

# ANSWERS: EXERCISE SHEET No. 8

## 1 Sequential modular approach

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Thu Feb 29 12:31:19 2024
4
5 @author: rafaed
6 """
7
8 import autograd.numpy as np
9 from autograd import jacobian
10 import scipy.optimize as sp
11 from numpy import linalg as LA
12
13
14 #parameters
15 sF = 1.0
16 sT = 1.0
17 rho = 50.0
18 a1 = 5.9755e9
19 a2 = 2.5962e12
20 a3 = 9.6283e15
21 V = 0.03
22 T = 6.744*sT
23 Fa = 13.357*sF
24 Fb = 24.482193508*sF
25 n = 0.9
26
27 #loop initial guess
28 Fr_a_inp = 4.69
29 Fr_b_inp = 14.54
30 Fr_c_inp = 0.769
31 Fr_e_inp = 14.40
32 Fr_p_inp = 1.91
33 Fr_g_inp = 0.318
34
35 # List of variables reactor (23)
36 # Fsum, Fa_eff, Fb_eff, Fc_eff, Fe_eff, Fp_eff, Fg, F_purge, k1, k2
37 #   , k3, k1e (5.71), k2e(7.58), k3e (8.80),
38 #   T2 (0.14), r1 (8.516), r2 (7.096), r3 (2.084), xa (0.128), xb
39 #   (0.397), xc (0.020), xe (0.393), xp (0.052), xg (0.00867), Fr_a
40 #   (4.69), Fr_b (14.54), Fr_c (0.769), Fr_e (14.40), Fr_p (1.91),
41 #   Fr_g (0.318)
42
43
44 ini_guess_reactor = np.array ([366.369, 46.907, 145.444, 7.692,
45                               144.033, 19.115, 3.178
46                               , 111.7, 567.6,
47                               1270.91521720,5.71,7.58,8.80,0.14, 8.516,7.096, 2.084,0.128,
48                               0.397,0.020,0.393, 0.052,0.00867 ])
49
50
51 def sequential_modular(Fr_a_inp,Fr_b_inp,Fr_c_inp,Fr_e_inp,Fr_p_inp
52 ,Fr_g_inp):
53     # define the residual function
54     def reactor (inp):
55         Fsum = inp[0]*sF
```

```

49     Fa_eff = inp[1]*sF
50     Fb_eff= inp[2]*sF
51     Fc_eff= inp[3]*sF
52     Fe_eff= inp[4]*sF
53     Fp_eff= inp[5]*sF
54     Fg= inp[6]*sF
55     k1= inp[7]
56     k2= inp[8]
57     k3= inp[9]
58     k1e = inp[10]
59     k2e = inp[11]
60     k3e = inp[12]
61     T2 = inp[13]
62     r1 = inp[14]
63     r2 = inp[15]
64     r3 = inp[16]
65     xa = inp[17]
66     xb = inp[18]
67     xc = inp[19]
68     xe = inp[20]
69     xp = inp[21]
70     xg = inp[22]
71
72     res0 = -k1e + np.log(a1) - 120.0*T2
73     res1 = -k2e + np.log(a2) - 150.0*T2
74     res2 = -k3e + np.log(a3) - 200*T2
75     res3 = T2*T - 1.0
76     res4 = k1 - np.exp(k1e)
77     res5 = k2 - np.exp(k2e)
78     res6 = k3 - np.exp(k3e)
79     res7 = -r1 + k1*xa*xb*V*rho
80     res8 = -r2 + k2*xc*xb*V*rho
81     res9 = -r3 + k3*xp*xc*V*rho
82     res10 = -Fa_eff + Fa + Fr_a_inp - r1
83     res11 = -Fb_eff + Fb + Fr_b_inp - (r1 + r2)
84     res12 = -Fc_eff + Fr_c_inp + 2.0*r1 - 2.0*r2 - r3
85     res13 = -Fe_eff + Fr_e_inp + 2.0*r2
86     res14 = -Fp_eff + 0.1*Fr_e_inp + r2 - 0.5*r3
87     res15 = -Fg + 1.5*r3
88     res16 = Fa_eff + Fb_eff + Fc_eff + Fe_eff + Fp_eff + Fg - Fsum
89     res17 = -Fa_eff + Fsum*xa
90     res18 = -Fb_eff + Fsum*xb
91     res19 = -Fc_eff + Fsum*xc
92     res20 = -Fe_eff + Fsum*xe
93     res21 = -Fp_eff + Fsum*xp
94     res22 = -Fg + Fsum*xg
95
96
97
98     res = np.array([res0 , res1 , res2, res3, res4, res5, res6,
99                     res7, res8, res9, res10, res11, res12, res13, res14, res15,
100                    res16, res17, res18, res19, res20, res21, res21, res22],dtype =
101                    np.float64)
102
103     return res # return a vector of residuals
104
105 # List of variables reactor (6)
106 #F_p, F_purge, Fr_a , Fr_b, Fr_c, Fr_e , Fr_p, Fr_g
107 def separator(inp_sep):
    Fa_eff = inp_sep[0]

```

```

1108     Fb_eff = inp_sep[1]
1109     Fc_eff = inp_sep[2]
1110     Fe_eff = inp_sep[3]
1111     Fp_eff = inp_sep[4]
1112     Fg = inp_sep[5]
1113
1114     Fp = Fp_eff - 0.1*Fe_eff
1115     F_purge = n*(Fa_eff + Fb_eff + Fc_eff + 1.1*Fe_eff)
1116     Fr_a = (1.0-n)*Fa_eff
1117     Fr_b = (1.0-n)*Fb_eff
1118     Fr_c = (1.0-n)*Fc_eff
1119     Fr_e = (1.0-n)*Fe_eff
1120     Fr_p = (1.0-n)*Fp_eff
1121     Fr_g = (1.0-n)*Fg
1122
1123     return Fp, F_purge, Fr_a, Fr_b, Fr_c, Fr_e, Fr_p, Fr_g
1124
1125
1126     jac_reactor = jacobian(reactor) # autograd to calculate the
1127                                     jacobian
1128     sol_reactor = sp.root (reactor,ini_guess_reactor,jac=jac_reactor,
1129                             method ="lm") # solve the problem
1130
1131     #print (sol_reactor)
1132
1133     inp_sep = sol_reactor.x[1:7]
1134
1135     Fp, F_purge, Fr_a_out, Fr_b_out, Fr_c_out, Fr_e_out, Fr_p_out,
1136     Fr_g_out = separator(inp_sep)
1137
1138     res =np.zeros(6)
1139     res [0] = Fr_a_inp - Fr_a_out
1140     res [1] =Fr_b_inp - Fr_b_out
1141     res [2] =Fr_c_inp - Fr_c_out
1142     res [3] =Fr_e_inp - Fr_e_out
1143     res [4] =Fr_p_inp - Fr_p_out
1144     res [5] =Fr_g_inp - Fr_g_out
1145
1146     return res, Fr_a_out, Fr_b_out, Fr_c_out, Fr_e_out, Fr_p_out,
1147     Fr_g_out
1148
1149 #Convergence loop - Successive Substitutions
1150 tol=1e-2
1151 res_norm = 100.0
1152 i=1
1153 while res_norm > tol:
1154     print(i)
1155     i = i +1
1156     res_k, Fr_a_out, Fr_b_out, Fr_c_out, Fr_e_out, Fr_p_out, Fr_g_out
1157     = sequential_modular(Fr_a_inp,Fr_b_inp,Fr_c_inp,Fr_e_inp,
1158                         Fr_p_inp,Fr_g_inp)
1159
1160
1161     res_norm = LA.norm(res_k)
1162     print(res_norm)
1163
1164     Fr_a_inp = Fr_a_out
1165     Fr_b_inp = Fr_b_out
1166     Fr_c_inp = Fr_c_out

```

```

164 Fr_e_inp = Fr_e_out
165 Fr_p_inp = Fr_p_out
166 Fr_g_inp = Fr_g_out
167
168 print(Fr_a_out, Fr_b_out, Fr_c_out, Fr_e_out, Fr_p_out, Fr_g_out)

```

1. //
2. //
3. Yes, we could instead make the convergence loop between the two models.  
An initial guess for  $F_g$  and  $F_{eff}$  would be needed.

## 2 Equation Oriented approach

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Thu Feb 29 12:31:19 2024
4
5  @author: rafaed
6  """
7
8  import autograd.numpy as np
9  from autograd import jacobian
10 import scipy.optimize as sp
11
12
13 #parameters
14 sF = 1.0
15 sT = 1.0
16 rho = 50.0
17 a1 = 5.9755e9
18 a2 = 2.5962e12
19 a3 = 9.6283e15
20 V = 0.03
21 T = 6.744*sT
22 Fa = 13.357*sF
23 Fb = 24.482193508*sF
24 n = 0.9
25
26 # List of variables (31)
27 # Fsum, Fa_eff, Fb_eff, Fc_eff, Fe_eff, Fp_eff, Fp, Fg, F_purge, k1
28 #   , k2, k3, k1e (5.71), k2e(7.58), k3e (8.80),
29 #   T2 (0.14), r1 (8.516), r2 (7.096), r3 (2.084), xa (0.128), xb
30 #   (0.397), xc (0.020), xe (0.393), xp (0.052), xg (0.00867), Fr_a
31 #   (4.69), Fr_b (14.54), Fr_c (0.769), Fr_e (14.40), Fr_p (1.91),
32 #   Fr_g (0.318)
33
34
35 ini_guess = np.array ([366.369, 46.907, 145.444, 7.692, 144.033,
36   19.115, 4.712, 3.178, 35.910
37   , 111.7, 567.6,
38   1270.91521720,5.71,7.58,8.80,0.14, 8.516,7.096, 2.084,0.128,
39   0.397,0.020,0.393, 0.052,0.00867, 4.69, 14.54, 0.769,14.40,
40   1.91, 0.318  ])
41
42
43
44
45 # solution = np.array([4.11439856e+01, 6.24409991e+00, 1.25004944e
46 +01, 8.72477464e-01,
47 #   1.22098633e+01, 3.76628040e+00, 2.54529407e+00, 5.55077018
48 #   e+00,
49 #   2.97431293e+01, 1.11870204e+02, 5.68543070e+02, 1.27091522
50 #   e+03,
51 #   4.71733931e+00, 6.34307707e+00, 7.14749256e+00, 1.48279953
52 #   e-01,
53 #   7.73731008e+00, 5.49443850e+00, 3.70051345e+00, 1.51762155
54 #   e-01,
55 #   3.03823127e-01, 2.12054678e-02, 2.96759372e-01, 9.15390267
56 #   e-02,
57 #   1.34910853e-01, 6.24409991e-01, 1.25004944e+00, 8.72477464
58 #   e-02,
59 #   1.22098633e+00, 3.76628040e-01, 5.55077018e-01])
60
61
62 # define the residual function
```

```

46 def fun (inp):
47     Fsum = inp[0]*sF
48     Fa_eff = inp[1]*sF
49     Fb_eff= inp[2]*sF
50     Fc_eff= inp[3]*sF
51     Fe_eff= inp[4]*sF
52     Fp_eff= inp[5]*sF
53     Fp= inp[6]*sF
54     Fg= inp[7]*sF
55     F_purge= inp[8]*sF
56     k1= inp[9]
57     k2= inp[10]
58     k3= inp[11]
59     k1e = inp[12]
60     k2e = inp[13]
61     k3e = inp[14]
62     T2 = inp[15]
63     r1 = inp[16]
64     r2 = inp[17]
65     r3 = inp[18]
66     xa = inp[19]
67     xb = inp[20]
68     xc = inp[21]
69     xe = inp[22]
70     xp = inp[23]
71     xg = inp[24]
72     Fr_a = inp[25]
73     Fr_b = inp[26]
74     Fr_c = inp[27]
75     Fr_e = inp[28]
76     Fr_p = inp[29]
77     Fr_g = inp[30]
78
79
80
81     res0 = -k1e + np.log(a1) - 120.0*T2
82     res1 = -k2e + np.log(a2) - 150.0*T2
83     res2 = -k3e + np.log(a3) - 200*T2
84     res3 = T2*T - 1.0
85     res4 = k1 - np.exp(k1e)
86     res5 = k2 - np.exp(k2e)
87     res6 = k3 - np.exp(k3e)
88     res7 = -r1 + k1*xa*xb*V*rho
89     res8 = -r2 + k2*xc*xb*V*rho
90     res9 = -r3 + k3*xp*xc*V*rho
91     res10 = -Fa_eff + Fa + Fr_a - r1
92     res11 = -Fb_eff + Fb + Fr_b - (r1 + r2)
93     res12 = -Fc_eff + Fr_c + 2.0*r1 - 2.0*r2 - r3
94     res13 = -Fe_eff + Fr_e + 2.0*r2
95     res14 = -Fp_eff + 0.1*Fr_e + r2 - 0.5*r3
96     res15 = -Fg + 1.5*r3
97     res16 = Fa_eff + Fb_eff + Fc_eff + Fe_eff + Fp_eff + Fg - Fsum
98     res17 = -Fa_eff + Fsum*xa
99     res18 = -Fb_eff + Fsum*xb
100    res19 = -Fc_eff + Fsum*xc
101    res20 = -Fe_eff + Fsum*xe
102    res21 = -Fp_eff + Fsum*xp
103    res22 = -Fg + Fsum*xg
104    res23 = -Fp_eff + 0.1*Fe_eff + Fp
105    res24 = -F_purge + n*(Fa_eff + Fb_eff + Fc_eff + 1.1*Fe_eff)
106    res25 = -Fr_a + (1.0-n)*Fa_eff
107    res26 = -Fr_b + (1.0-n)*Fb_eff

```

```

108     res27 = -Fr_c + (1.0-n)*Fc_eff
109     res28 = -Fr_e + (1.0-n)*Fe_eff
110     res29 = -Fr_p + (1.0-n)*Fp_eff
111     res30 = -Fr_g + (1.0-n)*Fg
112
113
114
115     res = np.array([res0 , res1 , res2, res3, res4, res5, res6,
116                    res7, res8, res9, res10, res11, res12, res13, res14, res15,
117                    res16, res17, res18, res19, res20, res21, res21, res22, res23,
118                    res24, res25, res26, res27, res28, res29, res30], dtype = np.
119                    float64)
120
121     return res # return a vector of residuals
122
123
124
125 jac = jacobian(fun) # autograd to calculate the jacobian
126
127 sol = sp.root (fun, ini_guess, jac=jac, method = "lm") # solve the
128                problem
129
130 print (sol)

```

1. We can use the solution of the sequential modular approach as an initial guess.
2. //
3. ODE (DAE) equations with the initial conditions. The equation-oriented approach would be the most suitable method since performing the convergence loop for each point in time would be time-consuming.