

ASTE 586 Computer Project Part 1

Andrew Gerth

19 February 2025

1 Introduction

This report details the methodology and solution used to complete the ASTE 586 Spacecraft and Attitude Dynamics Computer Project (Part 1).

The goal of this project was to numerically integrate a given initial value problem and compare the approximated solution to an exact analytical solution.

The initial value problem was given as a linear system of four Ordinary Differential Equations:

$$\vec{x}(t) = \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \\ \dot{x}_4(t) \end{bmatrix} = \begin{bmatrix} 0 & 2 & 0 & 0 \\ -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 200 \\ 0 & 0 & -200 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix}$$

The initial values were given as:

$$\vec{x}(0) = \begin{bmatrix} x_1(0) \\ x_2(0) \\ x_3(0) \\ x_4(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

The numerical and analytical solutions were to be compared to at least $t = 10$, ultimately achieving a maximum allowed difference of 10^{-3} or less for all four components of x .

The final demonstration to assess the accuracy of the numerical solution was to plot $x_1^2(t) + x_2^2(t) + x_3^2(t) + x_4^2(t) - 2$ versus time to demonstrate the result is very close to zero.

2 Analytical Solution

To solve the initial value problem, I recognized it was given in the form $\frac{d}{dt}\vec{x}(t) = [A]\vec{x}(0)$, which has a solution of $\vec{x}(t) = e^{At}\vec{x}(0)$.

To find e^{At} I found the eigenvalues of A by solving $\det(A - \lambda I) = 0$ for λ . This resulted in 4 eigenvalues:

$$\begin{bmatrix} \lambda_{1,2} \\ \lambda_{3,4} \end{bmatrix} = \begin{bmatrix} \pm 2i \\ \pm 200i \end{bmatrix}$$

Next I solved $(A - \lambda_i I)\vec{v}_i = \vec{0}$ to find the eigenvectors corresponding to each eigenvalue.

$$\begin{bmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vec{v}_3 \\ \vec{v}_4 \end{bmatrix} = \begin{bmatrix} [1, i, 0, 0] \\ [1, -i, 0, 0] \\ [0, 0, 1, i] \\ [0, 0, 1, -i] \end{bmatrix}$$

Using the general solution of the form:

$$\vec{x}(t) = c_1 e^{\lambda_1 t} \vec{v}_1 + c_2 e^{\lambda_2 t} \vec{v}_2 + c_3 e^{\lambda_3 t} \vec{v}_3 + c_4 e^{\lambda_4 t} \vec{v}_4$$

Next substitute the corresponding eigenvectors and eigenvalues:

$$\vec{x}(t) = c_1 e^{2it} \begin{bmatrix} 1 \\ i \\ 0 \\ 0 \end{bmatrix} + c_2 e^{-2it} \begin{bmatrix} 1 \\ -i \\ 0 \\ 0 \end{bmatrix} + c_3 e^{200it} \begin{bmatrix} 0 \\ 0 \\ 1 \\ i \end{bmatrix} + c_4 e^{-200it} \begin{bmatrix} 0 \\ 0 \\ 1 \\ -i \end{bmatrix}$$

$$\vec{x}(t) = \begin{bmatrix} c_1 e^{2it} + c_2 e^{-2it} \\ i c_1 e^{2it} - i c_2 e^{-2it} \\ c_3 e^{200it} + c_4 e^{-200it} \\ i c_3 e^{200it} - i c_4 e^{-200it} \end{bmatrix}$$

Use Euler's Formula:

$$e^{i\omega t} = \cos(\omega t) + i \sin(\omega t)$$

$$\vec{x}(t) = \begin{bmatrix} A_1\cos(2t) + A_2\sin(2t) \\ -A_1\sin(2t) + A_2\cos(2t) \\ A_3\cos(200t) + A_4\sin(200t) \\ -A_3\sin(200t) + A_4\cos(200t) \end{bmatrix}$$

where

$$\vec{A} = \begin{bmatrix} c_1 + c_2 \\ i(c_1 - c_2) \\ c_3 + c_4 \\ i(c_3 - c_4) \end{bmatrix}$$

Now to solve for the coefficient vector \vec{A} , evaluate at the initial condition $t = 0$.

$$\vec{A} = [1, 0, 1, 0]$$

This gives the analytical solution to the linear system of equations:

$$\vec{x}(t) = \begin{bmatrix} \cos(2t) \\ -\sin(2t) \\ \cos(200t) \\ -\sin(200t) \end{bmatrix}$$

3 Numerical Integration

My numerical integrations strategy was to use an already-built method of the Python library `scipy`. This allowed me to minimize scripting, debugging, and testing time.

The `scipy.integrate.solve_ivp()` function allows for many different numerical integration methods and tolerances to be used. I chose the Runge-Kutta45 method and specified desired relative and absolute tolerances of 10^{-8} . This combination ensured the solver met the specified accuracy from the problem statement while not being too computationally expensive.

4 Plotting and Comparing Results

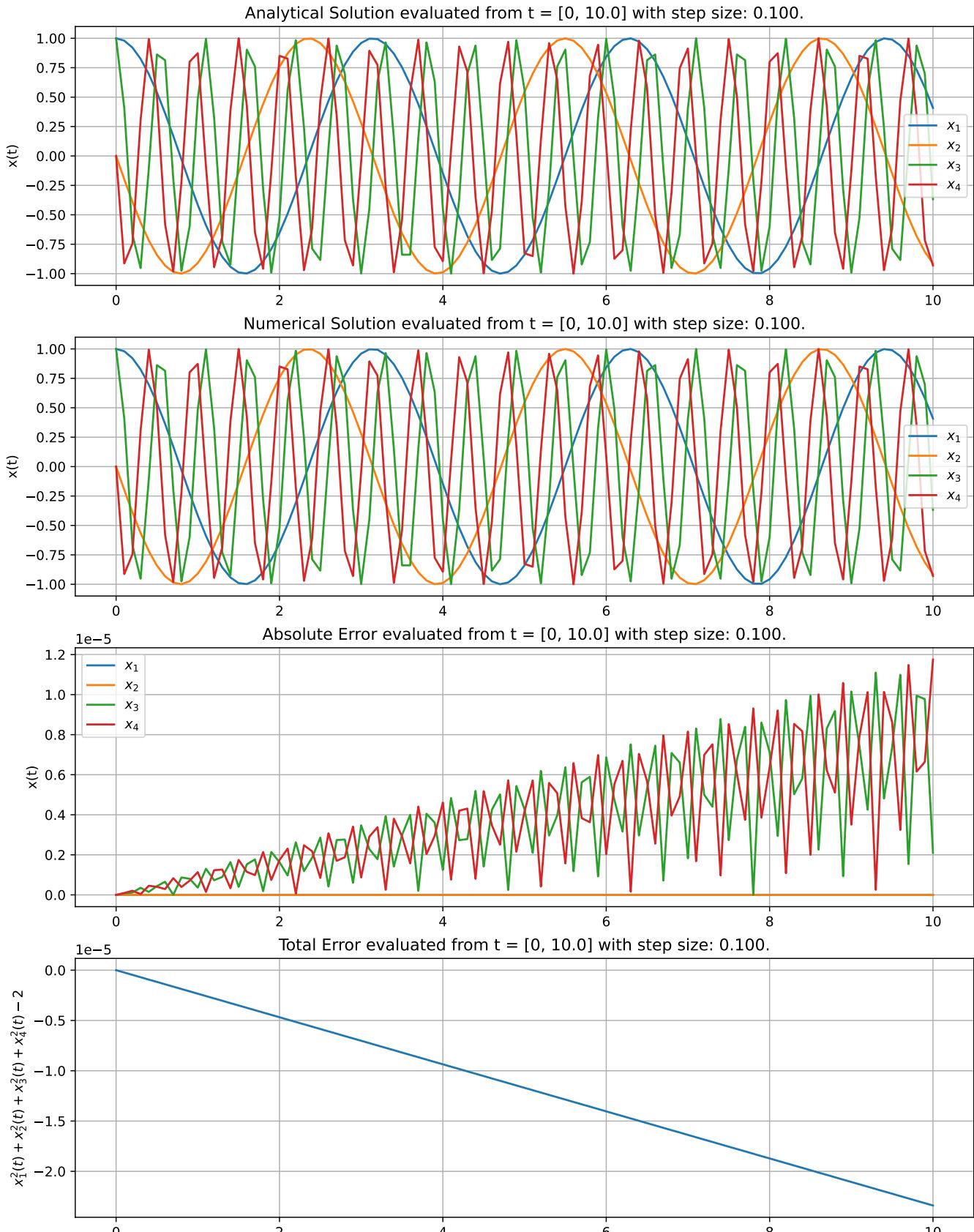
Due to the linear system containing different orders of effects x_1 and x_2 on different order of magnitude than x_3 and x_4 , I chose to create two sets of plots with different "step sizes" for clarity. Figure 1 shows four plots.

1. Analytical Solution evaluated from $t = [0, 10]$ with step size of 0.100.

2. Numerical Solution evaluated from $t = [0, 10]$ with step size of 0.100.
3. Absolute Error (defined as $x_{analytical}(t) - x_{numerical}(t)$) for the same range and step size.
4. The result of $x_1^2(t) + x_2^2(t) + x_3^2(t) + x_4^2(t) - 2$ for the same range and step size.

Figure 1 shows that the numerical solution is within the tolerances specified in the problem statement, but it does not show an accurate representation for the x_3 and x_4 terms due to the relatively large step size.

* This plot shows the whole range $t=[0,10]$ with a relatively big step size for plot clarity. *



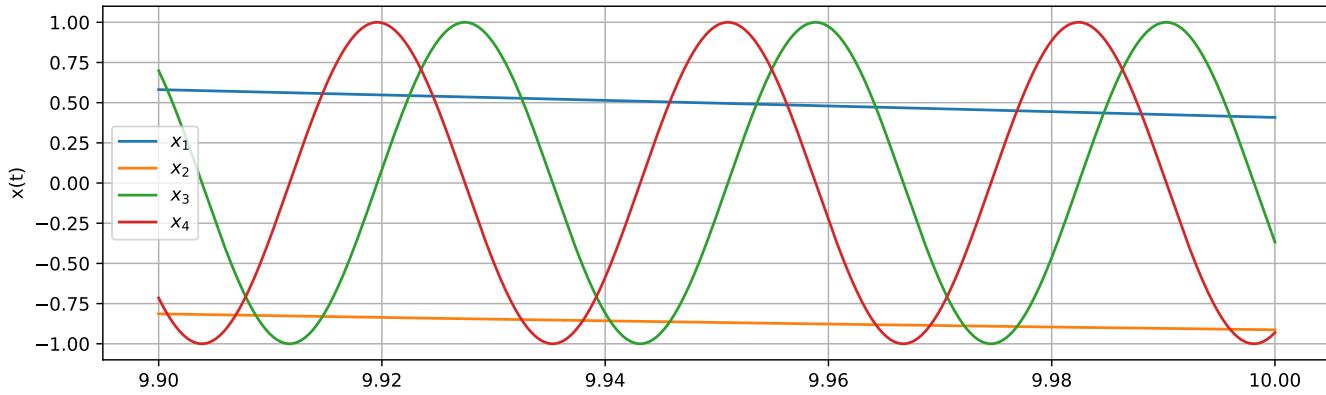
Maximum Error for x1: 7.327e-15
 Maximum Error for x2: 7.438e-15
 Maximum Error for x3: 1.110e-05
 Maximum Error for x4: 1.175e-05

Tolerance based on $x_1^2(t) + x_2^2(t) + x_3^2(t) + x_4^2(t) - 2$: [-2.340e-05, 0.000e+00]

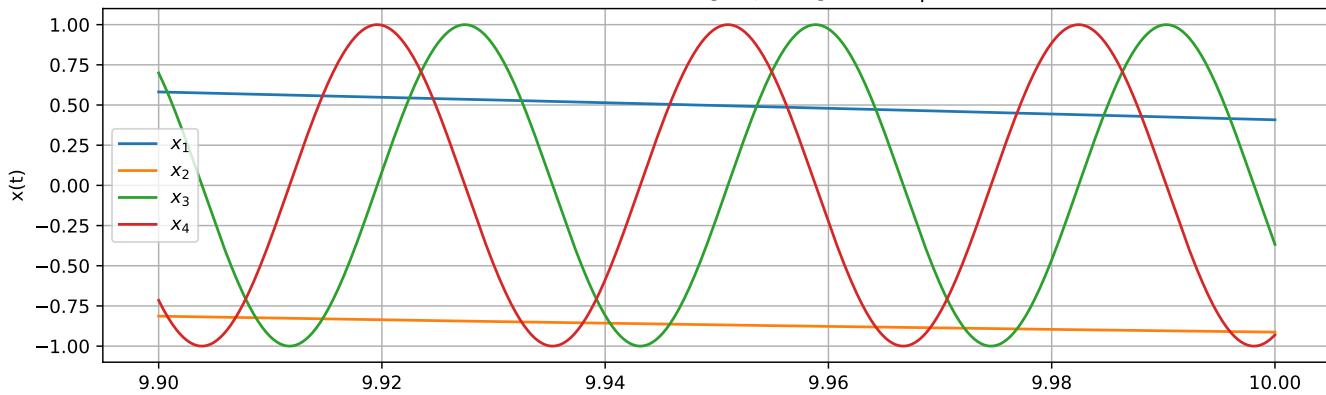
Figure 2 contains the same set of four plots, but evaluated from $t = [0, 10]$ with step size of 0.00001. The plots are then only shown from $t = [9.9, 10.0]$ as this is the most interesting region due to the highest amount of propagation error.

* This plot shows the end of the evaluated range with a much more discrete step size. *

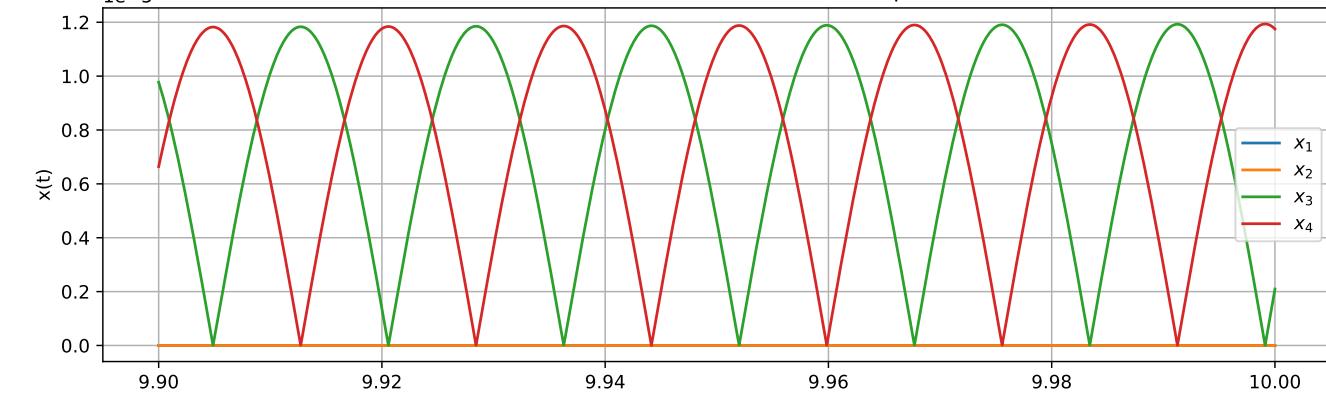
Analytical Solution shown from $t = [9.9, 10.0]$ with step size: $1.000\text{e}-05$.



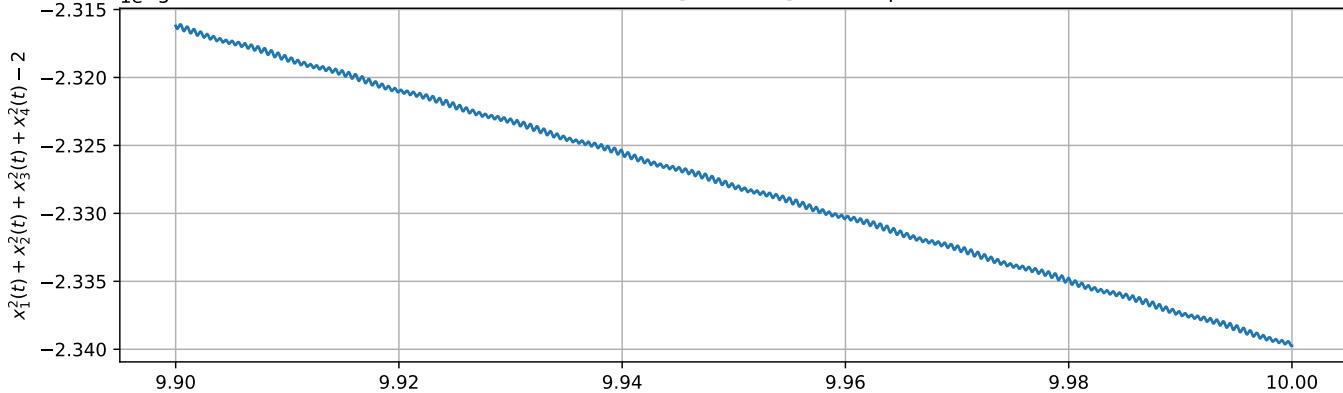
Numerical Solution shown from $t = [9.9, 10.0]$ with step size: $1.000\text{e}-05$.



Absolute Error shown from $t = [9.9, 10.0]$ with step size: $1.000\text{e}-05$.



Total Error shown from $t = [9.9, 10.0]$ with step size: $1.000\text{e}-05$.



Maximum Error for x_1 : $7.550\text{e}-15$
 Maximum Error for x_2 : $7.661\text{e}-15$
 Maximum Error for x_3 : $1.193\text{e}-05$
 Maximum Error for x_4 : $1.194\text{e}-05$

Tolerance based on $x_1^2(t) + x_2^2(t) + x_3^2(t) + x_4^2(t) - 2$: $[-2.340\text{e}-05, 2.346\text{e}-09]$

These four plots show the behavior of x_3 and x_4 much more accurately and serve as a good final confirmation that the goal of the computer project has been satisfied.

The maximum absolute error (*note all maximum errors do not occur at the same value of t):

$$x_1 : 7.550 \times 10^{-15}$$

$$x_2 : 7.661 \times 10^{-15}$$

$$x_3 : 1.193 \times 10^{-05}$$

$$x_4 : 1.194 \times 10^{-05}$$

The tolerance based on $x_1^2(t) + x_2^2(t) + x_3^2(t) + x_4^2(t) - 2$ was:

$$[-2.340 \times 10^{-05}, 2.346 \times 10^{-09}]$$

5 Conclusion

The initial value problem was solved analytically and numerically. The accuracy of the numerical solver met the requirements and is also easily tuned for different required accuracies within the method. For further details on the solver script and the analytical derivation, please see the attachments to this report.

6 Appendix A: Python Script

See following page.

```

1 ## ASTE 586 Computer Project
2 ## Part 1
3 ## Andrew Gerth
4 ## 20250209
5
6 import numpy as np
7 import scipy as sp
8 from matplotlib import pyplot as plt
9
10 # Define Analytical Solution (see paper work)
11 def x_fun_analytical(t):
12     ## Input: time "t"
13     ## Output: 4 element tuple of solutions for x(t)
14     x1t = np.cos(2*t)
15     x2t = -np.sin(2*t)
16     x3t = np.cos(200*t)
17     x4t = -np.sin(200*t)
18     x_t = [x1t, x2t, x3t, x4t]
19
20     return x1t, x2t, x3t, x4t
21
22 # Define diff eq system for Numerical Solver in scipy
23 def dx_fun(t, x):
24     A = np.array([[0, 2, 0, 0],
25                  [-2, 0, 0, 0],
26                  [0, 0, 0, 200],
27                  [0, 0, -200, 0]])
28
29     return A @ x
30
31 # Define initial values
32 initial_values = np.array([1, 0, 1, 0])
33
34 t = np.linspace(0,10,101) # Define t as a vector
35 x_analytical = np.zeros((len(t), 4)) # Define empty matrix for x values to go into
36
37 # For loop to evaluate x(t) and then assign into x_analytical matrix
38 for i in range(0, len(t)):
39     (x_analytical[i, 0], x_analytical[i, 1], x_analytical[i, 2], x_analytical[i, 3]) = x_fun_analytical(t[i])
40
41 # Numerical Solving-time
42 result = sp.integrate.solve_ivp(dx_fun,
43                                 t_span=[t[0], t[len(t)-1]],
44                                 y0=initial_values,
45                                 t_eval=t,
46                                 method='RK45',
47                                 rtol=1E-8,
48                                 atol=1E-8,
49                                 vectorized=True)
50
51 x_numerical = np.transpose(result.y)
52
53 ## Calculate Error
54 # Absolute Error
55 err_absolute = abs(x_analytical - x_numerical)
56 #print(err_absolute)
57 max_err_absolute = np.max(err_absolute, axis=0)
58 #print(max_err_absolute) # Find max of each column (x1, x2, x3, x4)
59 polynomial_check = x_numerical[:, 0]**2 + x_numerical[:, 1]**2 + x_numerical[:, 2]**2 +
60 x_numerical[:, 3]**2 - 2
61 #print(polynomial_check)

```

```

61
62
63 ## Report Results
64 text_results1 = ('Maximum Error for x1: {:.3e}\n'
65             'Maximum Error for x2: {:.3e}\n'
66             'Maximum Error for x3: {:.3e}\n'
67             'Maximum Error for x4: {:.3e}'.format(max_err_absolute[0],
68             max_err_absolute[1], max_err_absolute[2], max_err_absolute[3]))
68
69 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))
70
71 print(text_results1)
72 print(text_results2)
73
74
75 ## Plotting zone
76 fig, ax = plt.subplots(4, 1, figsize=(12,16))
77 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. *')
78 fig.canvas.manager.set_window_title('Plot 1')
79
80 ax[0].plot(t, x_analytical)
81 ax[1].plot(t, x_numerical)
82 ax[2].plot(t, err_absolute)
83 ax[3].plot(t, polynomial_check)
84
85 ax[0].set_title('Analytical Solution evaluated from t = [0, {}] with step size: {:.3f}.'
.
     format(t[len(t)-1], t[1]))
86 ax[1].set_title('Numerical Solution evaluated from t = [0, {}] with step size: {:.3f}.'
.
     format(t[len(t)-1], t[1]))
87 ax[2].set_title('Absolute Error evaluated from t = [0, {}] with step size: {:.3f}.'
.
     format(t[len(t)-1], t[1]))
88 ax[3].set_title('Total Error evaluated from t = [0, {}] with step size: {:.3f}.'
.
     format(t[len(t)-1], t[1]))
89
90
91
92
93
94 for i in range(0, len(ax)-1):
95     ax[i].legend(['$x_1$', '$x_2$', '$x_3$', '$x_4$'])
96     ax[i].set_ylabel('x(t)')
97 for i in range(0, len(ax)):
98     #ax[i].set_xlabel('t')
99     ax[i].grid(True)
100 ax[3].set_ylabel(r'$x_1^2(t) + x_2^2(t) + x_3^2(t) + x_4^2(t) - 2$')
101
102 ax[3].text(0.025,-0.45, text_results1, transform=ax[3].transAxes, bbox=dict(facecolor='
gray', alpha=0.25))
103 ax[3].text(0.4,-0.35, text_results2, transform=ax[3].transAxes, bbox=dict(facecolor='
gray', alpha=0.25))
104
105
106 ## Plotting zone (zoomed in with smaller range, smaller step size
107 del t, x_analytical, x_numerical, result, err_absolute, max_err_absolute,
polynomial_check
108
109 t = np.linspace(0,10,1000001) # Define t as a vector
110 x_analytical = np.zeros((len(t), 4)) # Define empty matrix for x values to go into
111
112 # For loop to evaluate x(t) and then assign into x_analytical matrix
113 for i in range(0, len(t)):
114     (x_analytical[i, 0], x_analytical[i, 1], x_analytical[i, 2], x_analytical[i, 3]) =
x_fun_analytical(t[i])

```

```

115
116 # Numerical Solving-time
117 result = sp.integrate.solve_ivp(dx_fun,
118                                     t_span=[t[0], t[len(t)-1]],
119                                     y0=initial_values,
120                                     t_eval=t,
121                                     method='RK45',
122                                     rtol=1E-8,
123                                     atol=1E-8,
124                                     vectorized=True)
125 #print(x_numerical.y[:, 0])
126 #print(x_numerical.y)
127 x_numerical = np.transpose(result.y)
128
129 ## Calculate Error
130 # Absolute Error
131 err_absolute = abs(x_analytical - x_numerical)
132 #print(err_absolute)
133 max_err_absolute = np.max(err_absolute, axis=0)
134 #print(max_err_absolute) # Find max of each column (x1, x2, x3, x4)
135 polynomial_check = x_numerical[:, 0]**2 + x_numerical[:, 1]**2 + x_numerical[:, 2]**2 +
136     x_numerical[:, 3]**2 - 2
137 #print(polynomial_check)
138
139 ## Report Results
140 text_results1 = ('Maximum Error for x1: {:.3e}\n'
141                  'Maximum Error for x2: {:.3e}\n'
142                  'Maximum Error for x3: {:.3e}\n'
143                  'Maximum Error for x4: {:.3e}'.format(max_err_absolute[0],
144                                              max_err_absolute[1], max_err_absolute[2], max_err_absolute[3]))
145 text_results2 = r'Tolerance based on $x_1^2(t) + x_2^2(t) + x_3^2(t) + x_4^2(t) - 2$  

146 :   ' + '[{:.3e}, {:.3e}]\n'.format(min(polynomial_check), max(polynomial_check))
147 print(text_results1)
148 print(text_results2)
149
150
151 index9 = np.where(t == 9.900)[0][0]
152 #print(index9)
153 #print(type(index9))
154
155 fig2, ax2 = plt.subplots(4, 1, figsize=(12,16))
156 fig2.suptitle('ASTE 586 Computer Project Part I\n* This plot shows the end of the  

157 evaluated range with a much more discrete step size. *')
158 fig2.canvas.manager.set_window_title('Plot 2 (Zoomed In)')
159 ax2[0].plot(t[index9:], x_analytical[index9:, :])
160 ax2[1].plot(t[index9:], x_numerical[index9:, :])
161 ax2[2].plot(t[index9:], err_absolute[index9:, :])
162 ax2[3].plot(t[index9:], polynomial_check[index9:])
163
164 ax2[0].set_title('Analytical Solution shown from t = [{}, {}] with step size: {:.3e}.',
165                   format(t[index9], t[len(t)-1], t[1]))
166 ax2[1].set_title('Numerical Solution shown from t = [{}, {}] with step size: {:.3e}.',
167                   format(t[index9], t[len(t)-1], t[1]))
168 ax2[2].set_title('Absolute Error shown from t = [{}, {}] with step size: {:.3e}.',
169                   format(t[index9], t[len(t)-1], t[1]))
170 ax2[3].set_title('Total Error shown from t = [{}, {}] with step size: {:.3e}.',
171                   format(t[index9], t[len(t)-1], t[1]))
172

```

```
173 for i in range(0, len(ax2)-1):
174     ax2[i].legend(['$x_1$', '$x_2$', '$x_3$', '$x_4$'])
175     ax2[i].set_ylabel('x(t)')
176 for i in range(0, len(ax)):
177     #ax2[i].set_xlabel('t')
178     ax2[i].grid(True)
179 ax2[3].set_ylabel(r'$x_1^2(t) + x_2^2(t) + x_3^2(t) + x_4^2(t) - 2$')
180
181 ax2[3].text(0.025,-0.45, text_results1, transform=ax2[3].transAxes, bbox=dict(facecolor=
182 'gray', alpha=0.25))
183 ax2[3].text(0.4,-0.35, text_results2, transform=ax2[3].transAxes, bbox=dict(facecolor='
184 gray', alpha=0.25))
185 fig.savefig("ASTE586_ComputerProject_Part1_Plot1.pdf")
186 fig2.savefig("ASTE586_ComputerProject_Part1_Plot2.pdf")
187 plt.show()
188
```

7 Appendix B: Detailed Derivation of Analytical Solution

See following page.

diff eq's:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \\ \dot{x}_4(t) \end{bmatrix} = \begin{bmatrix} 0 & 2 & 0 & 0 \\ -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 200 \\ 0 & 0 & -200 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix}$$

initial cond. $\vec{x}(0) = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$

given in form $\frac{d}{dt} \vec{x}(t) = [A] \vec{x}(t)$

use matrix exponential: $\vec{x}(t) = e^{At} \vec{x}(0)$

find e^{At} by finding eig vals of A. $\det(A - \lambda I) = 0$

$$\det \begin{bmatrix} -\lambda & 2 & 0 & 0 \\ -2 & -\lambda & 0 & 0 \\ 0 & 0 & -\lambda & 200 \\ 0 & 0 & -200 & -\lambda \end{bmatrix} = 0 \Rightarrow \cancel{\det} = \cancel{\begin{vmatrix} 2 & 0 & 0 \\ 0 & -\lambda & 200 \\ 0 & -200 & -\lambda \end{vmatrix}} = \cancel{\begin{vmatrix} 2 & 0 & 0 \\ 0 & -\lambda^2 + 40000 & 0 \\ 0 & 0 & -\lambda \end{vmatrix}}$$

$$\Rightarrow -\lambda \begin{vmatrix} -\lambda & 0 & 0 \\ 0 & -\lambda & 200 \\ 0 & -200 & -\lambda \end{vmatrix} + 2 \begin{vmatrix} 2 & 0 & 0 \\ 0 & -\lambda & 200 \\ 0 & -200 & -\lambda \end{vmatrix}$$

$$-\lambda \left[-\lambda \left[\lambda^2 + 40000 \right] \right] + 2 \left[2 \left[\lambda^2 + 40000 \right] \right] = \lambda^4 + 40000\lambda^2 + 4\lambda^2 + 80000 = 0$$

$$\lambda^4 + 40004\lambda^2 + 80000 = 0$$

$$\lambda_{1,2}^* = \pm(2i)$$

$$\lambda_{3,4}^* = \pm(200i)$$

for $\lambda_1 = 2i$ solve $(A - 2iI)\vec{x} = \vec{0} \Rightarrow \begin{bmatrix} -2i & 2 & 0 & 0 \\ -2 & -2i & 0 & 0 \\ 0 & 0 & -2i & 200 \\ 0 & 0 & -200 & -2i \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

$$-2ix_1 + 2x_2 = 0 \Rightarrow x_2 = ix_1$$

$$-2ix_3 + 200x_4 = 0 \Rightarrow x_4 = \frac{ix_3}{100} \quad \text{choose } x_1 = 1, x_3 = 0$$

$$\vec{v}_1 = \begin{bmatrix} 1 \\ i \\ 0 \\ 0 \end{bmatrix}$$

for $\lambda_2 = -2i$ $(A + 2iI) \vec{v}_2 = \vec{0}$

$$\begin{bmatrix} 2i & 2 & 0 & 0 \\ -2 & 2i & 0 & 0 \\ 0 & 0 & 2i & 200 \\ 0 & 0 & -200 & 2i \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$x_2 = -ix_1$$

$$x_4 = -\frac{ix_3}{100}$$

$$\vec{v}_2 = \begin{bmatrix} 1 \\ -i \\ 0 \\ 0 \end{bmatrix}$$

for $\lambda_3 = +200$:

$$\vec{v}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ i \end{bmatrix}, \vec{v}_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -i \end{bmatrix}$$

$$\begin{bmatrix} -200i & 2 & 0 & 0 \\ -2 & -200i & 0 & 0 \\ 0 & 0 & -200i & 200 \\ 0 & 0 & -200 & -200i \end{bmatrix}$$

General Sol'n:

$$x(t) = c_1 e^{\lambda_1 t} \vec{v}_1 + c_2 e^{\lambda_2 t} \vec{v}_2 + c_3 e^{\lambda_3 t} \vec{v}_3 + c_4 e^{\lambda_4 t} \vec{v}_4$$

Sub $\vec{\lambda} \notin \vec{v}$

$$x(t) = c_1 e^{2it} \begin{bmatrix} 1 \\ i \\ 0 \\ 0 \end{bmatrix} + c_2 e^{-2it} \begin{bmatrix} 1 \\ -i \\ 0 \\ 0 \end{bmatrix} + c_3 e^{200it} \begin{bmatrix} 0 \\ 0 \\ 1 \\ i \end{bmatrix} + c_4 e^{-200it} \begin{bmatrix} 0 \\ 0 \\ 0 \\ -i \end{bmatrix}$$

$$\vec{x}(t) = \begin{bmatrix} c_1 e^{2it} + c_2 e^{-2it} \\ i c_1 e^{2it} - i c_2 e^{-2it} \\ c_3 e^{200it} + c_4 e^{-200it} \\ i c_3 e^{200it} - i c_4 e^{-200it} \end{bmatrix}$$

use euler's formula $e^{iwt} = \cos(wt) + i \sin(wt)$

$$x_1(t) = c_1 [\cos(2t) + i \sin(2t)] + c_2 [\cos(-2t) + i \sin(-2t)]$$

$$x_2(t) = i c_1 [\cos(2t) + i \sin(2t)] \neq i c_2 [\cos(-2t) + i \sin(-2t)]$$

$$\begin{aligned} x_3(t) &= c_1 \cos(2t) + c_1 i \sin(2t) + c_2 \cos(2t) - c_2 i \sin(2t) \\ &= \cos(2t) [c_1 + c_2] + i \sin(2t) [c_1 - c_2] \end{aligned}$$

call $c_1 + c_2 \equiv A_1$ & $i[c_1 - c_2] \equiv A_2$

$$x_3(t) = i c_1 \cos(2t) + i^2 c_1 \sin(2t) - i c_2 \cos(2t) + i^2 c_2 \sin(2t)$$

$$x_3(t) = \cos(2t) [i c_1 - i c_2] + \sin(2t) [c_1 + c_2]$$

* same $A_1 \neq A_2$

→ continue on next page.

In terms of λ_1 & λ_2

$$x_1(t) = A_1 \cos(2t) + A_2 \sin(2t) \quad \dots \text{same pattern for } \lambda_3 \text{ & } \lambda_4$$

$$x_2(t) = -A_1 \sin(2t) + A_2 \cos(2t)$$

$$x_3(t) = A_3 \cos(200t) + A_4 \sin(200t)$$

$$x_4(t) = -A_3 \sin(200t) + A_4 \cos(200t)$$

bring back initial cond. $\vec{x}(0) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$

$$t=0$$

$$1 = A_1(1) + 0$$

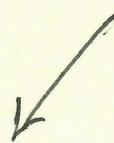
$$0 = A_2(1)$$

$$1 = A_3(1) + 0$$

$$0 = A_4(1)$$

$$\vec{A} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

now plug back into $\vec{x}(t)$



$$\boxed{\vec{x}(t) = \begin{bmatrix} \cos(2t) \\ -\sin(2t) \\ \cos(200t) \\ -\sin(200t) \end{bmatrix}}$$