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 $(i = 1, \ldots, n)$.
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:

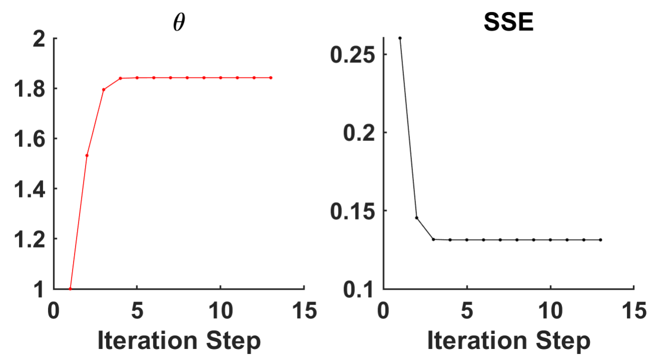


Figure 1: Convergence of θ and SSE

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 ($i = 1,\ldots,n$).
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:

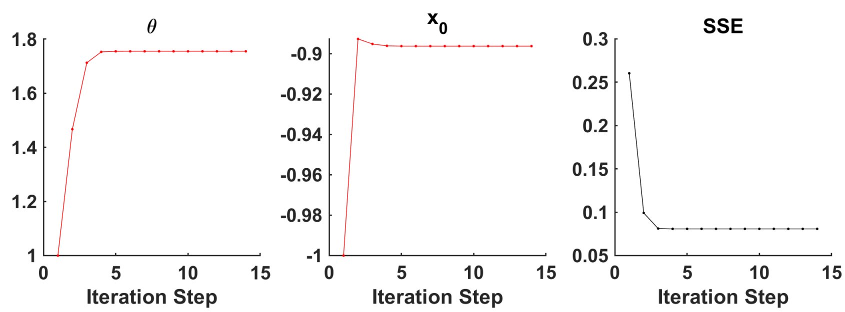


Figure 2: Convergence of θ, x_0 and SSE

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:

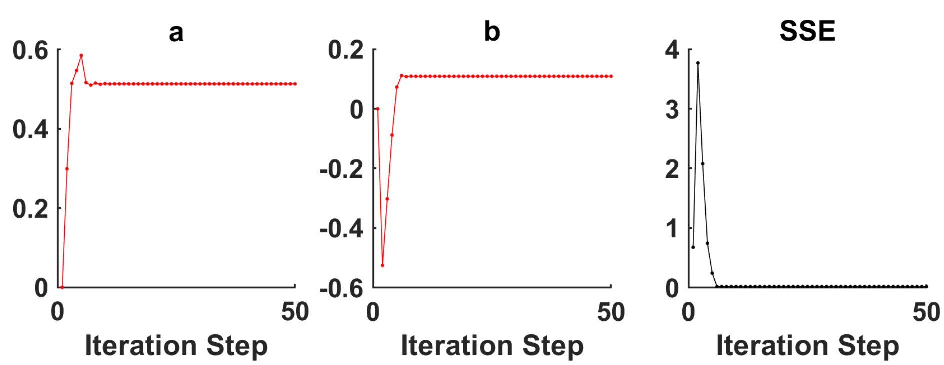
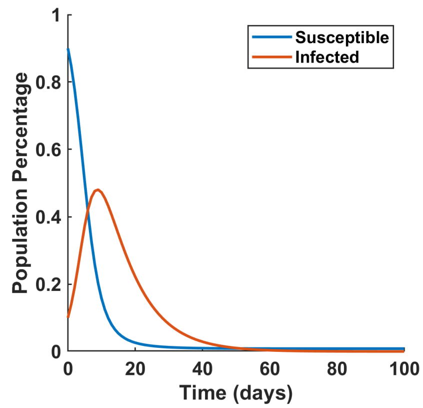


Figure 3: Convergence of a, b and SSE

(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

```
close all;
clear all;
format long;
plotting=1;

global xexp
global texp
global xt0

%Experimental data
xexp = [-0.9 -0.3 -0.28 -0.15 -0.13]';
texp = [1:5];
xt0 = -1;
a_opt_init = 1;

method = 2; % 1=fminsearch, 2=Newton

if (method==1)
    % Finding the parameter values minimizing SSE error
        using built-in function
    [a_opt,fval,exitflag,output] = fminsearch(@SSE,[
        a_opt_init])

else
    %Newton-Raphson method
    aa(1)=[a_opt_init];
    error(1) = SSE(aa(1));
```

```
    delta1 = 1e-16;
    delta2 = 1e-16;
    Nsteps = 200;

    for k=1:Nsteps

        %Need to solve ODE on a finer grid
        tt = linspace(texp(1),texp(end),1000);
        zt0 =[xt0,0]; %initial data for the sensitivity system
        [tspan,zz] = ode45(@(t,z)sens(t,z,aa(k)),tt,zt0);
        z = interp1(tt,zz,texp); %getting back to the coarser grid to compare with xexp

        %Jacobian calculation
        J1 = [z(:,2)];

        %Hessian with weights
        H = J1'*J1;
        g = J1'*(xexp - z(:,1));


        %Newton direction = inv(H)*g, but
        %Better to use slash command for a more stable calculation
        aa(k+1) = aa(k) + (H\g);
        a_opt = aa(k+1);
        error(k+1) = SSE(a_opt);

        if (norm(aa(k+1)-aa(k))<delta1) | (abs(SSE(aa(k+1))-SSE(aa(k)))<delta2)
            % Plotting convergence of parameters
            figure(2);
            subplot(1,2,1);
            set(gca,'FontName','Arial','FontSize',14,'FontWeight','Bold','LineWidth', 1);
            hold on;
            title('\theta')
            xlabel("Iteration Step")
            axis square;
            plot(1:length(aa),aa,'r.-');

            subplot(1,2,2);
            set(gca,'FontName','Arial','FontSize',14,'FontWeight','Bold','LineWidth', 1);
            hold on;
```

```
            title('SSE');
            xlabel("Iteration Step")
            axis square;
            plot(1:length(aa),error,'k.-');
            break;
        end
    end
end

if (plotting==1)

    [tspan,x] = ode45(@(t,x)system(t,x,a_opt),texp,xt0);

    figure(1);
    set(gca,'FontName','Arial','FontSize',14,'FontWeight'
        ,'Bold','LineWidth', 1);
    hold on;
    axis square;
    title('x vs. t');
    plot(tspan,x,'r.');
    plot(texp,xexp,'bo');
    legend('ODE','data');

end

function z = SSE(a)
global texp
global xexp
global xt0

[tspan,x] = ode45(@(t,x)system(t,x,a),texp,[xt0]);

z = norm(x-xexp,2);

end

function dxdt=system(t,x,a)

dxdt = a*x^2;

end

function dzdt = sens(t,z,a)

dzdt(1)=a*z(1)^2;
dzdt(2)=z(1)^2+2*a*z(1)*z(2);
```

```
dzdt = dzdt';
end
```

Problem 2

```
close all;
clear all;
format long;
plotting=1;

global xexp
global texp

%Experimental data
xexp = [-0.9 -0.3 -0.28 -0.15 -0.13]';
texp = [1:5];
a_opt_init = 1;
xt0_init = -1;

method =2; % 1=fminsearch, 2=Newton

if (method==1)
    % Finding the parameter values minimizing SSE error
        using built-in function
    [a_opt,fval,exitflag,output] = fminsearch(@SSE,[
        a_opt_init xt0_init])

else
    %Newton-Raphson method
    aa(1,:)=[a_opt_init xt0_init];
    error(1) = SSE(aa(1,:));
    delta1 = 1e-16;
    delta2 = 1e-16;
    Nsteps = 200;

    for k=1:Nsteps

        %Need to solve ODE on a finer grid
        tt = linspace(texp(1),texp(end),100);
        zt0 =[aa(k,2);0;1]; %initial data for the
            sensitivity system
        [tspan,zz] = ode45(@(t,z)sens(t,z,aa(k,1)),tt,zt0
            );
        z = interp1(tt,zz,texp); %getting back to the
            coarser grid to compare with xexp
```

```

        %Jacobian calculation
        J1 = [z(:,2:3)];

        %Hessian with weights
        H = J1'*J1;
        g = J1'*(xexp - z(:,1));


        %Newton direction = inv(H)*g, but
        %Better to use slash command for a more stable calculation
        aa(k+1,:) = aa(k,:) + (H\g)';
        a_opt = aa(k+1,:);
        error(k+1) = SSE(a_opt);

        if (norm(aa(k+1,:)-aa(k,:))<delta1) | (abs(SSE(aa(k+1,:))-SSE(aa(k,:)))<delta2)
            % Plotting convergence of parameters
            figure(2);
            subplot(1,3,1);
            set(gca,'FontName','Arial','FontSize',14,'FontWeight','Bold','LineWidth', 1);
            hold on;
            title('\theta')
            xlabel("Iteration Step")
            axis square;
            plot(1:length(aa(:,1)),aa(:,1),'r.-');

            subplot(1,3,2);
            set(gca,'FontName','Arial','FontSize',14,'FontWeight','Bold','LineWidth', 1);
            hold on;
            title('x_0')
            xlabel("Iteration Step")
            axis square;
            plot(1:length(aa(:,2)),aa(:,2),'r.-');

            subplot(1,3,3);
            set(gca,'FontName','Arial','FontSize',14,'FontWeight','Bold','LineWidth', 1);
            hold on;
            title('SSE');
            xlabel("Iteration Step")
            axis square;
            plot(1:length(aa(:,1)),error,'k.-');
```

```
                break;
            end
        end
end

if (plotting==1)

    [tspan,x] = ode45(@(t,x)system(t,x,a_opt(1)),texp,
        a_opt(2));

    figure(1);
    set(gca,'FontName','Arial','FontSize',14,'FontWeight'
        ,'Bold','LineWidth', 1);
    hold on;
    axis square;
    title('x vs. t');
    plot(tspan,x,'r.');
    plot(texp,xexp,'bo');
    legend('ODE','data');

end

function z = SSE(a)
global texp
global xexp

xt0 =a(2);
[tspan,x] = ode45(@(t,x)system(t,x,a(1)),texp,xt0);

z = norm(x-xexp,2);

end

function dxdt=system(t,x,a)

dxdt = a*x^2;

end

function dzdt = sens(t,z,a)

dzdt(1)=a*z(1)^2;
dzdt(2)=z(1)^2+2*a*z(1)*z(2);
dzdt(3)=2*a*z(1)*z(3);
dzdt = dzdt';
end
```

Problem 3

```
close all;
clear all;
format long;
plotting=1;

global xexp
global texp
global xt0

%Experimental data
xexp = [0.9 0.858 0.78 0.7 0.6 0.5;
    0.1 0.14 0.2 0.25 0.3 0.38]';
texp = [0:5];
xt0 = [0.9 0.1];

method =2; % 1=fminsearch, 2=Newton

if (method==1)
    % Finding the parameter values minimizing SSE error using built-in function
    [a_opt,fval,exitflag,output] = fminsearch(@SSE,[0 0])

else
    %Newton-Raphson method
    aa(1,:)=[0 0];
    error(1) = SSE(aa(1,:));
    delta1 = 1e-16;
    delta2 = 1e-16;
    Nsteps = 200;

    for k=1:Nsteps

        %Need to solve ODE on a finer grid
        tt = linspace(texp(1),texp(end),300);
        zt0 =[xt0 0 0 0 0]; %initial data for the sensitivity system
        [tspan,zz] = ode45(@(t,z)sens(t,z,aa(k,:)),tt,zt0);
        z = interp1(tt,zz,texp); %getting back to the coarser grid to compare with xexp

        %Jacobian calculation
        J1 = [z(:,3:4)];
        J2 = [z(:,5:6)];
```

```
        %Hessian with weights
        H = J1'*J1 + J2'*J2;
        g = J1'*(xexp(:,1) - z(:,1)) + J2'*(xexp(:,2) - z(:,2));


        %Newton direction = inv(H)*g, but
        %Better to use slash command for a more stable calculation
        aa(k+1,:) = aa(k,:) + (H\g)';
        a_opt = aa(k+1,:);
        error(k+1) = SSE(a_opt);

        if (norm(aa(k+1,:)-aa(k,:))<delta1) | (abs(SSE(aa(k+1,:))-SSE(aa(k,:)))<delta2)
            tt=[0:100];
            [tspan,xd] = ode45(@(t,x)system(t,x,a_opt),tt,xt0);
            figure(3)
            set(gca,'FontName','Arial','FontSize',14,'FontWeight','Bold','LineWidth', 1);
            hold on;
            xlabel("Time (days)")
            ylabel("Population Percentage")
            axis square;
            plot(tspan,xd,'LineWidth',2)
            legend("Susceptible","Infected")

            % Plotting convergence of parameters
            figure(2);
            subplot(1,3,1);
            set(gca,'FontName','Arial','FontSize',14,'FontWeight','Bold','LineWidth', 1);
            hold on;
            title('a')
            xlabel("Iteration Step")
            axis square;
            plot(1:length(aa(:,1)),aa(:,1),'r.-');

            subplot(1,3,2);
            set(gca,'FontName','Arial','FontSize',14,'FontWeight','Bold','LineWidth', 1);
            hold on;
            title('b')
            xlabel("Iteration Step")
```

```
            axis square;
            plot(1:length(aa(:,2)),aa(:,2),'r.-');

            subplot(1,3,3);
            set(gca,'FontName','Arial','FontSize',14,'FontWeight','Bold','LineWidth', 1);
            hold on;
            title('SSE');
            xlabel("Iteration Step")
            axis square;
            plot(1:length(aa(:,1)),error,'k.-');
            break;
        end
    end
end

if (plotting==1)

    [tspan,x] = ode45(@(t,x)system(t,x,a_opt),texp,xt0);

    figure(1);
    subplot(1,2,1);
    set(gca,'FontName','Arial','FontSize',14,'FontWeight','Bold','LineWidth', 1);
    hold on;
    axis square;
    title('x vs. t');
    plot(tspan(:,1),x(:,1),'r.');
    plot(texp,xexp(:,1),'bo');
    legend('ODE','data');

    subplot(1,2,2);
    set(gca,'FontName','Arial','FontSize',14,'FontWeight','Bold','LineWidth', 1);
    hold on;
    axis square;
    title('y vs. t');
    plot(tspan,x(:,2),'r.');
    plot(texp,xexp(:,2),'bo');
    legend('ODE','data');

end

function z = SSE(a)
global texp
global xexp
```

```
global xt0

[tspan,x] = ode45(@(t,x)system(t,x,a),texp,xt0);

z = norm(x-xexp,2);

end

function dxdt=system(t,x,a)

dxdt(1) = -a(1)*x(1)*x(2);
dxdt(2) = a(1)*x(1)*x(2) - a(2)*x(2);

dxdt = dxdt';

end


function dzdt = sens(t,z,a)
dzdt(1) = -a(1)*z(1)*z(2);
dzdt(2) = a(1)*z(1)*z(2)-a(2)*z(2);
dzdt(3) = -z(1)*z(2) - a(1)*z(2)*z(3) - a(1)*z(2)*z(4);
dzdt(4) = z(1)*z(2) + a(1)*z(2)*z(3) + (a(1)*z(1)-a(2))*z
    (4);
dzdt(5) = -a(1)*z(2)*z(5) -a(1)*z(1)*z(6);
dzdt(6) = -z(2) + a(1)*z(2)*z(5) + (a(1)*z(1)-a(2))*z(6);
dzdt = dzdt';
end
```