ASTP 720 Homework Assignment 2

Isabella Cox – `igc5972@rit.edu`

September 10, 2020

Task 1

Write a numerical calculus library, one that can perform at least one derivative (e.g., the symmetric one) and at least three integration functions including the midpoint rule, trapezoidal rule, and Simpson's rule.

This library exists in a file called: `calculus_library.py`.

The derivative function calculates a symmetric derivative using the following equation:

$$f'(x_i) = \frac{f(x+h) - f(x-h)}{2h} \tag{1}$$

where h is the incremental difference, the width of which to look around the desired point to take sort of an average.

The first integral method is the rectangular Riemann sums, I chose to do left Riemann sums where the height of the rectangle was set as the functional value for the left edge of the rectangle. The equation of the integral of each subunit is as follows:

$$\int_a^b f(x)dx = h \sum_{i=0}^{N-1} y_i \tag{2}$$

where h is the width of the sub-interval and is determined by dividing the total with of the overall interval by the number of intervals desired and y_i is the functional value evaluate at the left hand side of the sub-interval.

The second method for integration is the Trapezoid method, which works by using a functional height that is the average of the two endpoints. For an individual sub-interval, the area is:

$$\int_{x_i}^{x_{i+1}} f(x)dx = h\frac{y_{i+1} + y_i}{2} \tag{3}$$

The third and final method of integration for this assingment is Simpson's rule. Three consecutive points are used for this, and because of the limits of summing, the number of intervals (N) must be evenly divisible by two. The equation is:

$$\int_a^b f(x)dx = h/3 \sum_{i=0}^{N/2-1} y_{2i} + 4y_{2i+1} + y_{2i+2} \tag{4}$$

Task 2

Using your chosen parameters of c,v_{200}, and. r_{200}, numerically determine the mass enclosed $M_{enc}(r)$ (and plot), the total mass of the dark matter halo M, and then also M(r), the amount of mass in a little shell around $r \pm \delta r$ (i.e., what does the mass profile look like?), and $dM(r)/dr$. Compare this by repeating, holding c fixed and changing v_{200}. Again compare the first by repeating, holding $v_2 00$ fixed and changing c. Feel free to use your previous tools from last time if you find that you need to (though I don't think you should); in the future you'll be able to use built-in functions for those.

Chosen Parameters: c = 15 v_{200} = 160 km/s r_{200} = 230 kpc

To numerically determine the enclosed mass as a fucntion of radius, use Equation 5,

$$M_{enc}(r) = \frac{rv_c^2(r)}{G} \tag{5}$$

where G = $6.67 \times 10^{-20} km^3 kg^{-1} s^{-2}$ and $v_c^2(r)$ is given in Equation 6,

$$v_c^2(r) = v_{200}^2 \frac{1}{x} \frac{ln(1+cx) - \frac{cx}{1+cx}}{ln(1+c) - \frac{c}{1+c}} \tag{6}$$

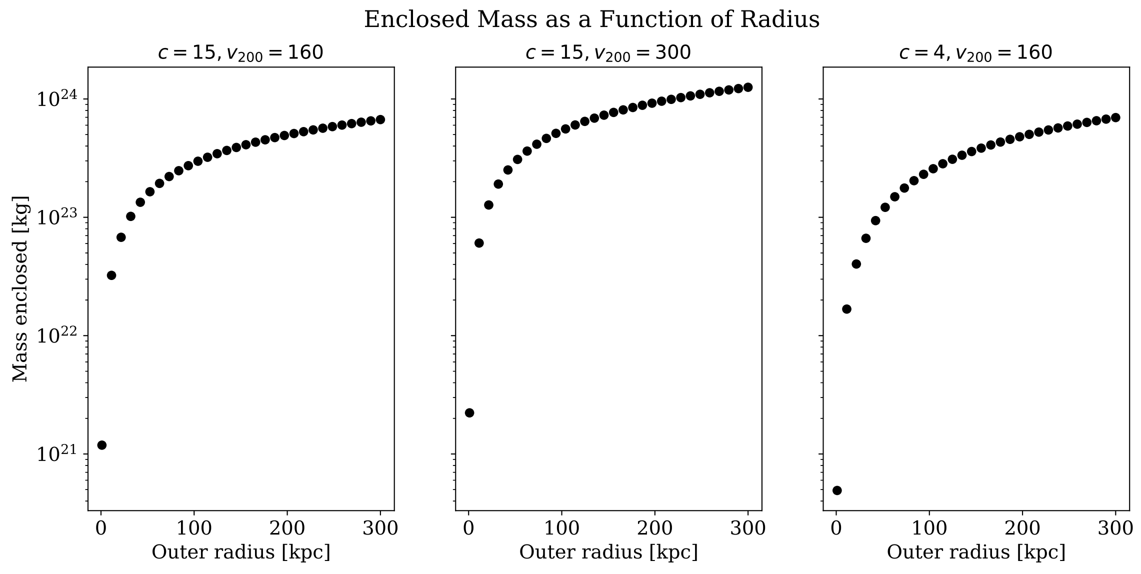

Figure 1: M_{enc} as a function of r where M_enc denotes all of the mass of the dark matter halo up until radius r. In the left panel, the parameters are a concentration (c) value of 15, and a v_{200} of 160 where v_{200} is the velocity at the viral radius. For the center panel, c = 15, v_{200} = 300 and for the right panel, c = 4, v_{200} = 160. This layout will be followed throughout this write-up.

where $x = r/r_{200}$.

By integrating the expression for M_{enc} from the limits $0 \rightarrow r$ for different values of r, we can plot the enclosed mass as a function of radius, as shown in Figure 1. It is evident from this Figure that a larger value of virial velocity increases the enclosed mass and a smaller value of concentration decreases the enclosed mass, so both quantities are positively correlated with the mass enclosed.

The total mass of the dark halo can be estimated by integrated the expression for M_{enc} from $0 \rightarrow \sim \infty$.

$\int_0^\infty M_{enc}(r) = ??$

Something was wrong with my expression because as I changed the upper limit for integration, I never found a number large enough that above that the M_{enc} stayed stable.

The other sub task for this task is to calculate $M(r)$ which is the mass enclosed by a shell of half width Δdr. I calculated this this using Equation 7 and the results (plotting M(r) as

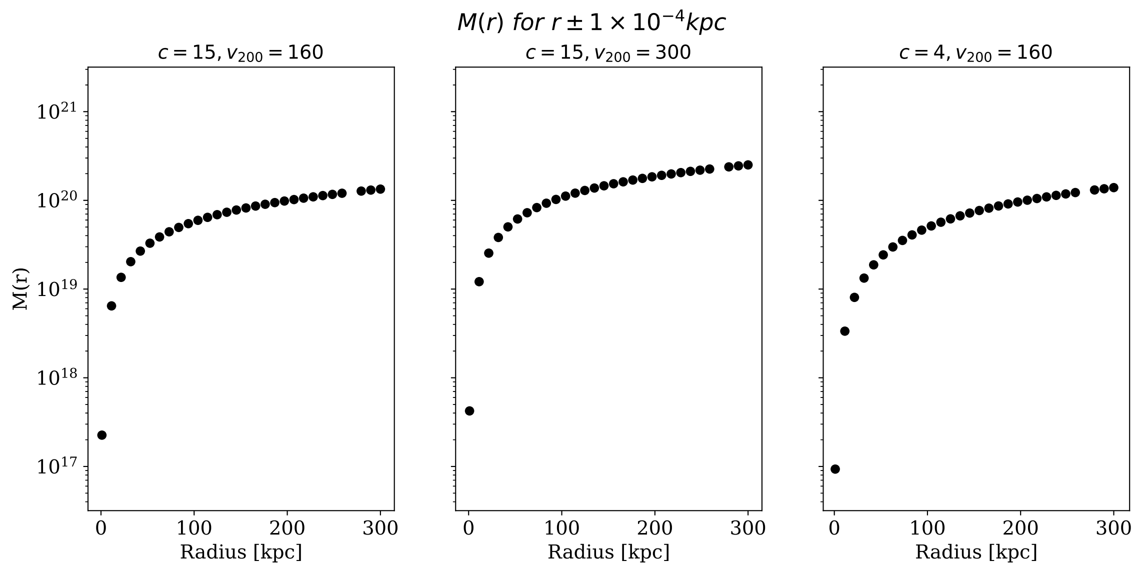

Figure 2: M(r) as a function of r where M(r) calculates the mass in a shell at radius r with width $2\Delta r$. It is clear that both c and v_{200} are positively correlated with $M(r)$.

function of r are in Figure 2).

$$\int_0^{r+\Delta r} M_{enc}(r)dr - \int_0^{r-\Delta r} M_{enc}(r)dr \tag{7}$$

The differential bit $\frac{dm}{dr}$ is calculated by passing Equation 7 to my derivative function I wrote earlier.

Task 3

In preparation for an actual astrophysical calculation moved into the next homework, write a matrix library that includes a Matrix class. There's lots of material online on how to write a class. Your class should be able to:

1. Add two matrices together
2. Multiply two matrices together
3. Transpose a matrix
4. Invert a matrix

5. Calculate trace of a matrix
6. Calculate determinant of matrix
7. Return LU decomponsition of matrix (return two Matrix objects representing L and U).

I have written a Matrix class saved in `matrix_library.py`. For each of the subtasks 1-7, I will demonstrate them on the main matrix, A and if another matrix is needed for the function, B.

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 & 2 \\ 3 & -3 & 3 & 1 \\ 7 & 8 & 9 & 1 \\ -2 & 4 & 9 & 7 \end{pmatrix} \quad \mathbf{B} = \begin{pmatrix} 3 & 4 & 5 & 2 \\ 3 & -7 & 1 & 8 \\ 2 & 8 & -2 & 8 \\ 1 & 3 & 1 & 4 \end{pmatrix}$$

1. Add two matrices together

To add two matrices together, the function `__add__` first checks that the dimensions of **A** and **B** match. If so, the summed matrix $((A + B))$ is computed as:

$$(\mathbf{A} + \mathbf{B})_{\mathbf{i,j}} = \mathbf{A}_{\mathbf{i,j}} + \mathbf{B}_{\mathbf{i,j}}$$

where i, j indexes into the i^{th} rows and j^{th} column of a matrix.

$$\mathbf{A} + \mathbf{B} = \begin{pmatrix} 4 & 6 & 8 & 4 \\ 6 & -10 & 5 & 9 \\ 9 & 16 & 7 & 9 \\ -1 & 7 & 10 & 11 \end{pmatrix}$$

2. Multiply two matrices together

To multiply two matrices together, the function `__mul__` first checks that the number of columns of **A** matches the number of columns of **B**. If so, the product of the two matrices $((A \times B))$ is computed as:

$$(\mathbf{A} \times \mathbf{B})_{\mathbf{i,j}} = \sum_{k=1}^{n} \mathbf{A}_{\mathbf{i,k}} \times \mathbf{B}_{\mathbf{k,j}}$$

where n is columns in **A** and the number of rows in **B**

$$\text{A } \times B = \begin{pmatrix} 17 & 20 & 3 & 50 \\ 9 & 68 & 5 & 18 \\ 64 & 47 & 26 & 154 \\ 31 & 57 & -17 & 128 \end{pmatrix}$$

3. Transpose a matrix

The transpose of a matrix is the same matrix with the position of the row and column indices are swapped. This looks like:

$\mathbf{A^T_{i,j}} = \mathbf{A_{j,i}}$

$$\mathbf{A^T} = \begin{pmatrix} 1 & 3 & 7 & -2 \\ 2 & -3 & 8 & 4 \\ 3 & 4 & 9 & 9 \\ 2 & 1 & 1 & 7 \end{pmatrix}$$

4. Invert a matrix

The inverse of a matrix can be determined by performing back substitution with the upper triangle form (see subsection 6 for how upper triangle form is determined).

The upper triangle form of a matrix is used to solve a system of equations where $\mathbf{AA^{-1}} = \mathbb{1}$. So each column of the inverse matrix can be solved for by using the corresponding column of the identity matrix. The coefficients of each column of the inverse matrix can be determined by

$$x_i = \frac{1}{\mathbf{A_{ii}}}\left[g_i - \sum_{j=i+1}^{n} \mathbf{A_{ij}} x_j\right] \tag{8}$$

where $i = n-1, n-2, ..., 1$ and g is the column vector of the corresponding identity matrix values. You can see that this is called substitution because each value in the column vector depends on the previous value. And we can successfully determining using our `__mult__` function to find that $\mathbf{A} \times \mathbf{A^{-1}} = \mathbb{1}$

$$\mathbf{A^{-1}} = \begin{pmatrix} 0.143 & -0.182 & 0.068 & 0.821 \\ 0 & 0 & 8-0.153 & 0.095 \\ 0 & 0 & 0 & -0.867 \\ 0 & 0 & 0 & 1.295 \end{pmatrix}$$

5. Calculate trace of a matrix

The trace of a matrix is the sum of its diagonal elements. The trace of matrix $\mathbf{A}$ is:

$\text{trace}(\mathbf{A}) = 14$

6. Calculate determinant of a matrix

To calculate the determinant of a matrix, I first did Gaussian elimination to put the matrix in upper triangle form. Then, the determinant can be calculated as the product of the diagonal entries.

The first step is to ensure that the pivot (the diagonal entry of each row) is not equal to 0, or if it is, that there is an entry in a column below the pivot that is non-zero. The reason why the second case is acceptable is because the code is setup to perform row swapping as needed. The row swapping works by iterating through each row and identifying the diagonal element of that row. Then, the code iterates through the elements in the same column but in a row below the diagonal element. If any of those values below are larger, the row with the larger values swaps places with the row with the current diagonal element.

After the row swapping, the Gaussian eliminations happens. First, each row is iterated through to identify the diagonal element of the row, call it, X. Next, for all the elements below X, the factor, f is determined as:

$$f = \mathbf{A_{ri}}/X \tag{9}$$

where the r indexes the row for the rows below X and i indexes the column as all these matrices are square (i.e., i indexes into row and column for diagonal entries).

Next, for each element in each row, the element is multiplied by the factor f derived for the row in Equation 8 and then that product is subtracted from the current element. The resultant matrix will have zeroes below each of its diagonal entries, so it will be in "upper diagonal form." And finally, the determinant can be taken as product of the elements along the diagonal once it is in upper triangle form. But it is important to note that the sign of the determinant is related to the number of rows swaps that were completed, so that must be considered. The results are

$$\mathbf{A_{upper\_tri}} = \begin{pmatrix} 7 & 8 & 9 & 1 \\ 0 & 6.285 & 11.571 & 7.285 \\ 0 & 0 & 11.977 & 8.023 \\ 0 & 0 & 0 & 0.772 \end{pmatrix} \quad \text{and } |A| = -407.$$

7. Return LU decomposition of a matrix

The LU decomposition breaks the matrix into two matrices, **L** and **U** where the former is a lower left triangle matrix and the latter is an upper right triangle matrix. These matrices are important because $\mathbf{A} = \mathbf{LU}$. To calculate these matrices, first, the first row of **L** is the same as the first row of **A** and the first column of **U** is the same as the first column of **A**.

This is constructed from the base cases where

$$\mathbf{L_{11}} = 1, \quad \mathbf{U_{1j}} = \mathbf{A_{1j}}/L_{11}, \quad \mathbf{L_{i1}} = \mathbf{A_{i1}}/\mathbf{U_{11}}.$$

Then, you can iterate over all the rows and for each row, do two separate iterations over columns, one each for **L** and **U**. To populate **L**, iterate from index 0 to $\rightarrow r$ and for populating **U**, iterate from index $= r \rightarrow n$ where r is the current row's index and n is the length of one side of the matrix (square matrix assumed).

Within each loop, the element [i][j] is calculated as

$= (\mathbf{A_{ij}} - \sum_{k=1}^{x-1} L_{ik}U_{lj})$ / Y_{xx}

where x (the terminating index) is equal to the current row's index for populating **U** and the current column's index for populating **L**, Y is equal to either **L** or **U** and i indexes the rows and j indexes the columns.

I was unable to get my code to work. I have a problem somewhere with the indicing I think. I know my code is not working because I compared the output to the `scipy.linalg` routine for LU decomposition. My **U** matrix is of the correct triangular form, but my **L** is not. This is an issue I will work on fixing before the next homework assignment.