Muhideen Ogunlowo Homework 8

Setup

```
close all, clear all, format compact, format short g, clc
```

Exercise 4.5.3 a

For the form $f(x) = 0$, we can use a vector function f to represent both equations as:

$$f(u, v) = \begin{bmatrix} u * \log(u) + v * \log(v) + 0.3 \\ u^4 + v^2 - 1 \end{bmatrix}$$

In terms of $f(x) = 0$, we can rewrite $f(u, v)$ as

$$f(x_1, x_2) = \begin{bmatrix} x_1 * \log(x_1) + x_2 * \log(x_2) + 0.3 \\ {x_1}^4 + {x_2}^2 - 1 \end{bmatrix}$$

Therefore, the intersection of the curves is represented by the set of (x_1, x_2) for which $f(x_1, x_2) = 0$

Exercise 4.5.3 b

To find the Jacobian matrix of f we differentiate each component of f with respect to u and v. The Jacobian matrix J is given by:

$$J = \begin{bmatrix} \frac{\mathrm{df}_1}{\mathrm{du}} & \frac{\mathrm{df}_1}{\mathrm{dv}} \\ \frac{\mathrm{df}_2}{\mathrm{du}} & \frac{\mathrm{df}_2}{\mathrm{dv}} \end{bmatrix}$$

Where, $f_1 = u * \log(u) + v * \log(v) + 0.3$ and $f_2 = u^4 + v^2 - 1$

Differentiating the components:

$$\frac{\mathrm{df}_1}{\mathrm{du}} = \log(u) + 1, \frac{\mathrm{df}_1}{\mathrm{dv}} = \log(v) + 1, \frac{\mathrm{df}_2}{\mathrm{du}} = 4u^3, \frac{\mathrm{df}_2}{\mathrm{dv}} = 2v$$

The Jacobian matrix J is now,

$$J = \begin{bmatrix} \log(u) + 1, & \log(v) + 1 \\ 4u^3 & 2v \end{bmatrix}$$

In terms of $f(x) = 0$, we can rewrite J as

$$J = \begin{bmatrix} \log(x_1) + 1, & \log(x_2) + 1 \\ 4x_1^3 & 2x_2 \end{bmatrix}$$

Exercise 4.5.3 c

```
%Testing [1;0.1] with Newton's method
x0 = [1; 0.1];
x = newtonsys(@nlsys1,x0)  % with Newton's method
```

```
x = 2×6
          1      0.99501      0.99357      0.99351      0.99351      0.99351
        0.1      0.14971      0.16003      0.16038      0.16038      0.16038
```

```
x(:,end)
```

```
ans = 2×1
    0.99351
    0.16038
```

Exercise 4.5.3 d

Now with 0.1 and (function at end)

```
%Testing [0.1;1] with Newton's method
x0 = [0.1; 1];
x = newtonsys(@nlsys1,x0)   % Newton method
```

```
x = 2×6
        0.1      0.15342      0.16717       0.1679      0.16791      0.16791
          1      0.99984      0.99962       0.9996       0.9996       0.9996
```

```
x(:,end)
```

```
ans = 2×1
    0.16791
     0.9996
```

Exercise 4.6.1 a

For the form $f(x) = 0$, we can use a vector function f to represent both equations as:

$$f(u, v) = \begin{bmatrix} u * \log(u) + v * \log(v) + 0.3 \\ u^4 + v^2 - 1 \end{bmatrix}$$

In terms of $f(x) = 0$, we can rewrite $f(u, v)$ as

$$f(x_1, x_2) = \begin{bmatrix} x_1 * \log(x_1) + x_2 * \log(x_2) + 0.3 \\ {x_1}^4 + {x_2}^2 - 1 \end{bmatrix}$$

Therefore, the intersection of the curves is represented by the set of (x_1, x_2) for which $f(x_1, x_2) = 0$

Exercise 4.6.1 b

```
%Testing [1;0.1] with Levenberg's method
x0 = [1; 0.1];
x2 = levenberg(@nlsys2,x0)
```

```
x2 = 2×10
          1      0.99607      0.99447      0.99409      0.99358      0.99351 ···
        0.1       0.1074      0.13444      0.15471      0.15989      0.16037
```

```
x2(:,end) % Levenberg with default tolerances
```

```
ans = 2×1
      0.99351
      0.16038
```

Exercise 4.6.1 c

```
%Testing [0.1;1] with Levenberg's method
x0 = [0.1; 1];
x2 = levenberg(@nlsys2,x0);
x2(:,end) % Levenberg with default tolerances
```

```
ans = 2×1
      0.16791
       0.9996
```

Exercise 4.7.3 a

```
% Given data
t = [5, 10, 17, 22, 30, 50, 51, 90, 120, 180, 292, 395, 445, 775, 780, 700,
698, 880, 925, 800, 578, 400, 350, 202, 105, 65, 55, 40, 30, 20]'; % Deaths
per week
%Graph
x= linspace(1,30,30)';
figure
plot(x,t, 'o', 'linewidth', 2)
title ('data')
xlabel("weeks"), ylabel ("deaths")
A=900; B= 0.25; C=17;
p1=[900 .5 14]';
c3= levenberg(@(q)misfit(q,x,t),p1,1e-5)
```

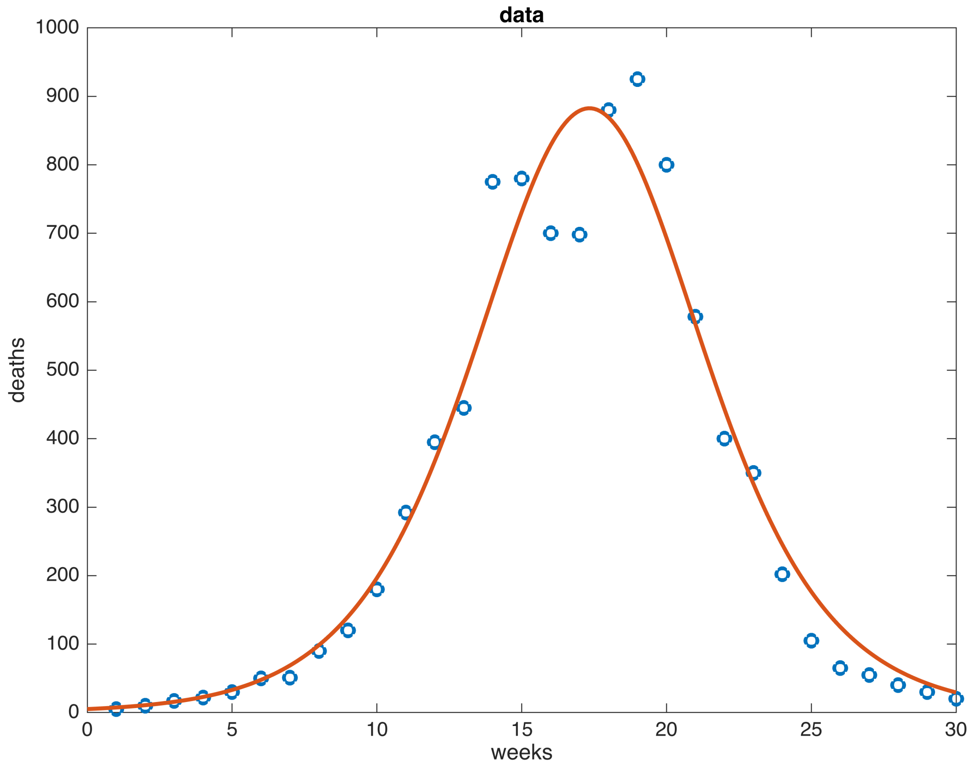
```
Warning: Iteration did not find a root.
c3 = 3×20
        900       847.08       780.89       800.32       882.22       882.03 ···
        0.5      0.18243      0.16047      0.16773      0.19043      0.19045
         14       14.759       17.138       17.488       17.276       17.312
```

```
A= c3(1,end), B=c3(2,end), C= c3(3,end)
```

```
A =
       882.65
B =
```

```
    0.18845
C =
    17.339
```

```
%model
model= @(t) A*(sech(B*(t-C)).^2);
%hold on
hold on
xx= linspace(0,30,301);
plot(xx, model(xx) , 'linewidth' ,2)
hold off %Hold off
```



Exercise 4.7.3 b

```
t12=[5, 10, 17, 22, 30, 50, 51, 90, 120, 180, 292, 395]'; %4.7.3 Part b
x= linspace(1,12,12)';
figure
plot(x,t12, 'o', 'linewidth', 2);
title ('data')
xlabel("weeks"), ylabel ("deaths")
A=900; B= 0.25; C=17;
p1=[900 .5 14]';
c3= levenberg(@(q)misfit(q,x,t12),p1,1e-5)
```

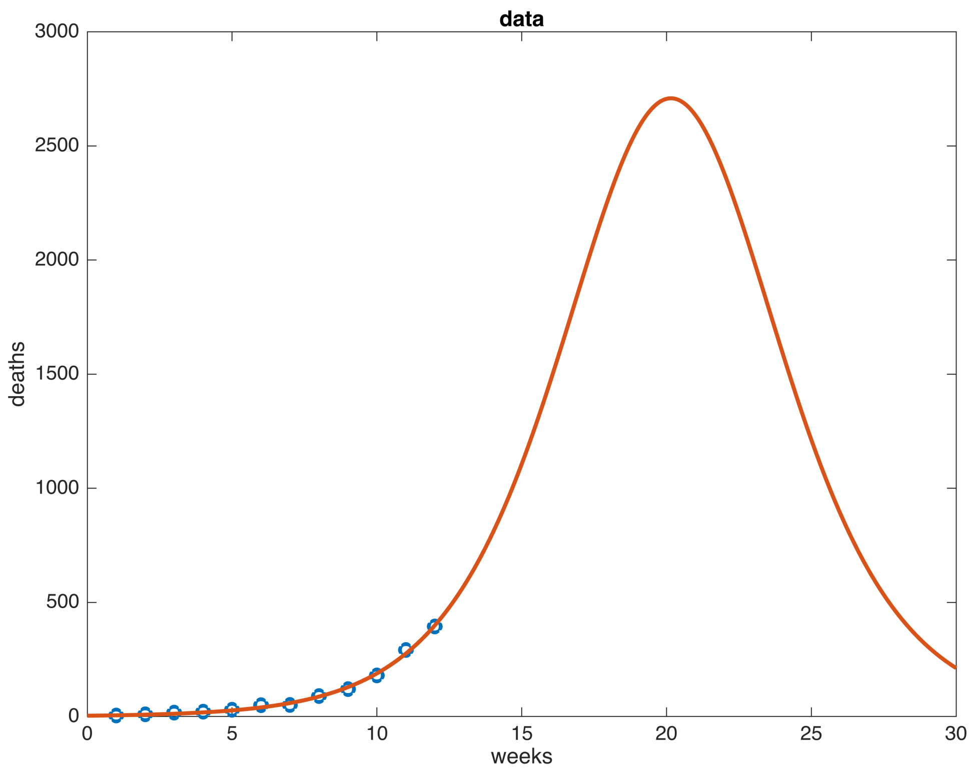
```
Warning: Iteration did not find a root.
c3 = 3×40
       900        900        900     900.03     900.03     900.03 ···
```

```
          0.5       0.26028        0.3938       0.28407       0.21405        0.2323
           14        14.714        14.741        15.264         16.21        16.232
```

```
A= c3(1,end), B=c3(2,end), C= c3(3,end)
```

```
A =
       2709.3
B =
      0.19759
C =
       20.153
```

```
model= @(t) A*(sech(B*(t-C)).^2);
hold on
xx= linspace(0,30,301);
plot(xx, model(xx) , 'linewidth' ,2)
hold off
```



This model is not a useful predictor of the value and timing of the death rate as it has a bad overestimation at the peak of death rates

Exercise 4.7.3 c

```
t13=[5, 10, 17, 22, 30, 50, 51, 90, 120, 180, 292, 395, 445]'; %4.7.3 Part b
x= linspace(1,13,13)';
figure
title ('data')
plot(x,t13, 'o' , 'linewidth' ,2);
```

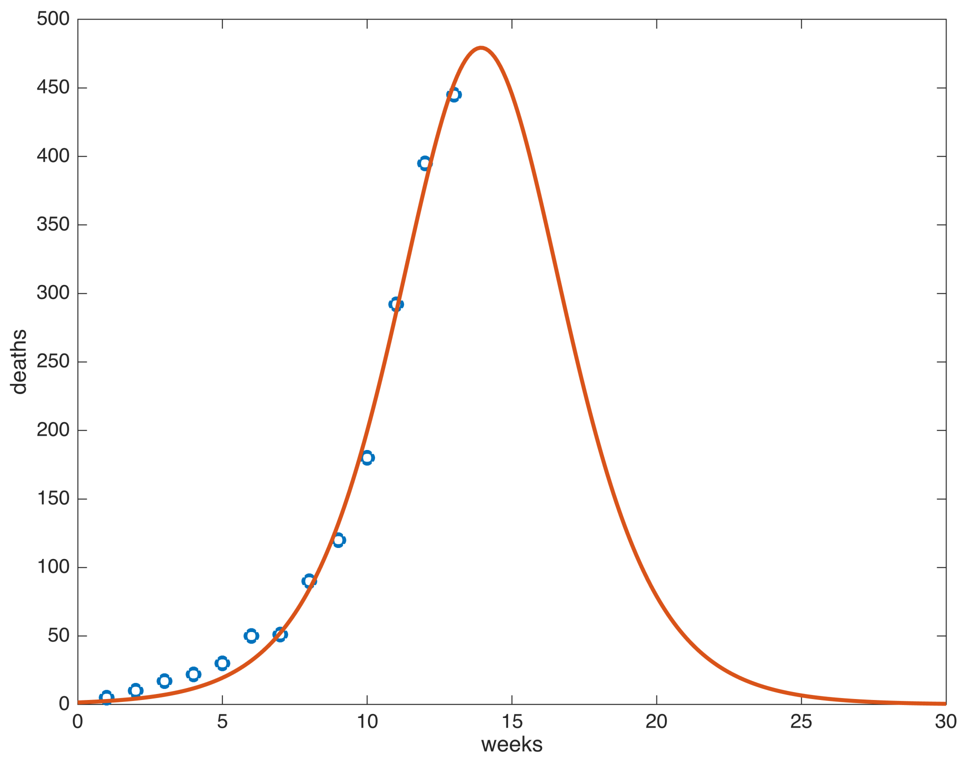
```
xlabel("weeks"), ylabel ("deaths")
A=900; B= 0.25; C=17;
p1=[900 0.25 17]';
c3= levenberg(@(q)misfit(q,x,t13),p1,1e-5)
```

```
Warning: Iteration did not find a root.
c3 = 3×23
         900          900          900       899.89       899.76       899.43 ···
        0.25      0.16683      0.19655      0.19402      0.18902      0.18938
          17       17.582       17.275       17.307       17.446       17.438
```

```
A= c3(1,end), B=c3(2,end), C= c3(3,end)
```

```
A =
       479.35
B =
      0.25559
C =
       13.933
```

```
model= @(t) A*(sech(B*(t-C)).^2);
hold on
xx= linspace(0,30,301);
plot(xx, model(xx), 'linewidth' ,2)
hold off
```



This model is not a useful predictor of the value and timing of the death rate as it has a bad underestimation at the peak of death rates

Functions used

Function used for Exercise 4.7.3

```
function [f]= misfit(p,x,y) %Function for 4.7.3
    A= p(1);
    B= p(2);
    C= p(3);
    f= A*(sech(B*(x-C)).^2)-y;
end
```

Function used to define residual function and Jacobian for Exercise 4.5.3

```
function [F, J] = nlsys1(x)
    u = x(1);
    v = x(2);

    % Residual function
    F = [u * log(u) + v * log(v) + 0.3;
         u^4 + v^2 - 1];

    % Jacobian
    J = [log(u) + 1, log(v) + 1;
         4*u^3, 2*v];
end
```

Function used to define residual function for Exercise 4.6.1

```
function [F] = nlsys2(x)
 u = x(1);
 v = x(2);
 %residual function
F = [u * log(u) + v * log(v) + 0.3;
         u^4 + v^2 - 1];
end
```

Newton Function for Exercise 4.5.3

```
function x = newtonsys(f,x1)
% NEWTONSYS   Newton's method for a system of equations.
% Input:
%   f        function that computes residual and Jacobian matrix
%   x1       initial root approximation (n-vector)
% Output
%   x        array of approximations (one per column, last is best)

% Operating parameters.
```

```
funtol = 1000*eps;  xtol = 1000*eps;  maxiter = 40;

x = x1(:);
[y,J] = f(x1);
dx = Inf;
k = 1;

while (norm(dx) > xtol) && (norm(y) > funtol) && (k < maxiter)
  dx = -(J\y);   % Newton step
  x(:,k+1) = x(:,k) + dx;

  k = k+1;
  [y,J] = f(x(:,k));
end

if k==maxiter, warning('Maximum number of iterations reached.'), end

end
```

Levenberg Function

```
function x = levenberg(f,x1,tol)
% LEVENBERG   Quasi-Newton method for nonlinear systems.
% Input:
%   f         objective function
%   x1        initial root approximation
%   tol       stopping tolerance (default is 1e-12)
% Output
%   x         array of approximations (one per column)

% Operating parameters.
if nargin < 3, tol = 1e-12; end
ftol = tol;  xtol = tol;  maxiter = 40;

x = x1(:);     fk = f(x1);
k = 1;  s = Inf;
Ak = fdjac(f,x(:,1),fk);   % start with FD Jacobian
jac_is_new = true;
I = eye(length(x));

lambda = 10;
while (norm(s) > xtol) && (norm(fk) > ftol) && (k < maxiter)
    % Compute the proposed step.
    B = Ak'*Ak + lambda*I;
    z = Ak'*fk;
    s = -(B\z);

    xnew = x(:,k) + s;   fnew = f(xnew);
```

```matlab
    % Do we accept the result?
    if norm(fnew) < norm(fk)    % accept
        y = fnew - fk;
        x(:,k+1) = xnew;  fk = fnew;
        k = k+1;

        lambda = lambda/10;  % get closer to Newton
        % Broyden update of the Jacobian.
        Ak = Ak + (y-Ak*s)*(s'/(s'*s));
        jac_is_new = false;
    else                       % don't accept
        % Get closer to steepest descent.
        lambda = lambda*4;
        % Re-initialize the Jacobian if it's out of date.
        if ~jac_is_new
            Ak = fdjac(f,x(:,k),fk);
            jac_is_new = true;
        end
    end
end

if (norm(fk) > 1e-3),  warning('Iteration did not find a root.'),  end

end
```