**CST-305: Project 4 – Degradation of Data Integrity**

**Jordan Scott & Angel Velazquez**

**Responsibilities:**

Each team member performed the calculations for both parts, modeling the graphs and finding the equations for each. Additionally, they collaborated on coding tasks and iteratively refined the code to ensure its functionality.

**System Performance Context:**

The software executes by initially solving and graphing the data degradation for both models in part 1 and part w (ODE). Subsequently, it demonstrates data degradation through graphical representation.

**Specific Problem:**

The task involved developing a program capable of solving the ODE segment of the differential equation and generating graphical representations of the results for all ODEs. Furthermore, the program plotted the solution using ODEINT and facilitated comparison with manually conducted work to ensure accuracy.

**Mathematical Approach:**

**Part 1:**

Given three processors, A, B, and C, each containing 100Mb of data, we know that all the processors have 25Mb used for I/O and 75Mb used for internal housekeeping.

Thus, we can say that:

* **x(t)** is the amount of I/O data in processor A at time *t*
* **x(t)** is the amount of I/O data in processor B at time *t*
* **x(t)** is the amount of I/O data in processor C at time *t*

From the provided graph, we can denote the first-order differential equations as follows:

From the equations above, the right-hand side represents the input and output between the processors.

We can create a matrix now that we have our first-order differential equations.

|  | -2  2  0 | 0  -3  1 | 3  0  -1 |  |
| --- | --- | --- | --- | --- |

To find the eigenvalues of matrix A, we find

Multiply first row by -1

|  | 2  2  0 | 0  -3  1 | -3  0  -1 |  |
| --- | --- | --- | --- | --- |

Subtract the 1st row from the 2nd row and restore it

|  | -2  0  0 | 0  -3  1 | -3  3  -1 |  |
| --- | --- | --- | --- | --- |

Divide the 2nd row by -3

|  | -2  0  0 | 0  1  1 | -3  -1  -1 |  |
| --- | --- | --- | --- | --- |

Subtract the 2nd row from the 3rd row

|  | -2  0  0 | 0  1  0 | -3  -1  0 |  |
| --- | --- | --- | --- | --- |

REstore the 2nd row to the original view

|  | -2  0  0 | 0  -3  0 | -3  3  0 |  |
| --- | --- | --- | --- | --- |

Multiply the main diagonal elements

|  | -2  0  0 | 0  -3  0 | -3  3  0 |  |
| --- | --- | --- | --- | --- |

The eigenvalues of the matrix *M* are those numbers within *C* such that *Mv* = *v* for some nonzero vector *v*. Thus, our eigenvalues are

**Part 2:**

In this section, we have two processors each containing 100Mb of memory. Data is transferred only between the two processors.

We can say that:

* **x(t)** is the amount of I/O data in processor A at time *t*
* **x(t)** is the amount of I/O data in processor B at time *t*

Hence, we can model our first-order differential equations as:

From the first-order differential equations, we can create our coefficient matrix:

|  | -2  2 | 3  -3 |  |
| --- | --- | --- | --- |

Now, we can use the matrix exponential method to find the solution to the initial value problem:

|  | 1  -1 |  |  |
| --- | --- | --- | --- |

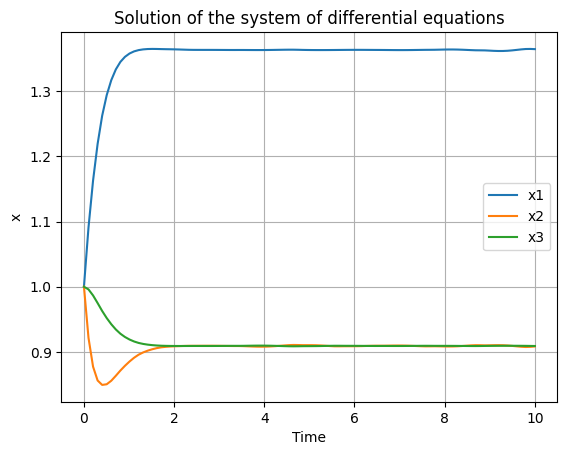
Now, the matrix exponential can be calculated using the power series expansion:

**Implementation of Code:**

**Part1:**

It begins by importing necessary libraries: NumPy for numerical operations, Matplotlib for plotting, and solve\_ivp from SciPy's scipy.integrate module for solving initial value problems. Next we define the system’s matrix A(t) and the function f(t). The differential equation function is defined to represent the system of equations. The time span for which the solution is computed is set to be between 0 and 10. . Initial conditions for the system are specified as x1=1,x2=1,x3=1*and these are arbitrary values.* The solve\_ivp function is then employed to solve the differential equation numerically. The resulting solution is plotted for each variable x1,x2,and x3 over time. The we create the graph/ plot and display the results of the a system of differential equations.

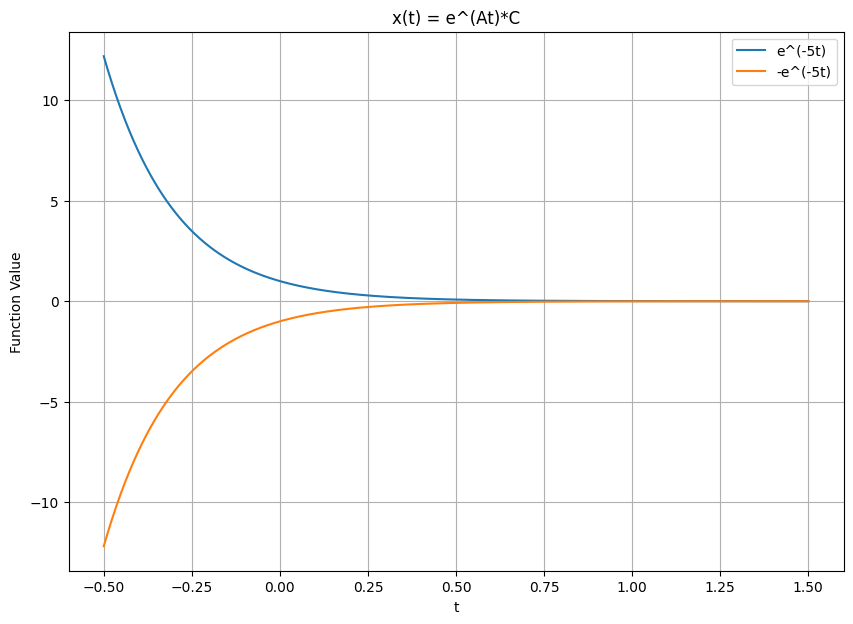
**Screenshots:**

****

**Part2:**

To do this, we need to import libraries such as numpy and matplotlib so that we can display our graph. The graph will display the function x(t)=e^(At) and its negative counterpart −*eAt*, where *A*=−5. It imports necessary libraries such as SciPy's solve\_ivp for solving initial value problems. Next We create an array of our t values. This array will have 1000 evenly spaced values between -0.5 and 1.5 to represent the *t* values. Using Matplotlib, it plots the functions *e*−5*t* and t−*e*−5*t* on the same graph, labeling each for clarity. Axes labels and a title are added for context, and a grid is included for better visualization. Finally, the plot is displayed.

**Screenshots:**



**GithubRepo:** [**https://github.com/angel-vlzqz/Modeling-and-Simulation/tree/main/projects/project%204%20CLC**](https://github.com/angel-vlzqz/Modeling-and-Simulation/tree/main/projects/project%204%20CLC)