

Project 1

Matthew Callahan, Jiajing Guan, and Aneesh Malhotra

September 26, 2018

1 Problem 1

(a) Let

$$J(\theta^{n-1}) = \begin{bmatrix} \frac{dx(t_1, \theta^{n-1})}{d\theta^{n-1}} \\ \frac{dx(t_2, \theta^{n-1})}{d\theta^{n-1}} \\ \frac{dx(t_3, \theta^{n-1})}{d\theta^{n-1}} \\ \vdots \\ \frac{dx(t_n, \theta^{n-1})}{d\theta^{n-1}} \end{bmatrix}$$

and

$$G(\theta^{n-1}) = \begin{bmatrix} y_1 - x(t_1, \theta^{n-1}) \\ y_2 - x(t_2, \theta^{n-1}) \\ y_3 - x(t_3, \theta^{n-1}) \\ \vdots \\ y_n - x(t_n, \theta^{n-1}) \end{bmatrix}$$

where y_i is the data at t_i and $x(t_i, \theta^{n-1})$ is the approximate at t_i using θ^{n-1} as the parameter ($i = 1, \dots, n$).

Then for each iteration step:

$$H(\theta^{n-1}) = J(\theta^{n-1})^T J(\theta^{n-1})$$

$$g(\theta^{n-1}) = J(\theta^{n-1})^T G(\theta^{n-1})$$

$$\theta^n = \theta^{n-1} + H(\theta^{n-1})^{-1} g(\theta^{n-1})$$

(b) Sensitivity System:

$$D \begin{bmatrix} x \\ \partial_\theta x \end{bmatrix} = \begin{bmatrix} x^2 \theta \\ x^2 + 2x\theta u \end{bmatrix}$$

$$\begin{bmatrix} x \\ u \end{bmatrix} (1) = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

(c) See code at the end of this document.

- (d) Using the built-in **fminsearch** function, we get $\theta \approx 1.8433$. The optimal parameter found by Gauss-Newton method was $\theta \approx 1.8319$.
- (e) The following figure shows the convergence of θ and SSE:

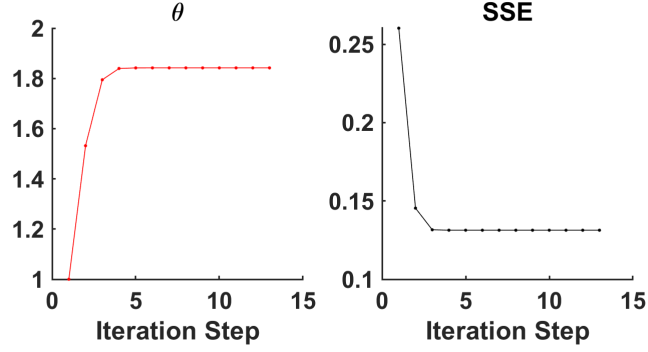


Figure 1: Convergence of θ and SSE

2 Problem 2

- (a) Let

$$J(\theta^{n-1}) = \begin{bmatrix} \frac{dx(t_1, \theta^{n-1}, x_0)}{d\theta^{n-1}} & \frac{dx(t_1, \theta^{n-1}, x_0)}{dx_0} \\ \frac{dx(t_2, \theta^{n-1}, x_0)}{d\theta^{n-1}} & \frac{dx(t_2, \theta^{n-1}, x_0)}{dx_0} \\ \frac{dx(t_3, \theta^{n-1}, x_0)}{d\theta^{n-1}} & \frac{dx(t_3, \theta^{n-1}, x_0)}{dx_0} \\ \vdots & \vdots \\ \frac{dx(t_n, \theta^{n-1}, x_0)}{d\theta^{n-1}} & \frac{dx(t_n, \theta^{n-1}, x_0)}{dx_0} \end{bmatrix}$$

and

$$G(\theta^{n-1}) = \begin{bmatrix} y_1 - x(t_1, \theta^{n-1}, x_0) \\ y_2 - x(t_2, \theta^{n-1}, x_0) \\ y_3 - x(t_3, \theta^{n-1}, x_0) \\ \vdots \\ y_n - x(t_n, \theta^{n-1}, x_0) \end{bmatrix}$$

where y_i is the data at t_i and $x(t_i, \theta^{n-1})$ is the approximate at t_i using θ^{n-1} as the parameter ($i = 1, \dots, n$).

Then for each iteration step:

$$H(\theta^{n-1}) = J(\theta^{n-1})^T J(\theta^{n-1})$$

$$g(\theta^{n-1}) = J(\theta^{n-1})^T G(\theta^{n-1})$$

$$\theta^n = \theta^{n-1} + H(\theta^{n-1})^{-1} g(\theta^{n-1})$$

(b) Sensitivity System:

$$D \begin{bmatrix} x \\ \partial_{\theta} x \\ \partial_{x_0} x \end{bmatrix} (1) = \begin{bmatrix} -0.9 \\ 0 \\ 1\theta v \end{bmatrix}$$

(c) See code at the end of this document.

(d) Using the build in function **fminsearch**, we get $\theta \approx 1.7553$ and $x_0 \approx -0.8963$. Using the Gauss-Newton method, we get $\theta \approx 1.7543$ and $x_0 \approx -0.8963$.

(e) The following figure shows the convergence of θ , x_0 and SSE:

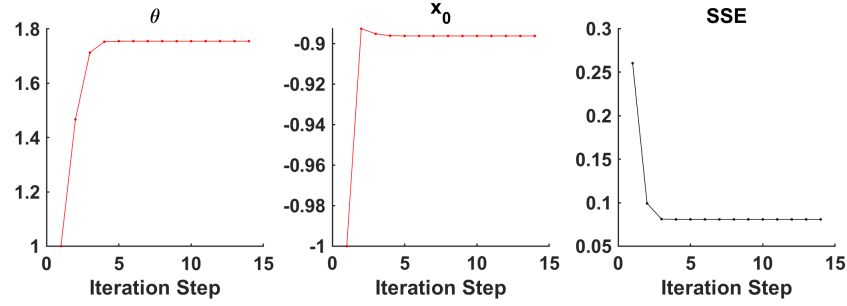


Figure 2: Convergence of θ , x_0 and SSE

3 Problem 3

(a) This problem is similar to problem 1 (a), just replace the Jacobian matrix with the one below:

$$J(a^{n-1}, b^{n-1}) = \begin{bmatrix} \frac{dx(t_1, a^{n-1}, b^{n-1})}{da^{n-1}} & \frac{dy(t_1, a^{n-1}, b^{n-1})}{da^{n-1}} & \frac{dx(t_1, a^{n-1}, b^{n-1})}{db^{n-1}} & \frac{dy(t_1, a^{n-1}, b^{n-1})}{db^{n-1}} \\ \frac{dx(t_2, a^{n-1}, b^{n-1})}{da^{n-1}} & \frac{dy(t_2, a^{n-1}, b^{n-1})}{da^{n-1}} & \frac{dx(t_2, a^{n-1}, b^{n-1})}{db^{n-1}} & \frac{dy(t_2, a^{n-1}, b^{n-1})}{db^{n-1}} \\ \frac{dx(t_3, a^{n-1}, b^{n-1})}{da^{n-1}} & \frac{dy(t_3, a^{n-1}, b^{n-1})}{da^{n-1}} & \frac{dx(t_3, a^{n-1}, b^{n-1})}{db^{n-1}} & \frac{dy(t_3, a^{n-1}, b^{n-1})}{db^{n-1}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{dx(t_n, a^{n-1}, b^{n-1})}{da^{n-1}} & \frac{dy(t_n, a^{n-1}, b^{n-1})}{da^{n-1}} & \frac{dx(t_n, a^{n-1}, b^{n-1})}{db^{n-1}} & \frac{dy(t_n, a^{n-1}, b^{n-1})}{db^{n-1}} \end{bmatrix}$$

In each iteration step, the first two columns are added with the last two columns to create the gradient matrix with respect to a and b .

(b)

$$D \begin{bmatrix} x \\ y \\ \partial_a x \\ \partial_a y \\ \partial_b x \\ \partial_b y \end{bmatrix} = \begin{bmatrix} -axy \\ axy - by \\ -xy - ay\partial_a x - ayx\partial_a y \\ xy + ay\partial_a x + (ax - b)\partial_a y \\ -ay\partial_b x - ax\partial_b y - y + ay\partial_b x + (ax - b)\partial_b y \end{bmatrix}$$

$$D \begin{bmatrix} x \\ y \\ \partial_a x \\ \partial_a y \\ \partial_b x \\ \partial_b y \end{bmatrix} (0) = \begin{bmatrix} 0.9 \\ 0.1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

(c) See code at the end of this document.

(d) Using the built-in **fminsearch** function, we get $a \approx 0.5023$ and $b \approx 0.1031$.
The optimal parameters found by Gauss-Newton method was $a \approx 0.5132$ and $b \approx 0.1091$.

(e) The following figure shows the convergence of a , b and SSE:

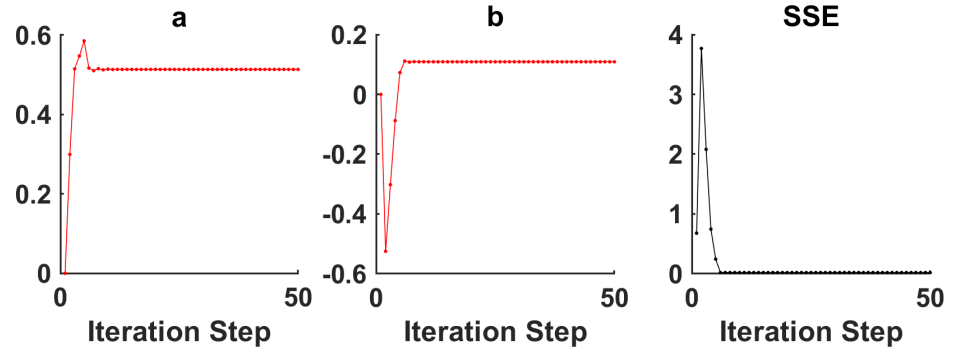
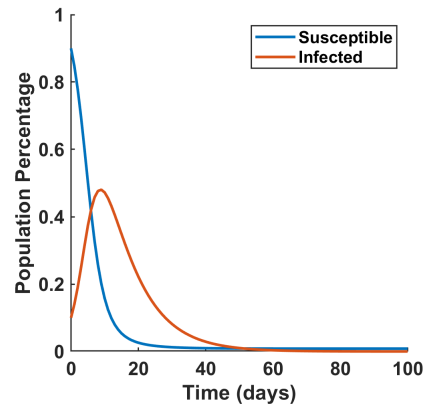


Figure 3: Convergence of a , b and SSE

(f) The figure below shows the population percentage over time: As shown in Figure (f), the susceptible population levels off to around 0.0085 while the infected population goes to zero.



Matlab Code

Problem 1

```

1  close all;
2  clear all;
3  format long;
4  plotting=1;
5
6  global xexp
7  global texp
8  global xt0
9
10 %Experimental data
11 xexp = [-0.9 -0.3 -0.28 -0.15 -0.13]';
12 texp = [1:5];
13 xt0 = -1;
14 a_opt_init = 1;
15
16 method = 2; % 1=fminsearch, 2=Newton
17
18 if (method==1)
19     % Finding the parameter values minimizing SSE error
        using built-in function
20     [a_opt,fval,exitflag,output] = fminsearch(@SSE,[
        a_opt_init])
21
22 else
23     %Newton-Raphson method
24     aa(1)=[a_opt_init];
25     error(1) = SSE(aa(1));

```

```

26     delta1 = 1e-16;
27     delta2 = 1e-16;
28     Nsteps = 200;
29
30     for k=1:Nsteps
31
32         %Need to solve ODE on a finer grid
33         tt = linspace(texp(1),texp(end),1000);
34         zt0 =[xt0,0]; %initial data for the sensitivity
            system
35         [tspan,zz] = ode45(@(t,z)sens(t,z,aa(k)),tt,zt0);
36         z = interp1(tt,zz,texp); %getting back to the
            coarser grid to compare with xexp
37
38         %Jacobian calculation
39         J1 = [z(:,2)];
40
41         %Hessian with weights
42         H = J1'*J1;
43         g = J1'*(xexp - z(:,1));
44
45
46         %Newton direction = inv(H)*g, but
47         %Better to use slash command for a more stable
            calculation
48         aa(k+1) = aa(k) + (H\g);
49         a_opt = aa(k+1);
50         error(k+1) = SSE(a_opt);
51
52         if (norm(aa(k+1)-aa(k))<delta1) | (abs(SSE(aa(k)
            +1))-SSE(aa(k)))<delta2)
53             % Plotting convergence of parameters
54             figure(2);
55             subplot(1,2,1);
56             set(gca,'FontName','Arial','FontSize',14,'
                FontWeight','Bold','LineWidth', 1);
57             hold on;
58             title('\theta')
59             xlabel("Iteration Step")
60             axis square;
61             plot(1:length(aa),aa,'r.-');
62
63             subplot(1,2,2);
64             set(gca,'FontName','Arial','FontSize',14,'
                FontWeight','Bold','LineWidth', 1);
65             hold on;

```

```

66         title('SSE');
67         xlabel("Iteration Step")
68         axis square;
69         plot(1:length(aa),error,'k.-');
70         break;
71     end
72 end
73 end
74
75 if (plotting==1)
76
77     [tspan,x] = ode45(@(t,x)system(t,x,a_opt),texp,xt0);
78
79     figure(1);
80     set(gca,'FontName','Arial','FontSize',14,'FontWeight',
81         'Bold','LineWidth',1);
82     hold on;
83     axis square;
84     title('x vs. t');
85     plot(tspan,x,'r. ');
86     plot(texp,xexp,'bo');
87     legend('ODE','data');
88 end
89
90 function z = SSE(a)
91     global texp
92     global xexp
93     global xt0
94
95     [tspan,x] = ode45(@(t,x)system(t,x,a),texp,[xt0]);
96
97     z = norm(x-xexp,2);
98
99 end
100
101 function dxdt=system(t,x,a)
102
103     dxdt = a*x^2;
104
105 end
106
107 function dzdt = sens(t,z,a)
108
109     dzdt(1)=a*z(1)^2;
110     dzdt(2)=z(1)^2+2*a*z(1)*z(2);

```

```

111
112 dzdt = dzdt';
113 end

```

Problem 2

```

1  close all;
2  clear all;
3  format long;
4  plotting=1;
5
6  global xexp
7  global texp
8
9  %Experimental data
10 xexp = [-0.9 -0.3 -0.28 -0.15 -0.13]';
11 texp = [1:5];
12 a_opt_init = 1;
13 xt0_init = -1;
14
15 method =2; % 1=fminsearch, 2=Newton
16
17 if (method==1)
18     % Finding the parameter values minimizing SSE error
19     % using built-in function
20     [a_opt,fval,exitflag,output] = fminsearch(@SSE,[
21         a_opt_init xt0_init])
22
23 else
24     %Newton-Raphson method
25     aa(1,:)=[a_opt_init xt0_init];
26     error(1) = SSE(aa(1,:));
27     delta1 = 1e-16;
28     delta2 = 1e-16;
29     Nsteps = 200;
30
31     for k=1:Nsteps
32
33         %Need to solve ODE on a finer grid
34         tt = linspace(texp(1),texp(end),100);
35         zt0 =[aa(k,2);0;1]; %initial data for the
36         %sensitivity system
37         [tspan,zz] = ode45(@(t,z)sens(t,z,aa(k,1)),tt,zt0
38             );
39         z = interp1(tt,zz,texp); %getting back to the
40         %coarser grid to compare with xexp

```



```

36
37 %Jacobian calculation
38 J1 = [z(:,2:3)];
39
40 %Hessian with weights
41 H = J1'*J1;
42 g = J1'*(xexp - z(:,1));
43
44
45 %Newton direction = inv(H)*g, but
46 %Better to use slash command for a more stable
   calculation
47 aa(k+1,:) = aa(k,:) + (H\g)';
48 a_opt = aa(k+1,:);
49 error(k+1) = SSE(a_opt);
50
51 if (norm(aa(k+1,:)-aa(k,:))<delta1) | (abs(SSE(aa
   (k+1,:))-SSE(aa(k,:)))<delta2)
52 % Plotting convergence of parameters
53 figure(2);
54 subplot(1,3,1);
55 set(gca,'FontName','Arial','FontSize',14,'
   FontWeight','Bold','LineWidth',1);
56 hold on;
57 title('\theta')
58 xlabel("Iteration Step")
59 axis square;
60 plot(1:length(aa(:,1)),aa(:,1),'r.-');
61
62 subplot(1,3,2);
63 set(gca,'FontName','Arial','FontSize',14,'
   FontWeight','Bold','LineWidth',1);
64 hold on;
65 title('x_0')
66 xlabel("Iteration Step")
67 axis square;
68 plot(1:length(aa(:,2)),aa(:,2),'r.-');
69
70 subplot(1,3,3);
71 set(gca,'FontName','Arial','FontSize',14,'
   FontWeight','Bold','LineWidth',1);
72 hold on;
73 title('SSE');
74 xlabel("Iteration Step")
75 axis square;
76 plot(1:length(aa(:,1)),error,'k.-');

```

```

77         break;
78     end
79 end
80 end
81
82 if (plotting==1)
83
84     [tspan,x] = ode45(@(t,x)system(t,x,a_opt(1)),texp,
85         a_opt(2));
86
87     figure(1);
88     set(gca,'FontName','Arial','FontSize',14,'FontWeight',
89         'Bold','LineWidth',1);
90     hold on;
91     axis square;
92     title('x vs. t');
93     plot(tspan,x,'r');
94     plot(texp,xexp,'bo');
95     legend('ODE','data');
96
97 end
98
99 function z = SSE(a)
100
101     global texp
102     global xexp
103
104     xt0 =a(2);
105     [tspan,x] = ode45(@(t,x)system(t,x,a(1)),texp,xt0);
106
107     z = norm(x-xexp,2);
108
109 end
110
111 function dxdt=system(t,x,a)
112
113     dxdt = a*x^2;
114
115 end
116
117 function dzdt = sens(t,z,a)
118
119     dzdt(1)=a*z(1)^2;
120     dzdt(2)=z(1)^2+2*a*z(1)*z(2);
121     dzdt(3)=2*a*z(1)*z(3);
122     dzdt = dzdt';
123
124 end

```

Problem 3

```
1 close all;
2 clear all;
3 format long;
4 plotting=1;
5
6 global xexp
7 global texp
8 global xt0
9
10 %Experimental data
11 xexp = [0.9 0.858 0.78 0.7 0.6 0.5;
12         0.1 0.14 0.2 0.25 0.3 0.38]';
13 texp = [0:5];
14 xt0 = [0.9 0.1];
15
16 method =2; % 1=fminsearch , 2=Newton
17
18 if (method==1)
19     % Finding the parameter values minimizing SSE error
20     % using built-in function
21     [a_opt,fval,exitflag,output] = fminsearch(@SSE,[0 0])
22 else
23     %Newton-Raphson method
24     aa(1,:)= [0 0];
25     error(1) = SSE(aa(1,:));
26     delta1 = 1e-16;
27     delta2 = 1e-16;
28     Nsteps = 200;
29
30     for k=1:Nsteps
31
32         %Need to solve ODE on a finer grid
33         tt = linspace(texp(1),texp(end),300);
34         zt0 =[xt0 0 0 0 0]; %initial data for the
35         %sensitivity system
36         [tspan,zz] = ode45(@(t,z)sens(t,z,aa(k,:)),tt,zt0);
37
38         z = interp1(tt,zz,texp); %getting back to the
39         %coarser grid to compare with xexp
40
41         %Jacobian calculation
42         J1 = [z(:,3:4)];
43         J2 = [z(:,5:6)];
```

```

41
42 %Hessian with weights
43 H = J1'*J1 + J2'*J2;
44 g = J1'*(xexp(:,1) - z(:,1)) + J2'*(xexp(:,2) - z
    (:,2));
45
46
47 %Newton direction = inv(H)*g, but
48 %Better to use slash command for a more stable
    calculation
49 aa(k+1,:) = aa(k,:) + (H\g)';
50 a_opt = aa(k+1,:);
51 error(k+1) = SSE(a_opt);
52
53 if (norm(aa(k+1,:)-aa(k,:))<delta1) | (abs(SSE(aa
    (k+1,:))-SSE(aa(k,:)))<delta2)
54     tt=[0:100];
55     [tspan,xd] = ode45(@(t,x)system(t,x,a_opt),tt
        ,xt0);
56     figure(3)
57     set(gca,'FontName','Arial','FontSize',14,'
        FontWeight','Bold','LineWidth',1);
58     hold on;
59     xlabel("Time (days)")
60     ylabel("Population Percentage")
61     axis square;
62     plot(tspan,xd,'LineWidth',2)
63     legend("Susceptible","Infected")
64
65 % Plotting convergence of parameters
66 figure(2);
67 subplot(1,3,1);
68 set(gca,'FontName','Arial','FontSize',14,'
    FontWeight','Bold','LineWidth',1);
69 hold on;
70 title('a')
71 xlabel("Iteration Step")
72 axis square;
73 plot(1:length(aa(:,1)),aa(:,1),'r.-');
74
75 subplot(1,3,2);
76 set(gca,'FontName','Arial','FontSize',14,'
    FontWeight','Bold','LineWidth',1);
77 hold on;
78 title('b')
79 xlabel("Iteration Step")

```

```

80         axis square;
81         plot(1:length(aa(:,2)),aa(:,2),'r.-');
82
83         subplot(1,3,3);
84         set(gca,'FontName','Arial','FontSize',14,'
            FontWeight','Bold','LineWidth',1);
85         hold on;
86         title('SSE');
87         xlabel("Iteration Step")
88         axis square;
89         plot(1:length(aa(:,1)),error,'k.-');
90         break;
91     end
92 end
93 end
94
95 if (plotting==1)
96
97     [tspan,x] = ode45(@(t,x)system(t,x,a_opt),texp,xt0);
98
99     figure(1);
100    subplot(1,2,1);
101    set(gca,'FontName','Arial','FontSize',14,'FontWeight'
        , 'Bold','LineWidth',1);
102    hold on;
103    axis square;
104    title('x vs. t');
105    plot(tspan(:,1),x(:,1),'r. ');
106    plot(texp,xexp(:,1),'bo');
107    legend('ODE','data');
108
109    subplot(1,2,2);
110    set(gca,'FontName','Arial','FontSize',14,'FontWeight'
        , 'Bold','LineWidth',1);
111    hold on;
112    axis square;
113    title('y vs. t');
114    plot(tspan,x(:,2),'r. ');
115    plot(texp,xexp(:,2),'bo');
116    legend('ODE','data');
117
118 end
119
120 function z = SSE(a)
121 global texp
122 global xexp

```

```

123 global xt0
124
125 [tspan,x] = ode45(@(t,x)system(t,x,a),texp,xt0);
126
127 z = norm(x-xexp,2);
128
129 end
130
131 function dxdt=system(t,x,a)
132
133 dxdt(1) = -a(1)*x(1)*x(2);
134 dxdt(2) = a(1)*x(1)*x(2) - a(2)*x(2);
135
136 dxdt = dxdt';
137
138 end
139
140
141 function dzdt = sens(t,z,a)
142 dzdt(1) = -a(1)*z(1)*z(2);
143 dzdt(2) = a(1)*z(1)*z(2)-a(2)*z(2);
144 dzdt(3) = -z(1)*z(2) - a(1)*z(2)*z(3) - a(1)*z(2)*z(4);
145 dzdt(4) = z(1)*z(2) + a(1)*z(2)*z(3) + (a(1)*z(1)-a(2))*z
      (4);
146 dzdt(5) = -a(1)*z(2)*z(5) -a(1)*z(1)*z(6);
147 dzdt(6) = -z(2) + a(1)*z(2)*z(5) + (a(1)*z(1)-a(2))*z(6);
148 dzdt = dzdt';
149 end

```