

ANSWERS: EXERCISE SHEET No. 6

ODE Pt. II.

1 Basic theory and manipulations

2 asod

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Thu Feb 17 12:23:21 2022
4
5 @author: P. Maxwell
6 """
7
8
9 import numpy as np
10 import matplotlib as mpl
11 import matplotlib.pyplot as plt
12 from scipy.integrate import solve_ivp
13
14
15
16
17 def f_lorenz(t, w, sigma, beta, rho):
18     """Lorenz differential equations"""
19     x = w[0]
20     y = w[1]
21     z = w[2]
22     return np.array([ sigma*(y-x), x*(rho-z) - y, x*y - beta*z ],
23                     dtype=np.float64 )
24
25
26 # Parameters in order sigma, beta, rho.
27 param_args = (10.0, 8/3, 28)
28
29 # Initial point
30 x0 = np.array([5.0, 5.0, 5.0])
31
32 # Time interval to integrate over
33 tspan = [0, 60]
34
35 # Set tolerances
36 reltol = 1e-12
37 abstol = 1e-14
38
39
40
41
42 def do_solve(f, tspan, x0, abstol, reltol, param_args):
43     """Use SciPy to solve system explicitly with 8th order RK"""
44     sol = solve_ivp(f, tspan, x0,
45                     args=param_args,
46                     method='DOP853', dense_output=True,
47                     rtol=reltol, atol=abstol)
48     return sol
49
50
```

```

51
52
53
54 def plot_lorenz(sol_lorenz, t_pts):
55     # Plot using a 2x2 subplot figure, with t-x, t-y, t-z, and 3d x
    -y-z.
56     fig = plt.figure(figsize=(30, 30))
57     ax1 = fig.add_subplot(221)
58     ax1.plot(t_pts, sol_lorenz.sol(t_pts)[0,:], 'b-', linewidth
    =1.0)
59     ax1.tick_params(axis='both', labelsz=22)
60     ax1.set_xlabel(r'$t$', fontsize=28)
61     ax1.set_ylabel(r'$x$', fontsize=28)
62
63     ax2 = fig.add_subplot(222)
64     ax2.plot(t_pts, sol_lorenz.sol(t_pts)[1,:], 'b-', linewidth
    =1.0)
65     ax2.tick_params(axis='both', labelsz=22)
66     ax2.set_xlabel(r'$t$', fontsize=28)
67     ax2.set_ylabel(r'$y$', fontsize=28)
68
69     ax3 = fig.add_subplot(223)
70     ax3.plot(t_pts, sol_lorenz.sol(t_pts)[2,:], 'b-', linewidth
    =1.0)
71     ax3.tick_params(axis='both', labelsz=22)
72     ax3.set_xlabel(r'$t$', fontsize=28)
73     ax3.set_ylabel(r'$z$', fontsize=28)
74
75     ax4 = fig.add_subplot(224, projection='3d')
76     ax4.plot3D(sol_lorenz.sol(t_pts)[0,:], sol_lorenz.sol(t_pts)
    [1,:], sol_lorenz.sol(t_pts)[2,:], 'b-', linewidth=1.0)
77     ax4.tick_params(axis='both', labelsz=22)
78
79
80
81
82
83 sol_lorenz = do_solve(f_lorenz, tspan, x0, abstol, reltol,
    param_args)
84
85 t_pts = np.linspace(tspan[0], tspan[1], 10000)
86 plot_lorenz(sol_lorenz, t_pts)

```

```

1     # -*- coding: utf-8 -*-
2     """
3     Created on Mon Feb 21 03:41:46 2022
4
5     @author: P. Maxwell
6     """
7
8     import autograd.numpy as np
9     from autograd import jacobian
10    from autograd import grad
11    import matplotlib as mpl
12    import matplotlib.pyplot as plt
13    from scipy.integrate import solve_ivp
14
15
16
17
18 # Parameters in order sigma, beta, rho.
19 param_args = ()
20

```

```

21 # Initial point
22 x0 = np.array([1.0, 0.0, 0.0])
23
24 # Time interval to integrate over
25 tspan = np.array([0.0, 10.0])
26
27 # Set tolerances
28 reltol = 1e-4
29 abstol = 1e-7
30
31
32
33
34 def f_robertson(t, w):
35     """Simplified Brusselator differential equations"""
36     y1 = w[0]
37     y2 = w[1]
38     y3 = w[2]
39     return np.array([ -0.04*y1 + 1e4*y2*y3,
40                       0.04*y1 - 1e4*y2*y3 - 3e7*y2**2,
41                       3e7*y2**2 ], dtype=np.float64 )
42
43
44
45 # Jacobian (used for the Newton iteration of an implicit method
46   like RADAU)
47 jac_robertson_autograd = jacobian(f_robertson, 1)
48 def jac_robertson(t, x):
49     return jac_robertson_autograd(t, x)
50
51
52
53 def do_solve_explicit(f, tspan, x0, abstol, reltol, param_args):
54     """Use SciPy to solve system explicitly with RK45"""
55     sol = solve_ivp(f, tspan, x0,
56                     args=param_args,
57                     method='RK45', dense_output=True,
58                     rtol=reltol, atol=abstol)
59     return sol
60
61
62
63
64 def do_solve_implicit(f, tspan, x0, abstol, reltol, param_args, jac
65 ):
66     """Use SciPy to solve system explicitly with Radau"""
67     sol = solve_ivp(f, tspan, x0,
68                     args=param_args,
69                     method='Radau', dense_output=True,
70                     jac=jac,
71                     rtol=reltol, atol=abstol)
72     return sol
73
74
75
76 def plot_robertson(sol_robertson_explicit, sol_robertson_implicit,
77 t_pts):
78     fig = plt.figure(figsize=(40, 40))
79     ax1 = fig.add_subplot(321)
80     ax1.semilogx(t_pts, sol_robertson_explicit.sol(t_pts)[0,:], 'b

```

```

- ', linewidth=3)
80 ax1.semilogx(sol_robertson_explicit.t, sol_robertson_explicit.y
[0,:], 'ro', linewidth=3, markersize=15)
81 ax1.grid()
82 ax1.tick_params(axis='both', labelsz=22)
83 ax1.set_title(r'Robertson $y_1$ (explicit)', fontsize=28)
84 ax1.set_xlabel(r'$t$', fontsize=28)
85 ax1.set_ylabel(r'$y_1$', fontsize=28)
86
87
88 ax2 = fig.add_subplot(323)
89 ax2.semilogx(t_pts, sol_robertson_explicit.sol(t_pts)[1,:], 'b
-', linewidth=3)
90 ax2.semilogx(sol_robertson_explicit.t, sol_robertson_explicit.y
[1,:], 'ro', linewidth=3, markersize=15)
91 ax2.grid()
92 ax2.tick_params(axis='both', labelsz=22)
93 ax2.set_title(r'Robertson $y_2$ (explicit)', fontsize=28)
94 ax2.set_xlabel(r'$t$', fontsize=28)
95 ax2.set_ylabel(r'$y_2$', fontsize=28)
96
97 ax3 = fig.add_subplot(325)
98 ax3.semilogx(t_pts, sol_robertson_explicit.sol(t_pts)[2,:], 'b
-', linewidth=3)
99 ax3.semilogx(sol_robertson_explicit.t, sol_robertson_explicit.y
[2,:], 'ro', linewidth=3, markersize=15)
100 ax3.grid()
101 ax3.tick_params(axis='both', labelsz=22)
102 ax3.set_title(r'Robertson $y_3$ (explicit)', fontsize=28)
103 ax3.set_xlabel(r'$t$', fontsize=28)
104 ax3.set_ylabel(r'$y_3$', fontsize=28)
105
106
107 ax4 = fig.add_subplot(322)
108 ax4.semilogx(t_pts, sol_robertson_implicit.sol(t_pts)[0,:], 'b
-', linewidth=3)
109 ax4.semilogx(sol_robertson_implicit.t, sol_robertson_implicit.y
[0,:], 'ro', linewidth=3, markersize=15)
110 ax4.grid()
111 ax4.tick_params(axis='both', labelsz=22)
112 ax4.set_title(r'Robertson $y_1$ (implicit)', fontsize=28)
113 ax4.set_xlabel(r'$t$', fontsize=28)
114 ax4.set_ylabel(r'$y_1$', fontsize=28)
115
116 ax5 = fig.add_subplot(324)
117 ax5.semilogx(t_pts, sol_robertson_implicit.sol(t_pts)[1,:], 'b
-', linewidth=3)
118 ax5.semilogx(sol_robertson_implicit.t, sol_robertson_implicit.y
[1,:], 'ro', linewidth=3, markersize=15)
119 ax5.grid()
120 ax5.tick_params(axis='both', labelsz=22)
121 ax5.set_title(r'Robertson $y_2$ (implicit)', fontsize=28)
122 ax5.set_xlabel(r'$t$', fontsize=28)
123 ax5.set_ylabel(r'$y_2$', fontsize=28)
124
125 ax6 = fig.add_subplot(326)
126 ax6.semilogx(t_pts, sol_robertson_implicit.sol(t_pts)[2,:], 'b
-', linewidth=3)
127 ax6.semilogx(sol_robertson_implicit.t, sol_robertson_implicit.y
[2,:], 'ro', linewidth=3, markersize=15)
128 ax6.grid()
129 ax6.tick_params(axis='both', labelsz=22)

```

```

130     ax6.set_title(r'Robertson $y_3$ (implicit)', fontsize=28)
131     ax6.set_xlabel(r'$t$', fontsize=28)
132     ax6.set_ylabel(r'$y_3$', fontsize=28)
133
134
135
136
137 sol_robertson_explicit = do_solve_explicit(f_robertson, tspan, x0,
138     abstol, reltol, param_args)
139 sol_robertson_implicit = do_solve_implicit(f_robertson, tspan, x0,
140     abstol, reltol, param_args, jac_robertson)
141
142 # t_pts = np.linspace(tspan[0], tspan[1], 1000)
143 t_pts = 10*np.linspace(-6, np.log10(tspan[1]), 1000)
144 plot_robertson(sol_robertson_explicit, sol_robertson_implicit,
145     t_pts)

```

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Feb 21 03:19:20 2022
4
5  @author: P. Maxwell
6  """
7
8  import numpy as np
9  import matplotlib as mpl
10 import matplotlib.pyplot as plt
11 from scipy.integrate import solve_ivp
12
13
14
15
16 # Parameters in order sigma, beta, rho.
17 param_args = (1.0, 3.0)
18
19 # Initial point
20 x0 = np.array([1.0, 3.08])
21
22 # Time interval to integrate over
23 tspan = [0.0, 15.0]
24
25 # Set tolerances
26 reltol = 1e-6
27 abstol = 1e-9
28
29
30
31
32 def f_brusselator_simple(t, w, A, B):
33     """Simplified Brusselator differential equations"""
34     y1 = w[0]
35     y2 = w[1]
36     return np.array([ A + y2*y1**2 -(B+1)*y1, B*y1 -y2*y1**2 ],
37         dtype=np.float64 )
38
39
40
41 def do_solve(f, tspan, x0, abstol, reltol, param_args):
42     """Use SciPy to solve system explicitly with RK45"""
43     sol = solve_ivp(f, tspan, x0,
44         args=param_args,

```

```

45         method='RK45', dense_output=True,
46         rtol=reltol, atol=abstol)
47     return sol
48
49
50
51 def plot_brusselator_simple(sol_brusselator_simple, t_pts):
52     fig, ax = plt.subplots(1, 1, figsize=(20, 20))
53     ax.plot(sol_brusselator_simple.sol(t_pts)[0,:],
54            sol_brusselator_simple.sol(t_pts)[1,:], linewidth=3)
55     ax.grid()
56     ax.tick_params(axis='both', labelsize=22)
57     ax.set_title(r'Brusselator Simple', fontsize=28)
58     ax.set_xlabel(r'$y_1$', fontsize=28)
59     ax.set_ylabel(r'$y_2$', fontsize=28)
60
61
62 sol_brusselator_simple = do_solve(f_brusselator_simple, tspan, x0,
63                                  abstol, reltol, param_args)
64
65 t_pts = np.linspace(tspan[0], tspan[1], 1000)
66 plot_brusselator_simple(sol_brusselator_simple, t_pts)

```

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Sun Feb 20 21:23:11 2022
4
5  @author: P. Maxwell
6  """
7
8
9  import autograd.numpy as np
10 from autograd import jacobian
11 from autograd import grad
12 import matplotlib as mpl
13 import matplotlib.pyplot as plt
14 #import scipy.optimize as sp0
15 from scipy.integrate import solve_ivp
16
17
18 # Remember to note that this is a continuous reformulation of a
19   discrete process
20
21
22
23 N = 1000000
24 Ethres = N * 0.2
25 R0 = 4.0
26 gamma = 1 / 5
27 beta_sir = R0 * gamma
28 beta_seirsd = R0 * (gamma + alpha1)
29 alpha1 = 0.015
30 # alpha2 = 0.03
31 alpha2 = 0.03
32 sigma = 1/5 # Original Covid
33 omega = 1/180
34
35
36 initial_seed = 10.0
37 w0_sir = np.array([N-initial_seed, initial_seed, 0.0])

```

```

38 w0_seirsd = np.array([N-initial_seed, initial_seed, 0.0, 0.0, 0.0])
39
40
41
42 param_args_sir = np.array([N, beta_sir, gamma])
43 param_args_seirsd = np.array([N, alpha1, alpha2, beta_seirsd, gamma
44 , sigma, omega, Ethres])
45 tspan = (0.0, 400.0)
46 reltol = 1e-4
47 abstol = 1e-7
48
49 def f_sir(t, w, N, beta, gamma):
50     """Calculate derivatives in SIR model"""
51     S = w[0]
52     I = w[1]
53     R = w[2]
54
55     dwdt = np.zeros((3,), dtype=np.float64)
56     # dS/dt
57     dwdt[0] = -beta*I*S/N
58     # dI/dt
59     dwdt[1] = beta*I*S/N - gamma*I
60     # dR/dt
61     dwdt[2] = gamma*I
62
63
64     if abs((S+I+R-N)) > 1.0:
65         print("Consistency error! S, I, R, N, diff:", S, I, R, N,
66               abs((S+I+R-N)))
67
68     return dwdt
69
70
71
72
73
74
75 def f_seirsd(t, w, N, alpha1, alpha2, beta, gamma, sigma, omega,
76             Ethres):
77     """Calculate derivatives in simple SEIRSD model."""
78     S = w[0]
79     E = w[1]
80     I = w[2]
81     R = w[3]
82     D = w[4]
83     alpha1_cntr = np.max(np.array([0.0, (alpha1*N - D)/N])) * I
84     alpha2_cntr = np.max(np.array([0.0, (alpha2*N - D)/N])) * np.
85     max(np.array([0.0, I-Ethres]))
86     dwdt = np.zeros((5,), dtype=np.float64)
87     # dS/dt
88     dwdt[0] = -beta*I*S/N + omega*R
89     # dE/dt
90     dwdt[1] = beta*I*S/N - sigma*E
91     # dI/dt
92     dwdt[2] = sigma*E - gamma*I - alpha1_cntr - alpha2_cntr
93     # dR/dt
94     dwdt[3] = gamma*I - omega*R
95     # dD/dt
96     dwdt[4] = alpha1_cntr + alpha2_cntr

```

```

96     return dwdt
97
98
99
100
101
102
103 def do_solve_cm(f, tspan, w0, abstol, reltol, param_args):
104     # Do the integration using explicit 8th order RK
105     sol_cm = solve_ivp(f, tspan, w0,
106                        args=param_args,
107                        method='RK45', dense_output=True,
108                        rtol=reltol, atol=abstol)
109     return sol_cm
110
111
112
113
114
115
116
117 def plot_solution_sir(sol_sir, end_day):
118     t_pts = np.float64(np.arange(0, end_day+1))
119     fig, ax = plt.subplots(1, 1, figsize=(30, 15))
120     ax.plot(t_pts, sol_sir.sol(t_pts)[0, :], label='Susceptible',
121            linewidth=3)
122     ax.plot(t_pts, sol_sir.sol(t_pts)[1, :], label='Infectious',
123            linewidth=3)
124     ax.plot(t_pts, sol_sir.sol(t_pts)[2, :], label='Recovered',
125            linewidth=3)
126     ax.grid()
127     ax.tick_params(axis='both', labelsize=22)
128     ax.legend(fontsize=28)
129     ax.set_title(r'SIR model output', fontsize=28)
130     ax.set_xlabel(r'$t$ (days)', fontsize=28)
131     ax.set_ylabel(r'Individuals (millions)', fontsize=28)
132
133
134 def plot_solution_seirsd(sol_seirsd, end_day):
135     t_pts = np.float64(np.arange(0, end_day+1))
136     fig, ax = plt.subplots(1, 1, figsize=(30, 15))
137     ax.plot(t_pts, sol_seirsd.sol(t_pts)[0, :], label='Susceptible',
138            linewidth=3)
139     ax.plot(t_pts, sol_seirsd.sol(t_pts)[1, :], label='Exposed',
140            linewidth=3)
141     ax.plot(t_pts, sol_seirsd.sol(t_pts)[2, :], label='Infectious',
142            linewidth=3)
143     ax.plot(t_pts, sol_seirsd.sol(t_pts)[3, :], label='Recovered',
144            linewidth=3)
145     ax.plot(t_pts, sol_seirsd.sol(t_pts)[4, :], label='Dead',
146            linewidth=3)
147     ax.grid()
148     ax.tick_params(axis='both', labelsize=22)
149     ax.legend(fontsize=28)
150     ax.set_title(r'SEIRSD model output', fontsize=28)
151     ax.set_xlabel(r'$t$ (days)', fontsize=28)
152     ax.set_ylabel(r'Individuals (millions)', fontsize=28)

```



```
150
151 # sol_sir = do_solve_cm(f_sir, tspan, w0_sir, abstol, reltol,
    param_args_sir)
152 # plot_solution_sir(sol_sir, np.int_(tspan[1]))
153
154
155 sol_seirsd = do_solve_cm(f_seirsd, tspan, w0_seirsd, abstol, reltol
    , param_args_seirsd)
156 plot_solution_seirsd(sol_seirsd, np.int_(tspan[1]))
```