Astro 310 HW3

September 22, 2024

1 ASTR 310 HW 3

1.0.1 1. Binary Search (25 points total)

A common strategy in computer algorithms is "divide and conquer:" split the data you need to work on into two parts, then work on each of the parts. Continue splitting until the amount of data is so small that the operation to be performed is trivial. An example is the binary search algorithm, which searches a sorted list of objects (strings, numbers, etc.) for a specified object, then returns the location of the object in the list (or indicates that the specified object is not present in the list). In this problem you will implement binary search in Python. Yes, we know there are many versions of this binary search algorithm that you can find on the internet. The exercise here is to write your own, not copy someone else's. The basic binary search algorithm is as follows. We'll assume that we have a list L with N entries sorted in ascending order, and we are looking for an entry E.

Pseudocode:

```
set low_index to 0
set high_index to N-1
loop until E is found or high_index == low_index + 1:
    set test_index to int( (low_index + high_index) / 2 )
    if L[test_index] <= E then
        low_index = test_index  # keep top half
    otherwise
        high_index = test_index  # keep bottom half
```

If L[low index] or L[high index] equals E, you have found it; otherwise E is not present in the list.

a) First, obtain a sorted list of numbers. We'lll use the first 10,000 primes. Download the list at https://primes.utm.edu/lists/small/10000.txt. Your code should read this file and construct a 1D list of the primes. We suggest using readlines and split and append to deal with the fact that there are multiple primes on each line of the file, but other methods will work too. (You may have to manually strip out the four header lines and one footer line at the end.) [5 pts]

[128]:
```
f = open("primes.txt", 'r')
lines = f.readlines()
f.close()
lines = lines[4:-1]
nums = []
for l in lines:
```

```
    subs = l.split(" ")
    for sub in subs:
        if (sub != str('')) & (sub != str('\n')):
            nums.append(int(sub))
```

b) Write the code to use a binary search technique to search the list you constructed in part (a). [15 pts]

[129]:
```
from math import *
def binary_search(L: list, e: int):
    i_low = 0
    i_high = len(L) - 1

    while (i_low <= i_high):
        i_test = int((i_low + i_high) / 2)

        if L[i_test] == e:
            print(f"Target found at index {i_test}.")
            return
        elif L[i_test] <= e:
            i_low = i_test + 1
        else:
            i_high = i_test - 1



    print("Target not found in list.")
```

c) Test your code with the following search targets: [5 pts]
i. 98299
ii. 104729
iii. 105001
iv. 2
v. 1
Your program should print a message indicating the list index corresponding to each search target, or a message indicating that it was not found.

[130]:
```
binary_search(nums, 98299)
binary_search(nums, 104729)
binary_search(nums, 105001)
binary_search(nums, 2)
binary_search(nums, 1)
```

```
Target found at index 9438.
Target found at index 9999.
Target not found in list.
Target found at index 0.
Target not found in list.
```

1.0.2 2. Improved cosmological computation (25 points)

Modify your program from problem #3 of homework #2 to accept a more general choice of cosmology in addition to the flat Λ cosmology we originally worked with.

The angular diameter distance formula for a flat cosmology was given in HW 2.
For a matter-only universe, the angular diameter distance is given by

$$d_A = \frac{c}{H_0 q_0^2} \frac{z q_0 + (q_0 - 1)(\sqrt{2 z q_0 + 1} - 1)}{(1+z)^2} \quad \text{with} \quad q_0 \equiv -\frac{\ddot{a} a}{\dot{a}^2} = \frac{\Omega_m}{2}.$$

This is a classic equation known as the Mattig relation. The quantity q_0 is called the deceleration parameter, and $a = (1+z)^{-1}$ is the scale factor of the universe.

We're also interested in the luminosity distance d_L. The luminosity distance is defined as the distance measure that preserves the r^{-2} law relating luminosity and flux, and it is given by $d_L = d_A(1+z)^2$.

a) The program should first ask the user for the value of Ω_m, then ask if the user wants to assume $\Omega_\Lambda = 1 - \Omega_m$ (a flat cosmology) or $\Omega_\Lambda = 0$ (a matter-only cosmology). [3 pts]

The program should also ask the user if they wish to specify a range of redshifts (min, max, increment) or a space-separated list of redshifts at which to compute dA. On the basis of this choice, either accept three numbers as input and generate the list of redshifts to use, or else accept the list of redshifts from the user. [3 pts]

[131]:
```
from scipy import constants as const
import numpy as np
import math

# Calculations for a flat cosmology
def flat_cosmology(H, m, z):

    H = H * 3.24078e-20 # convert to 1/s

    def n(a, o):
        s = ((1 - o) / o) ** (1/3)
        return 2 * math.sqrt(s**3 + 1) * ((1/a**4) - (0.1540 * s/a**3) + (0.
↪4304*s**2/a**2) + (0.19097*s**3/a) + (0.066941*s**4))**(-1/8)


    da = 1/(1+z) * (2.998e8 / H) * (n(1,m) - n((1/(1+z)),m))
    da # m/radian

    da_kpc_arcsec = da / 206265 / 3.086e+19
    da_mpc_radian = da / 3.086e+22

    dl = da_mpc_radian * (1 + z) ** 2

    return {"r": z, "a": 0, "da": da_kpc_arcsec, "dl": dl}
```

```
# Calculations for general cosmology
def gen_cosmology(H, m, z):

    H = H * 3.24078e-20 # convert to 1/s

    a = (1 + z) ** -1
    q0 = m / 2.0

    da = (const.c / (H * q0 ** 2)) * ((z * q0) + ((q0 - 1) * (sqrt(2 * z * q0 +␣
↪1) - 1))) / ((1 + z) ** 2)
    da # m/radian

    da_kpc_arcsec = da / 206265 / 3.086e+19
    da_mpc_radian = da / 3.086e+22

    dl = da_mpc_radian * (1 + z) ** 2

    return {"r": z, "a": a, "da": da_kpc_arcsec, "dl": dl}
```

b) Once the program has the cosmology and the list of redshifts to use, cycle through the list and compute d_A for each redshift using the appropriate cosmological expression.

Also for each redshift, compute the luminosity distance d_L in Mpc. (Note that here you need to use d_A in units of Mpc/radian to get d_L in Mpc.)

As you go along, print values of z, a, d_A, and d_L in a nicely formatted table using three digits of precision after the decimal. Print d_A in units of kpc/arcsec as before. [15 pts]

Run your program for both types of cosmological model using the same values of H_0 and Ω_m used in homework 2, and use a range of 10 or 11 redshifts between 0 and 5. If you want to check your results, try visiting Ned Wright's Cosmology Calculator page at https://www.astro.ucla.edu/~wright/CosmoCalc.html [4 pts]

[133]:
```
# THIS IS THE FINAL PROGRAM
def final_program():
    H = 72
    m_matter = float(input("Matter Density Parameter: "))
    matter_only = input("General (1) or Flat Model (2): ") == "1"

    if int(input("List (1) or Range (2) of Redshifts: ")) == 1:
        redshifts = input("Enter space-separated list of redshifts: ").split("␣
↪")
    else:
        params = input("Enter min, max, and increment (space-separated): ").
↪split(" ")
        redshifts = np.arange(float(params[0]), float(params[1]),␣
↪float(params[2])).tolist()

    if matter_only:
```

```
        print("Matter Only (general)")
        print("{r:5} {a:5} {da:8} {dl:8}".format(r="z", a="a", da="da (kpc/
↪\")", dl="dl (Mpc)"))
        ans = [gen_cosmology(H, m_matter, r) for r in redshifts]
        for e in ans:
            print("{r:5.3f} {a:5.3f} {da:8.3e} {dl:8.3e}".
↪format(r=e['r'],a=e['a'],da=e['da'],dl=e['dl']))
    else:
        print("Flat Cosmology")
        print("{r:5} {a:5} {da:8} {dl:8}".format(r="z", a="a", da="da (kpc/
↪\")", dl="dl (Mpc)"))
        ans = [flat_cosmology(H, m_matter, r) for r in redshifts]
        for e in ans:
            print("{r:5.3f} {a:5.3f} {da:8.3e} {dl:8.3e}".
↪format(r=e['r'],a=e['a'],da=e['da'],dl=e['dl']))
```

[134]:
```
# Run for General Model
# H = 72
# m = 0.26
# z = range(0,5,0.5)
final_program()
```

```
Matter Only (general)
z     a     da (kpc/") dl (Mpc)
0.000 1.000 0.000e+00 0.000e+00
0.500 0.667 5.402e+00 2.507e+03
1.000 0.500 6.995e+00 5.771e+03
1.500 0.400 7.507e+00 9.678e+03
2.000 0.333 7.616e+00 1.414e+04
2.500 0.286 7.553e+00 1.908e+04
3.000 0.250 7.411e+00 2.446e+04
3.500 0.222 7.233e+00 3.021e+04
4.000 0.200 7.041e+00 3.631e+04
4.500 0.182 6.847e+00 4.272e+04
```

[135]:
```
# Run for Flat Model
# H = 72
# m = 0.26
# z = range(0,5,0.5)
final_program()
```

```
Flat Cosmology
z     a     da (kpc/") dl (Mpc)
0.000 0.000 0.000e+00 0.000e+00
0.500 0.000 6.036e+00 2.801e+03
1.000 0.000 7.990e+00 6.592e+03
1.500 0.000 8.491e+00 1.095e+04
2.000 0.000 8.433e+00 1.565e+04
```

```
2.500 0.000 8.156e+00 2.061e+04
3.000 0.000 7.801e+00 2.575e+04
3.500 0.000 7.429e+00 3.103e+04
4.000 0.000 7.064e+00 3.643e+04
4.500 0.000 6.719e+00 4.193e+04
```