

```

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     return A @ x
29
30 # Define initial values
31 initial_values = np.array([1, 0, 1, 0])
32
33 t = np.linspace(0,10,101) # Define t as a vector
34 x_analytical = np.zeros((len(t), 4)) # Define empty matrix for x values to go into
35
36 # For loop to evaluate x(t) and then assign into x_analytical matrix
37 for i in range(0, len(t)):
38     (x_analytical[i, 0], x_analytical[i, 1], x_analytical[i, 2], x_analytical[i, 3]) =
39     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 #print(x_numerical.y[:, 0])
51 #print(x_numerical.y)
52 x_numerical = np.transpose(result.y)
53
54 ## Calculate Error
55 # Absolute Error
56 err_absolute = abs(x_analytical - x_numerical)
57 #print(err_absolute)
58 max_err_absolute = np.max(err_absolute, axis=0)
59 #print(max_err_absolute) # Find max of each column (x1, x2, x3, x4)
60 polynomial_check = x_numerical[:, 0]**2 + x_numerical[:, 1]**2 + x_numerical[:, 2]**2 +
61 x_numerical[:, 3]**2 - 2
62 #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]))
69 text_results2 = r'Tolerance based on $x_1^2(t) + x_2^2(t) + x_3^2(t) + x_4^2(t) - 2$
70 : ' + '[{:.3e}, {:.3e}]\n'.format(min(polynomial_check), max(polynomial_check))
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
78 ,10] with a relatively big step size for plot clarity. *')
79 fig.canvas.manager.set_window_title('Plot 1')
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}.'.
86                 format(t[len(t)-1], t[1]))
87 ax[1].set_title('Numerical Solution evaluated from t = [0, {}] with step size: {:.3f}.'.
88                 format(t[len(t)-1], t[1]))
89 ax[2].set_title('Absolute Error evaluated from t = [0, {}] with step size: {:.3f}.'.
90                 format(t[len(t)-1], t[1]))
91 ax[3].set_title('Total Error evaluated from t = [0, {}] with step size: {:.3f}.'.
92                 format(t[len(t)-1], t[1]))
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='
103 gray', alpha=0.25))
104 ax[3].text(0.4,-0.35, text_results2, transform=ax[3].transAxes, bbox=dict(facecolor='
105 gray', alpha=0.25))
106
107 ## Plotting zone (zoomed in with smaller range, smaller step size)
108 del t, x_analytical, x_numerical, result, err_absolute, max_err_absolute,
109 polynomial_check
110 t = np.linspace(0,10,1000001) # Define t as a vector
111 x_analytical = np.zeros((len(t), 4)) # Define empty matrix for x values to go into
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]) =
115     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\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
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=
    'gray', alpha=0.25))
182 ax2[3].text(0.4,-0.35, text_results2, transform=ax2[3].transAxes, bbox=dict(facecolor='
    gray', alpha=0.25))
183
184 fig.savefig("ASTE586_ComputerProject_Part1_Plot1.pdf")
185 fig2.savefig("ASTE586_ComputerProject_Part1_Plot2.pdf")
186 plt.show()
187
188
```