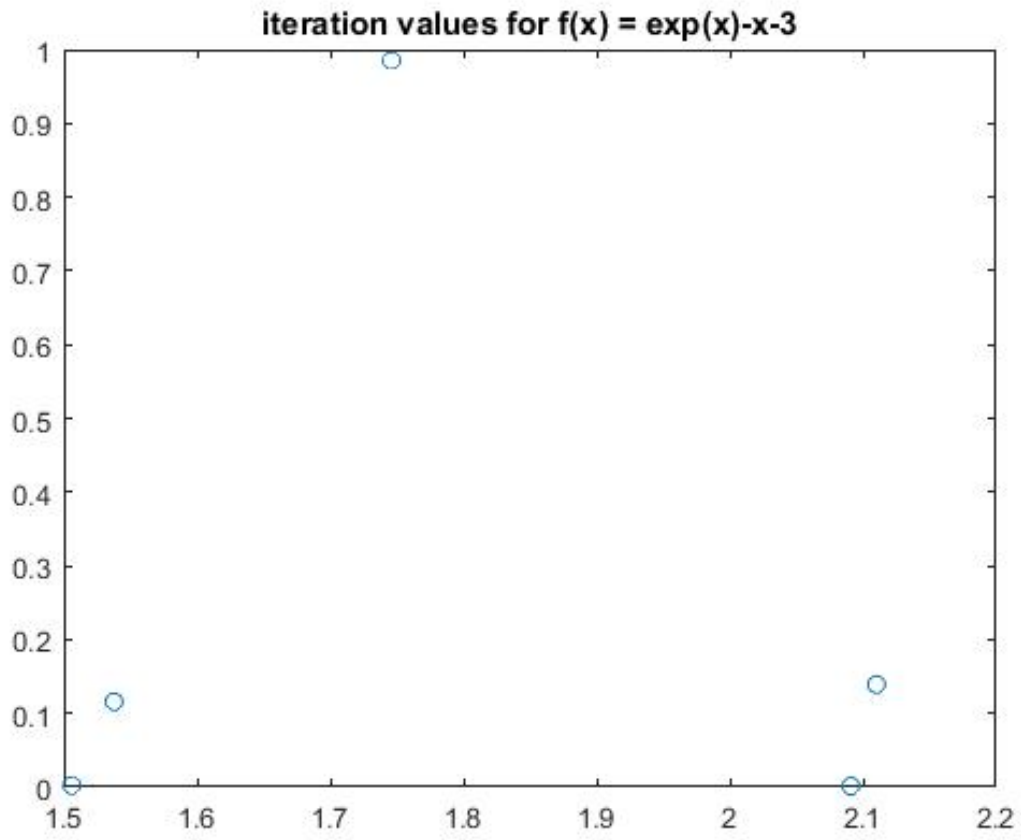
GEOG210A Assignment Week3

Lily Cheng

**Question1**

(a)

```
f = @(x)exp(x)-x-3;
initial_guess = 1;
x = initial_guess;

y = f(x); %call function to evaluate initial value
itc = 0; %iteration counter
while(abs(y) > 1e-2) %that means 10^-2; as long as fx hasn't reached 0.01
    y = f(x);
    dx = 1e-3;
    fup = f(x+dx);
    fdown = f(x-dx);
    dfdx = (fup-fdown)/(2*dx);

    dif = -f(x)./dfdx;
    x = x+dif;

    itc = itc+1;

    xi(itc) = x;
    yi(itc) = f(x);
    y = f(x);

end;
% equation has two roots x1 = 1.5059, y1 = 0.0023; x2 = 2.0909, y2 =
% 0.0015.
figure(1);
plot(xi,yi,'o')
title('iteration values for f(x) = exp(x)-x-3')
```



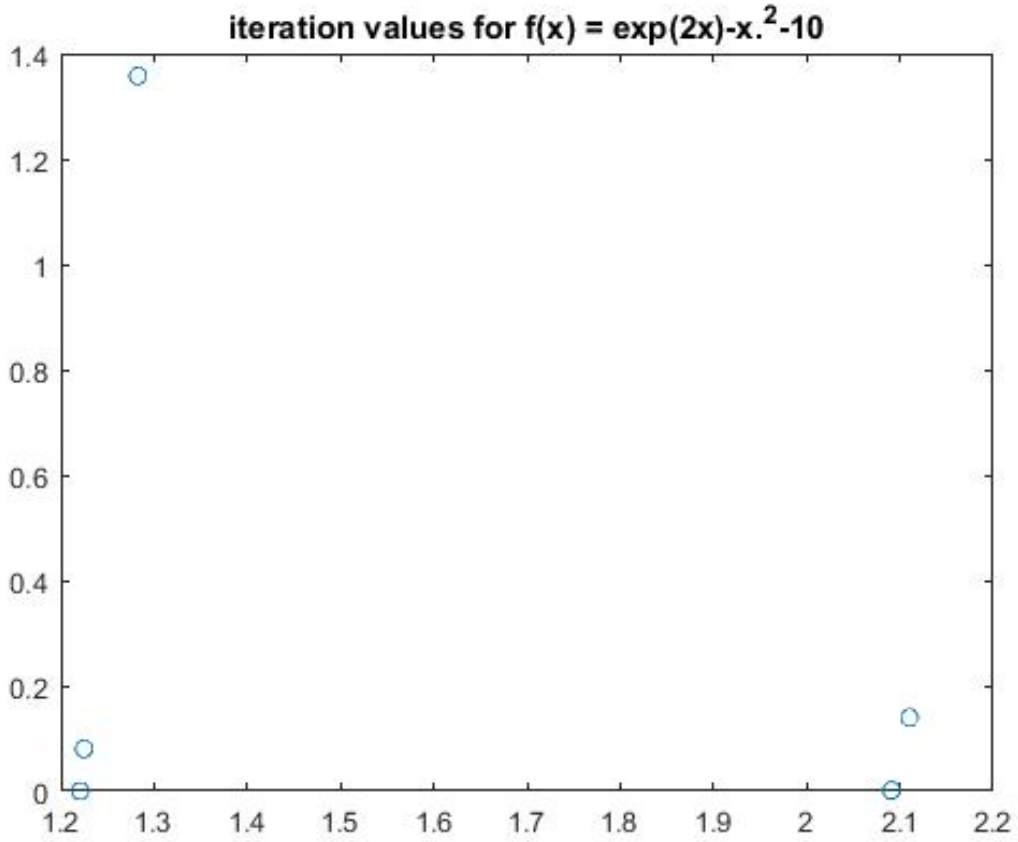

**Plot#1**

Roots of equation exp(x)-x-3') = 0 are x1 = 1.5059, x2 = 2.0909.

(b)

```
f = @(x)exp(2.*x)-x.^2-10;
initial_guess = 1;
x = initial_guess;

y = f(x); %call function to evaluate initial value
itc = 0; %iteration counter
while(abs(y) > 1e-2) %that means 10^-2; as long as fx hasn't reached 0.01
    y = f(x);
    dx = 1e-3;
    fup = f(x+dx);
    fdown = f(x-dx);
    dfdx = (fup-fdown)/(2*dx);

    dif = -f(x)./dfdx;
    x = x+dif;

    itc = itc+1;

    xi(itc) = x;
    yi(itc) = f(x);
```

```
    y = f(x);

end;
% equation has two roots x1 = 1.2208, y1 = 0.0003; x2 = 2.0909, y2 =
% 0.0015.
figure(1);
plot(xi,yi,'o')
title('iteration values for f(x) = exp(2x)-x.^2-10')
```



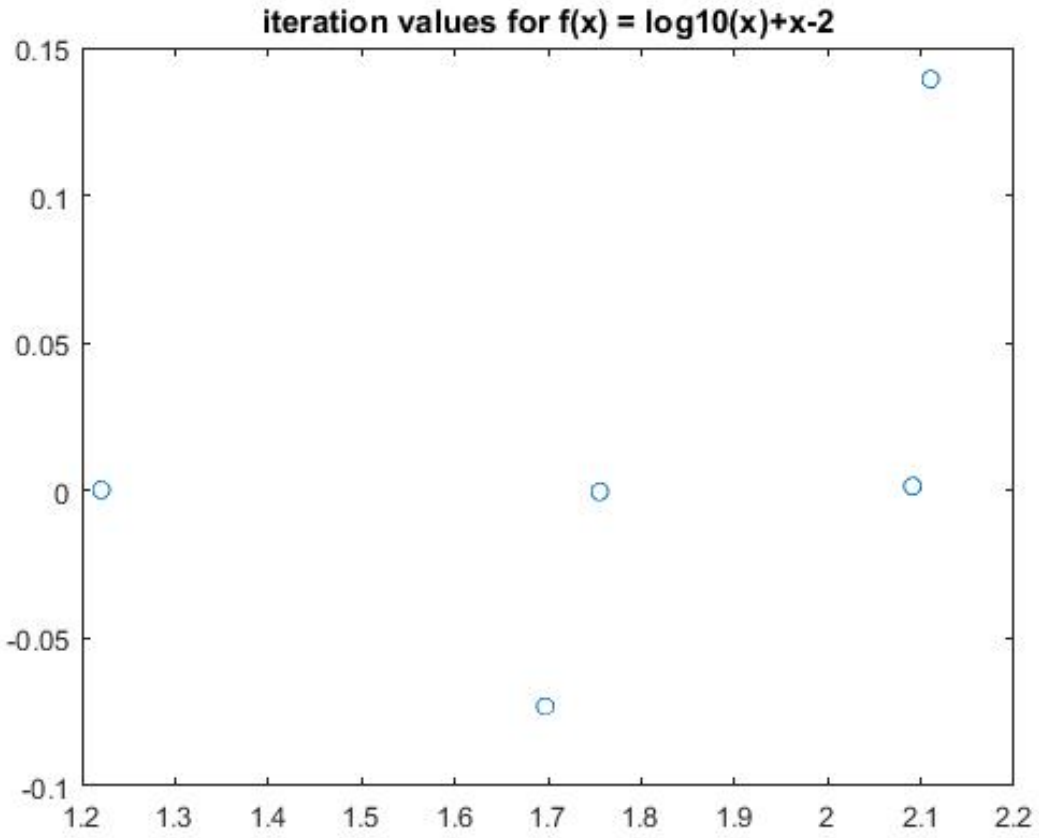

**Plot#2**

Roots of equation exp(2x)-x.^2-10') = 0 are x1 = 1.5059, x2 = 2.0909.

(c)

```
f = @(x)log10(x)+x-2;

initial_guess = 1;
x = initial_guess;

y = f(x); %call function to evaluate initial value
itc = 0; %iteration counter
while(abs(y) > 1e-2) %that means 10^-2; as long as fx hasn't reached 0.01
   y = f(x);
```

```
    dx = 1e-3;
    fup = f(x+dx);
    fdown = f(x-dx);
    dfdx = (fup-fdown)/(2*dx);

    dif = -f(x)./dfdx;
    x = x+dif;

    itc = itc+1;

    xi(itc) = x;
    yi(itc) = f(x);

    y = f(x);

end;
% equation has two roots x1 = 1.7554, y1 = -0.0002; x2 = 2.0909, y2 =
% 0.0015.
figure(1);
plot(xi,yi,'o')
title('iteration values for f(x) = log10(x)+x-2')
```

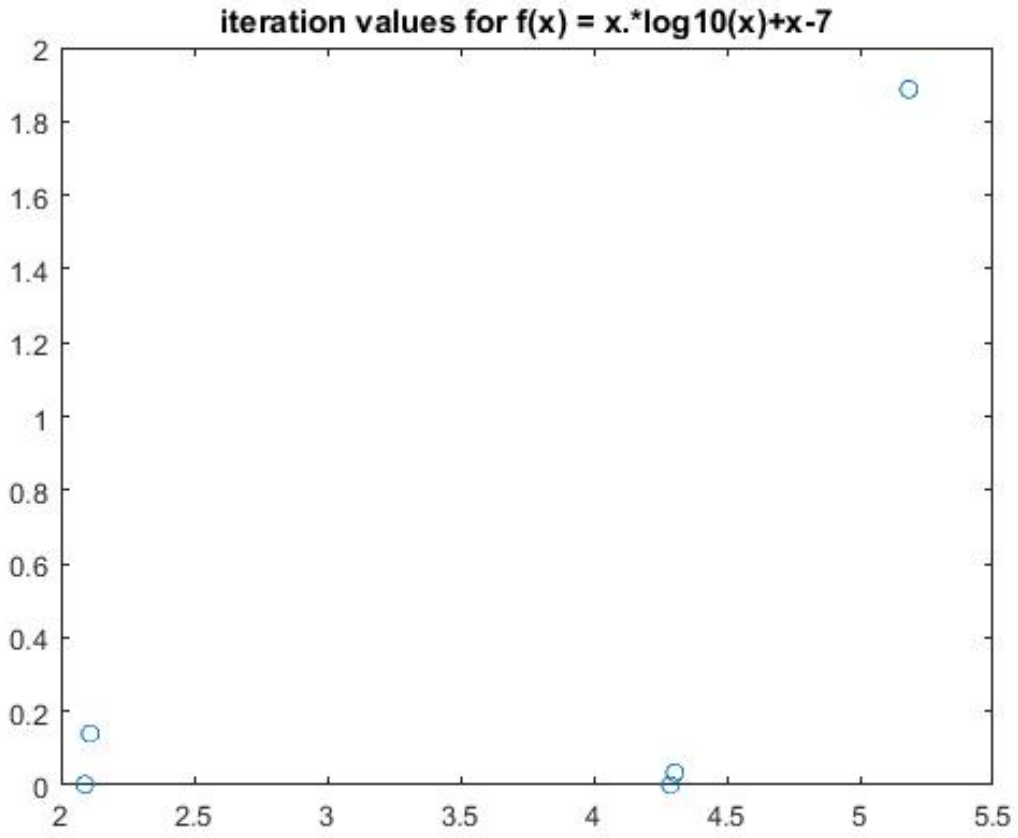



**Plot#3**

Roots of equation log10(x)+x-2') = 0 are x1 = 1.7554, x2 = 2.0909.

(d)

```
f = @(x)x.*log10(x)+x-7;
initial_guess = 1;
x = initial_guess;

y = f(x); %call function to evaluate initial value
itc = 0; %iteration counter
while(abs(y) > 1e-2) %that means 10^-2; as long as fx hasn't reached 0.01
    y = f(x);
    dx = 1e-3;
    fup = f(x+dx);
    fdown = f(x-dx);
    dfdx = (fup-fdown)/(2*dx);

    dif = -f(x)./dfdx;
    x = x+dif;

    itc = itc+1;

    xi(itc) = x;
    yi(itc) = f(x);

    y = f(x);

end;
% equation has two roots x1 = 4.2884, y1 = 0; x2 = 2.0909, y2 =
% 0.0015.
figure(1);
plot(xi,yi,'o')
title('iteration values for f(x) = x.*log10(x)+x-7')
```



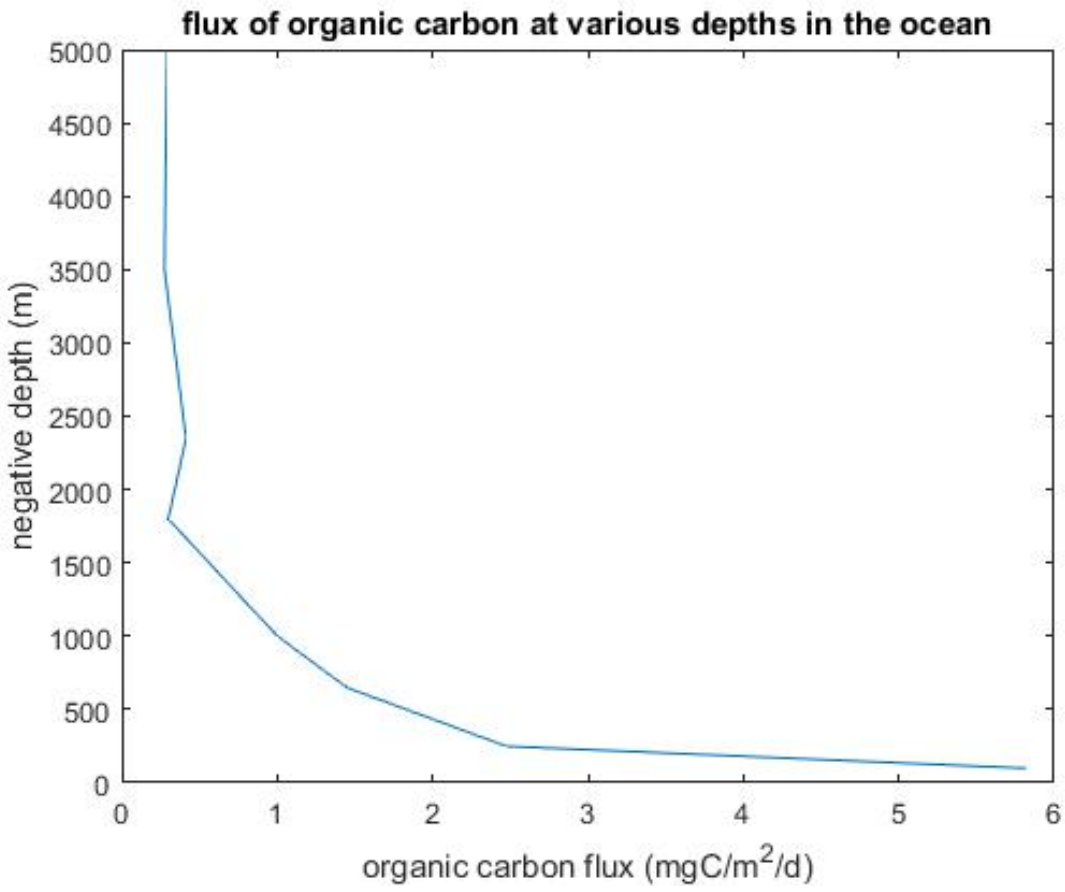

**Plot #4**

Roots of equation log10(x)+x-2') = 0 are x1 = 4.2884, x2 = 2.0909.

## Question2

**(a)** Plot the organic carbon flux as a function of depth. Depth is typically plotted on the yaxis with negative depths representing depths below the ocean surface.

```
depth = [100, 250,650,1000,1800, 2350,3500,5000];%depth (m)
flux = [5.8166, 2.4736, 1.4481, 1.0013, 0.2982, 0.4126, 0.2739, 0.2897];%(mgC/m^2/d)
figure(2);
plot(flux, depth, '-');
xlabel('organic carbon flux (mgC/m^2/d)');
ylabel('negative depth (m)');
title('flux of organic carbon at various depths in the ocean');
```



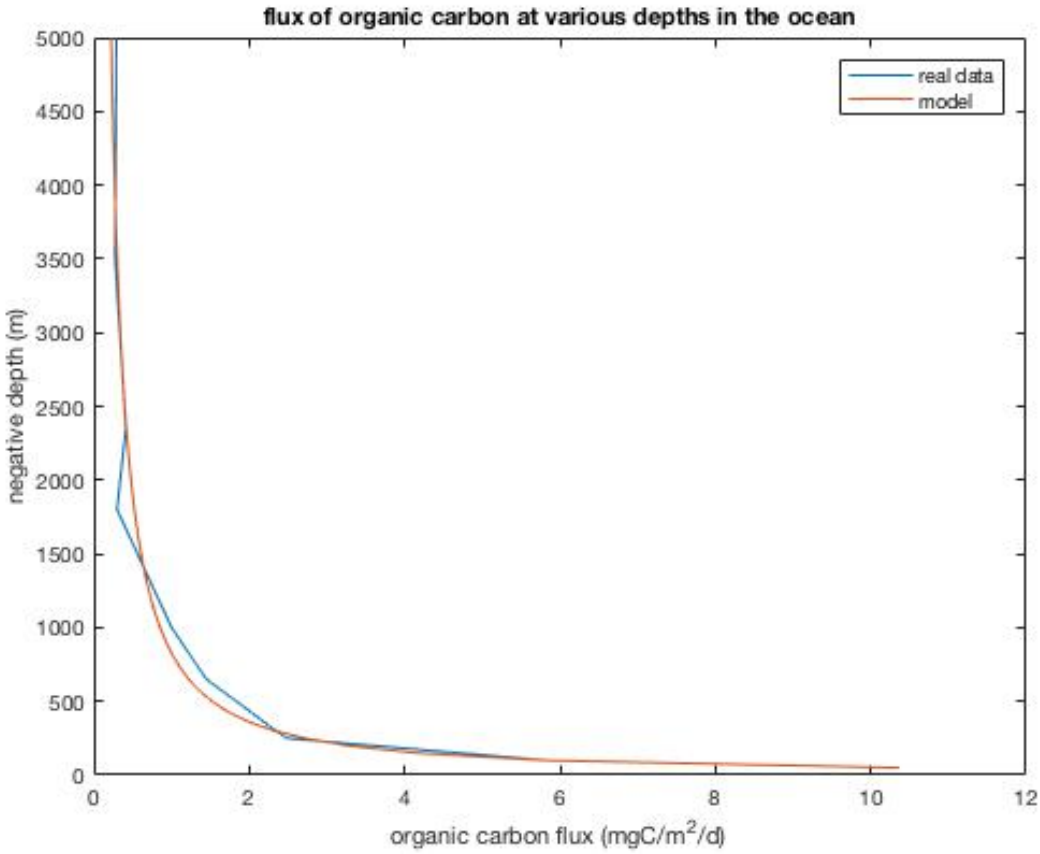

**Plot #5**

(b) Assuming $z_o$ = 100 m, find the value of b that minimizes the misfit (in the leastsquaressense) between the flux predicted by a power-law model of the form $F(z) = F(z_o)(z/z_o)^{-b}$, and the observed flux from the sediment traps. (Use the procedureoutlined in class to set up a "cost function" that measures the model-data misfit, anddetermine the minimum of the cost function using Newton's method on the costfunction derivative. Write your own MATLAB script and include it in your submittedassignment.) Use a starting guess of b = 0.1. Plot the model-predicted carbon flux for the optimal value of b in the plot you created for (a).

```
F(Zo) = 5.8166;
y = @(b)(F(Zo).*((x/Zo).^(-b))); % model
% the cost function
cost = @(b) (sum(y(b)-yobs).^2);

initial_guess = 0.1;
b = initial_guess;
c = cost(b); %call function to evaluate initial value
itc = 0; %iteration counter
while(abs(c) > 1e-2) %that means 10^-2; as long as fx hasn't reached 0.01
    c = cost(b);
    db = 1e-3;
    cup = cost(b+db);
    cdown = cost(b-db);
    dfdb = (cup-cdown)/(2*db);

    dif = -cost(b)./dfdb;
    b = b+dif;

    itc = itc+1;

    bi(itc) = b;
    ci(itc) = cost(b);
    dfdbi(itc) = (cost(b+db)-cost(b-db))./(2.*db);
```

```
    c = cost(b);

end;
% bimin = 0.8333 ci = 0.0031
```

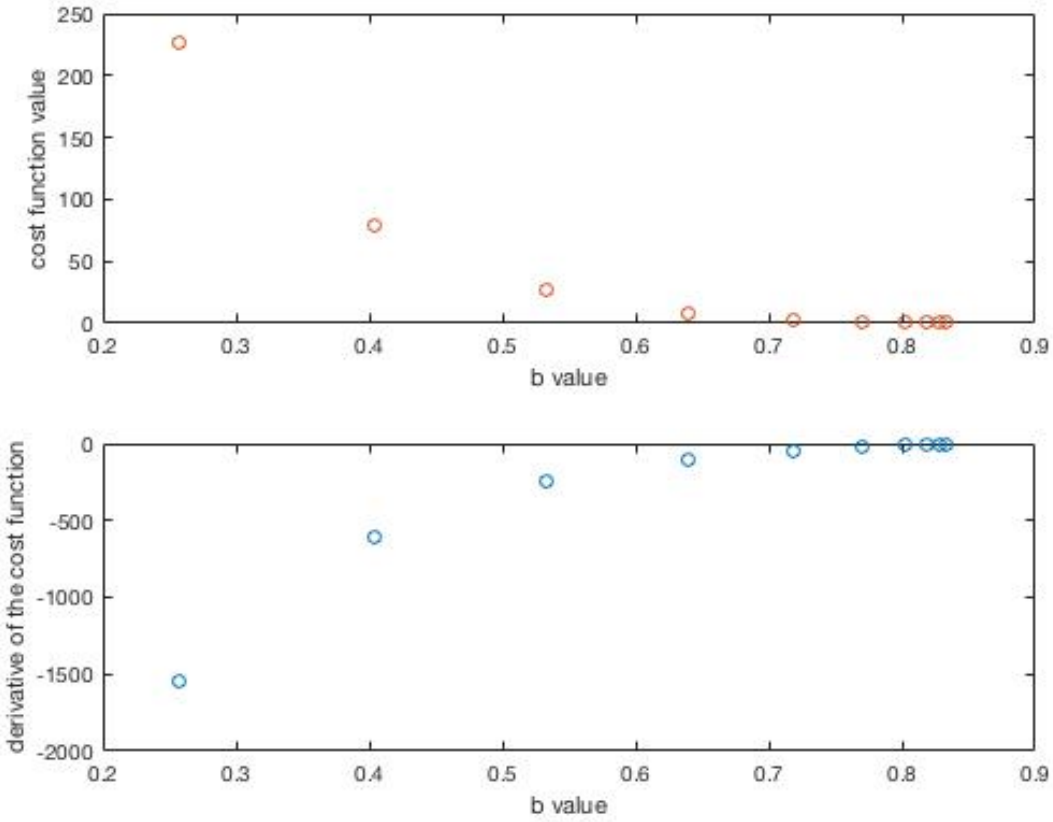
I switched to matlab 2017b so the script looks a little different from previous ones. The b value that minimizes the misfit between the flux predicted by the power-law model and the observed flux is 0.8333.




**Plot #6**

c) Use subplot to create 2 plots windows. In the upper window, plot the value of b and the cost function value at each Newton iteration. In the lower window, plot the value of b and the derivative of the cost function at each Newton iteration.

```
figure(3);
subplot(2,1,1);
plot(bi,ci,'o');
xlabel('b value');
ylabel('cost function value');
hold on
subplot(2,1,2);
plot(bi,dfdbi,'o');
xlabel('b value');
ylabel('derivative of the cost function')
```



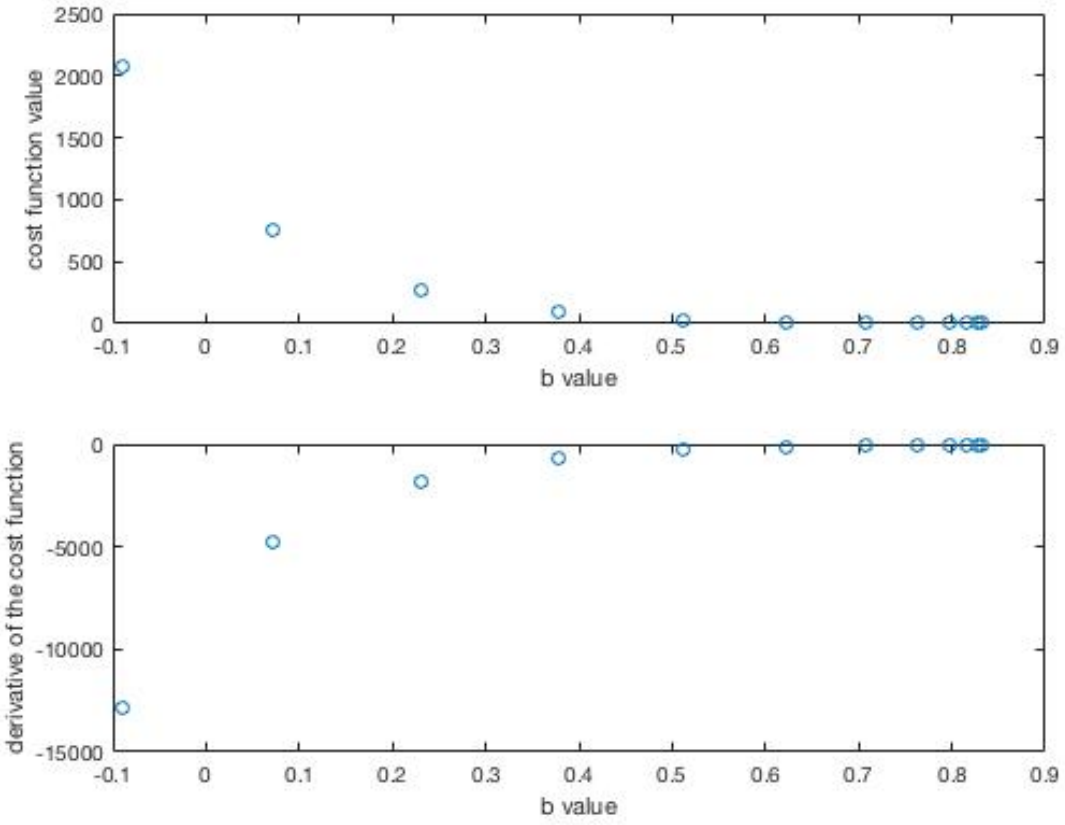

**Plot #7**

d) Now repeat (b) and (c) using a different initial guess for b (try something >1).

Set the initial guess for b to be 2.1

```
depth = [100, 250,650,1000,1800, 2350,3500,5000];%depth (m)
flux = [5.8166, 2.4736, 1.4481, 1.0013, 0.2982, 0.4126, 0.2739, 0.2897];%(mgC/m^2/d)

yobs = flux;
x = depth;

Zo = 100;
F(Zo) = 5.8166;
y = @(b)(F(Zo).*((x/Zo).^(-b))); % model
% the cost function
cost = @(b) (sum(y(b)-yobs).^2);

initial_guess = 2.1;
b = initial_guess;
c = cost(b); %call function to evaluate initial value
itc = 0; %iteration counter
while(abs(c) > 1e-2) %that means 10^-2; as long as fx hasn't reached 0.01
    c = cost(b);
    db = 1e-3;
    cup = cost(b+db);
    cdown = cost(b-db);
    dfdb = (cup-cdown)/(2*db);

    dif = -cost(b)./dfdb;
    b = b+dif;

    itc = itc+1;

    bi(itc) = b;
    ci(itc) = cost(b);
    dfdbi(itc) = (cost(b+db)-cost(b-db))./(2.*db);

    c = cost(b);
```

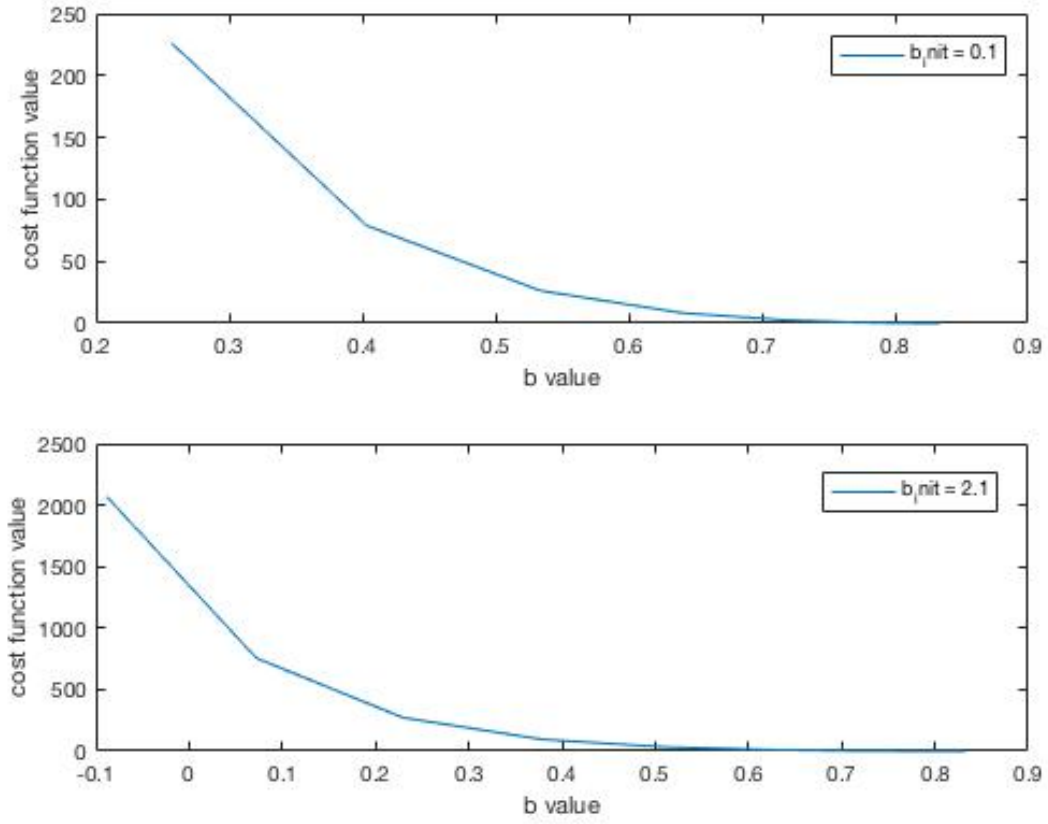
```
end;
% bimin = 0.8326 ci = 0 b = 2.1

%2d(2)

figure(3);
subplot(2,1,1);
plot(bi,ci,'o');
xlabel('b value');
ylabel('cost function value');
hold on
subplot(2,1,2);
plot(bi,dfdbi,'o');
xlabel('b value');
ylabel('derivative of the cost function')
```




**Plot #8**

(e) Compare the convergence of Newton's method in (b) and (d). Did it converge to the same answer? Which one converged more quickly and why?

Yes, they almost converged to the same answer. In order to tell which convergence is more quickly, I plot the lower subplot of (b) and (d) in one plot:




**Plot #9**

From the plot, we could tell that the smaller the initial guess of b, the quicker the convergence is because it is closer to the minimum value.