda, z_data):
    cosmo = {'omega_M_0' : omega_M, 'omega_lambda_0' : omega_lambda, 'h' : H0}
    cosmo = cd.set_omega_k_0(cosmo)
    a = cd.luminosity_distance(z_data, **cosmo)
    #distfunc, redfunc = cd.quick_distance_function(cd.luminosity_distance,␣
 ↪return_inverse=True, **cosmo)

    dm = 5*np.log10(a/10**(-7))
    return dm
```

Let's first try using this cosmolopy package to generate expected results from three universes in Part 4.

As a reminder, 1st universe is flat, and contains only matter ($\Omega_{m,0} = 1, q_0 = 0.5$). 2nd universe is negatively curved, and contains only matter ($\Omega_{m,0} = 0.3, q_0 = 0.15$). 3rd universe is flat, and contains both matter and a cosmological constant ($\Omega_{m,0} = 0.3, \Omega_{\Lambda,0} = 0.7, q_0 = -0.55$). The data are best fitted by the thrid model.

[205]:
```
plt.errorbar(sn_data["redshift"], sn_data["distance modulus"], yerr =␣
 ↪sn_data["distance modulus error"], xerr=None, fmt='o', zorder=0)

z_data = np.linspace(10**(-2), 1.4, num=300)

plt.plot(z_data, calculate_dm_from_cosmolopy(1, 0, z_data), 'r', label="q0=0.5")
plt.plot(z_data, calculate_dm_from_cosmolopy(0.3, 0, z_data), 'c', label="q0=0.
 ↪15")
plt.plot(z_data, calculate_dm_from_cosmolopy(0.3, 0.7, z_data), 'm',␣
 ↪label="q0=-0.55")


plt.xlabel('Redshift')
plt.ylabel('Distance Modulus')
plt.title('Supernova Distance Modulus vs. Redshift')
plt.legend()
plt.show()
```



Here we clearly see that the benchmark model(q0=-0.55) fits our data most.

Now let's use a grid of different combinations of Ω_m and Ω_Λ to calculate the expected values of distance modulus, and calculate its difference with the observed data.

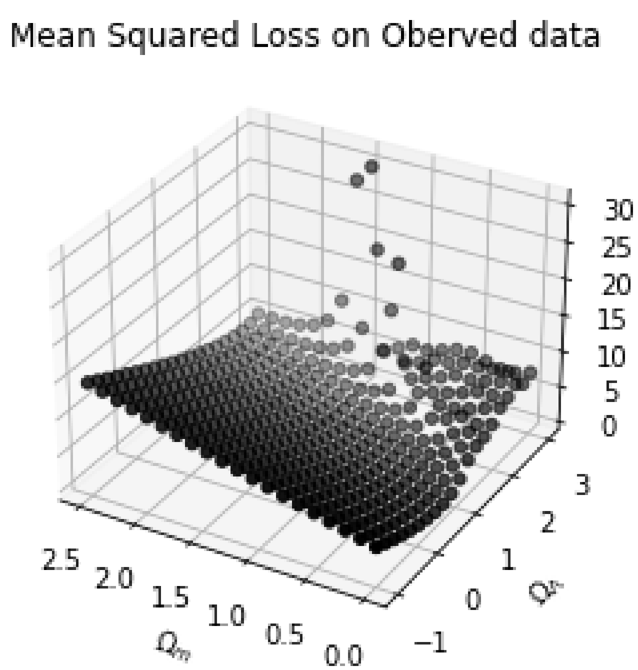
[207]:
```
omega_m = np.linspace(0, 2.5, num = 20)
omega_lambda = np.linspace(-1, 3, num = 20)
weights = 1 / np.array(sn_data["distance modulus error"]) ** 2

X, Y = np.meshgrid(omega_m, omega_lambda)
Z = np.copy(X)

for i in range(len(X)):
    for j in range(len(X[0])):
        m = X[i][j]
        labda = Y[i][j]
        dm_lst = calculate_dm_from_cosmology(m, labda, sn_data["redshift"])
        loss = weighted_mean_squared_error(dm_lst, sn_data["distance modulus"],␣
↪weights)
        Z[i][j] = loss
```

```
/Users/shentingwei/opt/anaconda3/lib/python3.9/site-
packages/cosmolopy/distance.py:102: RuntimeWarning: invalid value encountered in
double_scalars
  e_z = (omega_M_0 * (1+z)**3. +
```

```
/Users/shentingwei/opt/anaconda3/lib/python3.9/site-
packages/cosmolopy/distance.py:167: IntegrationWarning: The occurrence of
roundoff error is detected, which prevents
  the requested tolerance from being achieved.  The error may be
  underestimated.
  si.quad(_comoving_integrand, z0, z, limit=1000,
/Users/shentingwei/opt/anaconda3/lib/python3.9/site-
packages/numpy/lib/function_base.py:2197: RuntimeWarning: invalid value
encountered in <lambda> (vectorized)
  outputs = ufunc(*inputs)
/Users/shentingwei/opt/anaconda3/lib/python3.9/site-
packages/pandas/core/arraylike.py:364: RuntimeWarning: invalid value encountered
in log10
  result = getattr(ufunc, method)(*inputs, **kwargs)
/Users/shentingwei/opt/anaconda3/lib/python3.9/site-
packages/cosmolopy/distance.py:167: IntegrationWarning: Extremely bad integrand
behavior occurs at some points of the
  integration interval.
  si.quad(_comoving_integrand, z0, z, limit=1000,
```

[208]:

```
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.scatter(X, Y, Z, color='black')
ax.invert_xaxis()
ax.set_xlabel('$\Omega_m$')
ax.set_ylabel('$\Omega_{\Lambda}$')
ax.set_title('Mean Squared Loss on Observed data');
```



The graph above shows how different cosmological constants produce different losses. We can observe that the loss is lower when Ω_Λ is less than 1 and Ω_m is less than 1. Let's see what combination produces the least mean square error loss comparing with our observed data.

[209]:
```
ind = np.unravel_index(np.argmin(Z), X.shape)
print("Best fit Omega_m:", X[ind], "\nBest fit Omega_Lambda:", Y[ind])
```

```
Best fit Omega_m: 0.39473684210526316
Best fit Omega_Lambda: 1.1052631578947367
```

[ ]:
```

```