**Numerical Analysis Project 3** Fernando Thaureaux: 6236069

**Writing the Code**

-Language: Python

-Input:

- h - the step size, which determines the time intervals at which we find approximate solutions
- t - time points inside the time interval
- initial condition - defining y(0) = 0, the starting value for the diff eq.
- Derivative Func - defining y'(t) = t + y, the differential equation that represents the slope of the solution curve for approximations
- Exact Solution - defining y(t) = $e^t - t - 1$, so we can calculate the true values at t for comparison with approximations

-Output:

Euler's, Modified Euler's, Heun's, and Midpoint methods compute numerical approximations for y(t) at each time point t, based on the given step size h. The true solution is calculated at each iteration for error analysis. For each method and for each iteration, the percentage error is calculated as: |(Approx - Exact)/ Exact| * 100 which quantifies the difference between the approximations and exact solutions.

The table presents t for each step size, the true solutions, the approximate solutions using each method, and the percentage error. The exact values generate a smooth graph, while the numerical solutions are just a set of points connected by a simple line, this is to visualize the comparison between the true vs approximate graph. Euler's method is quick and simple, but prone to error. Modified Euler's, Heun's, and Midpoint methods use additional computations per step to improve precision and accurately approximate the curve.

**Choice of Parameters**

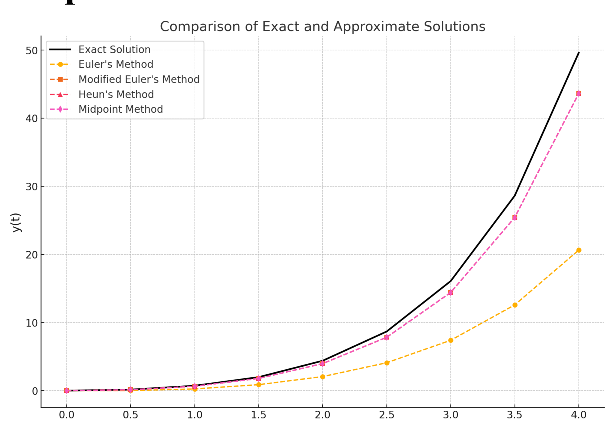
Given h = 0.5, the step size is moderate to limit mesh points, having a balance between accuracy and computational efficiency. A smaller step size would improve accuracy but increase the calculations required.

Given interval: $0 \leq t \leq 4$, we have 8 mesh points, which is a reasonable amount for evaluation without overstepping computational time. Especially given we want to compute these points for multiple methods.

## Results and Analysis

| t | Exact | Euler | Modified Euler | Heun | Midpoint | Euler Error (%) | Modified Euler Error (%) | Heun Error (%) | Midpoint Error (%) |
|---|---|---|---|---|---|---|---|---|---|
| 0.0 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 0.5 | 0.148721 | 0.000000 | 0.125000 | 0.125000 | 0.125000 | 100.000000 | 15.950153 | 15.950153 | 15.950153 |
| 1.0 | 0.718282 | 0.250000 | 0.640625 | 0.640625 | 0.640625 | 65.194720 | 10.811471 | 10.811471 | 10.811471 |
| 1.5 | 1.981689 | 0.875000 | 1.791016 | 1.791016 | 1.791016 | 55.845747 | 9.621764 | 9.621764 | 9.621764 |
| 2.0 | 4.389056 | 2.062500 | 3.972900 | 3.972900 | 3.972900 | 53.008119 | 9.481668 | 9.481668 | 9.481668 |
| 2.5 | 8.682494 | 4.093750 | 7.830963 | 7.830963 | 7.830963 | 52.850529 | 9.807445 | 9.807445 | 9.807445 |
| 3.0 | 16.085537 | 7.390625 | 14.412815 | 14.412815 | 14.412815 | 54.054223 | 10.398918 | 10.398918 | 10.398918 |
| 3.5 | 28.615452 | 12.585938 | 25.420825 | 25.420825 | 25.420825 | 56.016989 | 11.163994 | 11.163994 | 11.163994 |
| 4.0 | 49.598150 | 20.628906 | 43.621340 | 43.621340 | 43.621340 | 58.407912 | 12.050470 | 12.050470 | 12.050470 |

## Graph




As expected, the Euler method being the simplest method is the most inaccurate with the highest percentage error for every step. The other methods are all the same for this problem. They all have noticeably lower percentage errors (as expected) being higher-order while evaluating a linear function for f(t,y). It is worth noting that the code for the Modified Euler's here uses the Predictor-Corrector steps, but testing the code without including those extra steps for correction, it actually becomes the most inaccurate method.

# Code

*Method Comparison:*

```
import numpy as np
import pandas as pd
# Define the derivative function and exact solution
def dydt(t, y):
   return t + y
def exact_solution(t):
   return np.exp(t) - t - 1
# Euler's Method
def euler_method(f, t_values, y0, h):
   y_values = [y0]
   for i in range(len(t_values) - 1):
       y_next = y_values[-1] + h * f(t_values[i], y_values[-1])
       y_values.append(y_next)
   return y_values
# Modified Euler's Method
def modified_euler_method(f, t_values, y0, h):
   y_values = [y0]
   for i in range(len(t_values) - 1):
       # Predictor step
       y_pred = y_values[-1] + h * f(t_values[i], y_values[-1])
       # Corrector step
       y_next = y_values[-1] + (h / 2) * (f(t_values[i], y_values[-1]) + f(t_values[i + 1],
y_pred))
       y_values.append(y_next)
   return y_values
# Heun's Method (equivalent to Modified Euler for this problem)
def heun_method(f, t_values, y0, h):
   y_values = [y0]
   for i in range(len(t_values) - 1):
       y_pred = y_values[-1] + h * f(t_values[i], y_values[-1])
       y_next = y_values[-1] + (h / 2) * (f(t_values[i], y_values[-1]) + f(t_values[i+1],
y_pred))
       y_values.append(y_next)
   return y_values
# Midpoint Method
def midpoint_method(f, t_values, y0, h):
   y_values = [y0]
   for i in range(len(t_values) - 1):
       y_mid = y_values[-1] + (h / 2) * f(t_values[i], y_values[-1])
       y_next = y_values[-1] + h * f(t_values[i] + h / 2, y_mid)
       y_values.append(y_next)
   return y_values
# Define parameters
h = 0.5  # Step size
t_values = np.arange(0, 4.5, h)  # Time points
```

```
y0 = 0  # Initial condition
# Compute solutions
exact_values = exact_solution(t_values)
euler_values = euler_method(dydt, t_values, y0, h)
mod_euler_values = modified_euler_method(dydt, t_values, y0, h)
heun_values = heun_method(dydt, t_values, y0, h)
midpoint_values = midpoint_method(dydt, t_values, y0, h)
# Compute errors
def compute_errors(approx_values, exact_values):
   return [abs((a - e) / e) * 100 if e != 0 else 0 for a, e in zip(approx_values,
exact_values)]
euler_errors = compute_errors(euler_values, exact_values)
mod_euler_errors = compute_errors(mod_euler_values, exact_values)
heun_errors = compute_errors(heun_values, exact_values)
midpoint_errors = compute_errors(midpoint_values, exact_values)
# Create a DataFrame for the results
data = {
   "t": t_values,
   "Exact": exact_values,
   "Euler": euler_values,
   "Modified Euler": mod_euler_values,
   "Heun": heun_values,
   "Midpoint": midpoint_values,
   "Euler Error (%)": euler_errors,
   "Modified Euler Error (%)": mod_euler_errors,
   "Heun Error (%)": heun_errors,
   "Midpoint Error (%)": midpoint_errors,
}
df = pd.DataFrame(data)
# Display the results in a simple table
print(df.to_string(index=False))
```