CSCI 2202 PYTHON FOR SCIENTISTS LAB D 16 MARCH 2021
INTEGRATION & MATRIX APPLICATION

(1) **Representing Networks as Matrices:**
Graphs are an important tool used to model problems in many areas of applied sciences. Graphs are defined to be sets of *vertices* $V = \{v_1, v_2, v_3, v_4, v_5\}$ together with the connections (*edges*) between them, represented as ordered pairs:
$\{v_1, v_2\}, \{v2, v1\}, \{v_2, v_5\}, \{v_3, v_2\}, \{v_3, v_5\}, \{v_4, v_3\}, \{v_4, v_5\}, \{v_5, v_2\}, \{v_5, v_4\}$. An actual network may have many thousands of vertices. The graph can be represented as an $n \times n$ matrix ($n = 5$ in this example), with entries:

$$a_{ij} = \begin{cases} 1 \quad 1 \quad \text{if} \quad (v_i, v_j) \quad \text{is an edge} \\ 2 \quad 0 \quad \text{if no edge between } v_i \quad \text{and} \quad v_j \\ 3 \quad 0 \quad \text{if} \quad i = j \end{cases}$$

Create the adjacency matrix A for the graph. Note that A is symmetric (*i.e.* $ai, j = a_{j,i}$). A *walk* on a graph is a sequence of *edges* linking one vertex to another. In general, we can determine the number of walks of length k between any two vertices, by taking the k^{th} power of A.
Write a program to calculate the number of walks of length k, entered by the user, between vertices of the above graph. Report your results for $k = 3$.

(2) **Numerical Integration** (Empirical approach):
In this exercise, you are going to experimentally examine the errors in the the two rules of integration we examined in class: **The Trapezoidal Rule** and the **Midpoint Rule**, using the integral:

$$\int_1^2 \frac{1}{x} dx$$

which has the *exact* value: $\ln 2$ (the natural log of 2).

(a) Write a function `trapezoidal(f, a, b, n)` (see Lecture 8 note: **Integration** for details) and save it in a file: `integrate.py`. The function `trapezoidal(f, a, b, n)` takes the function `f` to be integrated, the limits of integration, `a, b` and the number of subintervals n, each of length $\Delta x = \dfrac{b-a}{n}$, to divide the domain of integration into.
Note:The Trapezoidal Rule is obtained by approximating the integrand *over each sub-interval in x by a straight line, between the left endpoint and the right endpoint of the interval (see fig. on Slide 5b Integration.)*

(i) Test your implementation on the integral above for $n = 2, 10, 50, 250$.

(ii) In each case, record the `Error = True Value - Approximated Value` to 7 places after the decimal.
(iii) Calculate the ratio of the errors for the adjacent values of `n`
(*i.e.* Calculate `Error(2)/Error(10); Error(10)/Error(50)` *etc.*)
What does this tell you about the error?
Summarize your results in a table with columns:
) `n`, `Error Ratio-of-Errors`.

(b) Add a function `midpoint(f, a, b, n)` to `integrate.py`. The Midpoint Rule is given by evaluating the integrand at the *mid-point* of the sub-interval.

$$\int_a^b f(x)dx \approx \Delta x \cdot [f(\frac{x_1 + x_2}{2}) + f(\frac{x_2 + x_3}{2}) + \ldots + f(\frac{x_{n-1} + x_n}{2})]$$

(i) Test your implementation on the integral above for $n = 2, 10, 50, 250$.
(ii) In each case, record the
`Error = True Value - Approximated Value` to 7 places after the decimal.
(iii) Calculate the ratio of the errors for the adjacent values of `n` (*i.e.*. Calculate `Error(2)/Error(10); Error(10)/Error(50)` *etc.*) What does this tell you about the error?

(c) Finally, add a function `main()` to `integrate.py`. This function should test each of the integration routines in the `module` (write the test commands you used above into main). Add the (global) line:
`if __name__ == '__main__':main()`.
This line should be outside the `main()` function definition (it is usually put at the end of the file, outside all function definitions).

You have created your first module. You can now `import` this module into any python programs as you would any other module, and use the functions available within. If you run `integrate.py` it will run the `main()` function within the module.

Note: Next lab, we will continue with integration. In the next lab, you will learn about **vectorizing** your code (this is a way to eliminate loops using `Python & numpy`). You will also implement a **Monte-Carlo** method for integration.