```python
## ASTE 586 Computer Project
##      Part 1
## Andrew Gerth
## 20250209

import numpy as np
import scipy as sp
from matplotlib import pyplot as plt

# Define Analytical Solution (see paper work)
def x_fun_analytical(t):
    ## Input: time "t"
    ## Output: 4 element tuple of solutions for x(t)
    x1t = np.cos(2*t)
    x2t = -np.sin(2*t)
    x3t = np.cos(200*t)
    x4t = -np.sin(200*t)
    x_t = [x1t, x2t, x3t, x4t]

    return x1t, x2t, x3t, x4t

# Define diff eq system for Numerical Solver in scipy
def dx_fun(t, x):
    A = np.array([[0, 2, 0, 0],
                  [-2, 0, 0, 0],
                  [0, 0, 0, 200],
                  [0, 0, -200, 0]])
    return A @ x

# Define initial values
initial_values = np.array([1, 0, 1, 0])

t = np.linspace(0,10,101) # Define t as a vector
x_analytical = np.zeros((len(t), 4)) # Define empty matrix for x values to go into

# For loop to evaluate x(t) and then assign into x_analytical matrix
for i in range(0, len(t)):
    (x_analytical[i, 0], x_analytical[i, 1], x_analytical[i, 2], x_analytical[i, 3]) = x_fun_analytical(t[i])

# Numerical Solving-time
result = sp.integrate.solve_ivp(dx_fun,
                                     t_span=[t[0], t[len(t)-1]],
                                     y0=initial_values,
                                     t_eval=t,
                                     method='RK45',
                                     rtol=1E-8,
                                     atol=1E-8,
                                     vectorized=True)
#print(x_numerical.y[:, 0])
#print(x_numerical.y)
x_numerical = np.transpose(result.y)

## Calculate Error
# Absolute Error
err_absolute = abs(x_analytical - x_numerical)
#print(err_absolute)
max_err_absolute = np.max(err_absolute, axis=0)
#print(max_err_absolute) # Find max of each column (x1, x2, x3, x4)
polynomial_check = x_numerical[:, 0]**2 + x_numerical[:, 1]**2 + x_numerical[:, 2]**2 + x_numerical[:, 3]**2 - 2
#print(polynomial_check)
```

```python


## Report Results
text_results1 = ('Maximum Error for x1: {:.3e}\n'
                 'Maximum Error for x2: {:.3e}\n'
                 'Maximum Error for x3: {:.3e}\n'
                 'Maximum Error for x4: {:.3e}'.format(max_err_absolute[0], max_err_absolute[1], max_err_absolute[2], max_err_absolute[3]))

text_results2 = r'Tolerance based on $x_1^2(t) + x_2^2(t) + x_3^2(t) + x_4^2(t) - 2$:   ' + '[{:.3e}, {:.3e}]\n'.format(min(polynomial_check), max(polynomial_check))

print(text_results1)
print(text_results2)


## Plotting zone
fig, ax = plt.subplots(4, 1, figsize=(12,16))
fig.suptitle('ASTE 586 Computer Project Part I\n\n* This plot shows the whole range t=[0,10] with a relatively big step size for plot clarity. *')
fig.canvas.manager.set_window_title('Plot 1')

ax[0].plot(t, x_analytical)
ax[1].plot(t, x_numerical)
ax[2].plot(t, err_absolute)
ax[3].plot(t, polynomial_check)

ax[0].set_title('Analytical Solution evaluated from t = [0, {}] with step size: {:.3f}.'.
            format(t[len(t)-1], t[1]))
ax[1].set_title('Numerical Solution evaluated from t = [0, {}] with step size: {:.3f}.'.
            format(t[len(t)-1], t[1]))
ax[2].set_title('Absolute Error evaluated from t = [0, {}] with step size: {:.3f}.'.
            format(t[len(t)-1], t[1]))
ax[3].set_title('Total Error evaluated from t = [0, {}] with step size: {:.3f}.'.
            format(t[len(t)-1], t[1]))

for i in range(0, len(ax)-1):
    ax[i].legend(['$x_1$', '$x_2$', '$x_3$', '$x_4$'])
    ax[i].set_ylabel('x(t)')
for i in range(0, len(ax)):
    #ax[i].set_xlabel('t')
    ax[i].grid(True)
ax[3].set_ylabel(r'$x_1^2(t) + x_2^2(t) + x_3^2(t) + x_4^2(t) - 2$')

ax[3].text(0.025,-0.45, text_results1, transform=ax[3].transAxes, bbox=dict(facecolor='gray', alpha=0.25))
ax[3].text(0.4,-0.35, text_results2, transform=ax[3].transAxes, bbox=dict(facecolor='gray', alpha=0.25))


## Plotting zone (zoomed in with smaller range, smaller step size
del t, x_analytical, x_numerical, result, err_absolute, max_err_absolute, polynomial_check

t = np.linspace(0,10,1000001) # Define t as a vector
x_analytical = np.zeros((len(t), 4)) # Define empty matrix for x values to go into

# For loop to evaluate x(t) and then assign into x_analytical matrix
for i in range(0, len(t)):
    (x_analytical[i, 0], x_analytical[i, 1], x_analytical[i, 2], x_analytical[i, 3]) = x_fun_analytical(t[i])
```

```

# Numerical Solving-time
result = sp.integrate.solve_ivp(dx_fun,
                                     t_span=[t[0], t[len(t)-1]],
                                     y0=initial_values,
                                     t_eval=t,
                                     method='RK45',
                                     rtol=1E-8,
                                     atol=1E-8,
                                     vectorized=True)
#print(x_numerical.y[:, 0])
#print(x_numerical.y)
x_numerical = np.transpose(result.y)

## Calculate Error
# Absolute Error
err_absolute = abs(x_analytical - x_numerical)
#print(err_absolute)
max_err_absolute = np.max(err_absolute, axis=0)
#print(max_err_absolute) # Find max of each column (x1, x2, x3, x4)
polynomial_check = x_numerical[:, 0]**2 + x_numerical[:, 1]**2 + x_numerical[:, 2]**2 +
x_numerical[:, 3]**2 - 2
#print(polynomial_check)


## Report Results
text_results1 = ('Maximum Error for x1: {:.3e}\n'
                 'Maximum Error for x2: {:.3e}\n'
                 'Maximum Error for x3: {:.3e}\n'
                 'Maximum Error for x4: {:.3e}'.format(max_err_absolute[0],
max_err_absolute[1], max_err_absolute[2], max_err_absolute[3]))

text_results2 = r'Tolerance based on $x_1^2(t) + x_2^2(t) + x_3^2(t) + x_4^2(t) - 2$
:   ' + '[{:.3e}, {:.3e}]\n'.format(min(polynomial_check), max(polynomial_check))

print(text_results1)
print(text_results2)


index9 = np.where(t == 9.900)[0][0]
#print(index9)
#print(type(index9))

fig2, ax2 = plt.subplots(4, 1, figsize=(12,16))
fig2.suptitle('ASTE 586 Computer Project Part I\n\n* This plot shows the end of the
evaluated range with a much more discrete step size. *')
fig2.canvas.manager.set_window_title('Plot 2 (Zoomed In)')

ax2[0].plot(t[index9:], x_analytical[index9:, :])
ax2[1].plot(t[index9:], x_numerical[index9:, :])
ax2[2].plot(t[index9:], err_absolute[index9:, :])
ax2[3].plot(t[index9:], polynomial_check[index9:])

ax2[0].set_title('Analytical Solution shown from t = [{}, {}] with step size: {:.3e}.'.
            format(t[index9], t[len(t)-1], t[1]))
ax2[1].set_title('Numerical Solution shown from t = [{}, {}] with step size: {:.3e}.'.
            format(t[index9], t[len(t)-1], t[1]))
ax2[2].set_title('Absolute Error shown from t = [{}, {}] with step size: {:.3e}.'.
            format(t[index9], t[len(t)-1], t[1]))
ax2[3].set_title('Total Error shown from t = [{}, {}] with step size: {:.3e}.'.
            format(t[index9], t[len(t)-1], t[1]))

```

```python
for i in range(0, len(ax2)-1):
    ax2[i].legend(['$x_1$', '$x_2$', '$x_3$', '$x_4$'])
    ax2[i].set_ylabel('x(t)')
for i in range(0, len(ax)):
    #ax2[i].set_xlabel('t')
    ax2[i].grid(True)
ax2[3].set_ylabel(r'$x_1^2(t) + x_2^2(t) + x_3^2(t) + x_4^2(t) - 2$')

ax2[3].text(0.025,-0.45, text_results1, transform=ax2[3].transAxes, bbox=dict(facecolor=
'gray', alpha=0.25))
ax2[3].text(0.4,-0.35, text_results2, transform=ax2[3].transAxes, bbox=dict(facecolor='
gray', alpha=0.25))

fig.savefig("ASTE586_ComputerProject_Part1_Plot1.pdf")
fig2.savefig("ASTE586_ComputerProject_Part1_Plot2.pdf")
plt.show()


```