

Sam Snarr
MATH 448 Final Exam
4/25/19

1.

(a)

% Golden Search minimization algorithm

```
function ret=goldsearch(f, a, b)
n=0;
phi = (1+sqrt(5))/2;
tol = 1e-10;

c=b-(b-a)/phi;
d=a+(b-a)/phi;
while (abs(a-b)>tol && n<50000)

    fc = f(c);
    fd = f(d);
    if fc<fd
        b=d; d=c;
        c = b-(b-a)/phi;
        fd=fc;
    elseif fc>fd
        a=c; c=d;
        d=a+(b-a)/phi;
    end
    fprintf('%10.15f, %10.15f, %10.15f \n', a, b, abs(a-b))
    n=n+1;
end

ret=d;
truemin = 2.387425886722793;

fprintf('\nthe approximate value of the minimum is %0.10f with true value %0.10f\n', d, truemin)
fprintf('error %0.10f\n', abs(d-truemin))
```

(b)

Matlab code gives
>> goldsearch(g, 1, 4)

the approximate value of the minimum is 2.3874258866 with true value 2.3874258867
error 0.0000000001

Calculus gives
2.387425886722793

2.

(a)

>> x1=(0.4+0.9i)^0;

Durand Method

```
>> x1=(0.4+0.9i)^0;
>> x2=(0.4+0.9i)^1;
>> x3=(0.4+0.9i)^2;
>> x4=(0.4+0.9i)^3;

>> a=0;b=0;c=-1;d=-1;
First iteration
>> durand
1.086422045817177 - 0.251514793202897i
-0.265722915469425 + 0.048908509685217i
-0.128345843587953 - 0.312879109981729i
-0.692353286759799 + 0.515485393499408i
```

After running durand 10 times the approximate roots are:

```
>> durand
1.220744084605760 - 0.000000000000000i
-0.724491959000516 + 0.000000000000000i
-0.248126062802622 - 1.033982060975968i
-0.248126062802622 + 1.033982060975968i
```

Aberth's Method

```
First iteration
>> z1=1i; z2=3i; z3=5i; z4=7i;
>> z2=z2-(df(z2)/f(z2)-1/(z2-z1)-1/(z2-z3)-1/(z2-z4))^-1;
>> z3=z3-(df(z3)/f(z3)-1/(z3-z1)-1/(z3-z2)-1/(z3-z4))^-1;
>> z4=z4-(df(z4)/f(z4)-1/(z4-z1)-1/(z4-z2)-1/(z4-z3))^-1;
>> z1=z1-(df(z1)/f(z1)-1/(z1-z4)-1/(z1-z2)-1/(z1-z3))^-1;
>> disp([z1 z2 z3 z4]')

-0.192673350273651 - 0.894396677727069i
-0.014889794922061 - 2.374796428585050i
-0.005913350896374 - 3.508227265037532i
-0.007313998314218 -17.246445778349297i
```

```
10th iteration
>> z2=z2-(df(z2)/f(z2)-1/(z2-z1)-1/(z2-z3)-1/(z2-z4))^-1;
>> z3=z3-(df(z3)/f(z3)-1/(z3-z1)-1/(z3-z2)-1/(z3-z4))^-1;
>> z4=z4-(df(z4)/f(z4)-1/(z4-z1)-1/(z4-z2)-1/(z4-z3))^-1;
>> z1=z1-(df(z1)/f(z1)-1/(z1-z4)-1/(z1-z2)-1/(z1-z3))^-1;
>> disp([z1 z2 z3 z4]')
-0.248126062802622 - 1.033982060975968i
1.220744084605760 - 0.000000000000000i
-0.248126062802622 + 1.033982060975968i
-0.724491959000516 + 0.000000000000000i
```

Plugging in the previous roots into the function will give the y values as well...

Matlab's roots() command gives

```
>> roots([a, b, c, d, e])
ans =
1.220744084605760 + 0.000000000000000i
-0.248126062802622 + 1.033982060975968i
```

-0.248126062802622 - 1.033982060975968i
-0.724491959000515 + 0.000000000000000i

So yes they match up correctly.

(b)

%Higher Dimension Newton Minimization

```
>> x=[1;2];  
>> y=JM(x)\(-FM(x))  
y =  
    1.0e-10 *  
    -0.161018297136683  
    -0.128691347485762
```

```
>> x=x+y  
x =  
    1.220744084605760  
    1.490216120099954
```

```
>> x=[-2;-4];  
>> y=JM(x)\(-FM(x))  
y =  
    1.0e-06 *  
    0.164011640443312  
    0.155864980652063
```

```
>> x=x+y  
x =  
   -0.724491959000536  
    0.524888598656408
```

```
>> x=[2; 3i];  
>> y=JM(x)\(-FM(x))  
y =  
    1.0e-16 *  
   -0.329299250518749 + 0.486602944217349i  
   -0.842861977247513 + 0.187766243773091i  
>> x=x+y  
x =  
   -0.248126062802622 + 1.033982060975968i  
  -1.007552359378179 - 0.513115795597015i
```

```
>> x=[2;-3i]  
>> y=JM(x)\(-FM(x))  
y =  
    1.0e-15 *  
   -0.058590373495930 - 0.003641105572739i  
    0.021545921698697 + 0.122969696661371i  
x=x+y  
x =  
   -0.248126062802622 - 1.033982060975968i  
  -1.007552359378179 + 0.513115795597015i
```

(c)

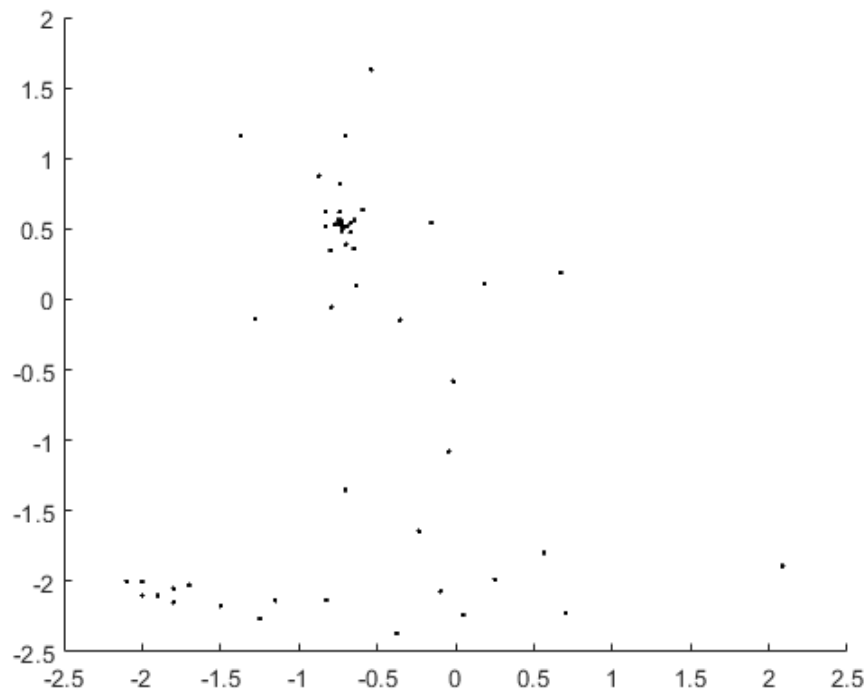
Because at a minimum both of the equations must be zero since both parts of the function are squared and therefore are never negative in R.

(d)

```
>> fminsearch(@f, [-2, -2], opt)
```

ans =

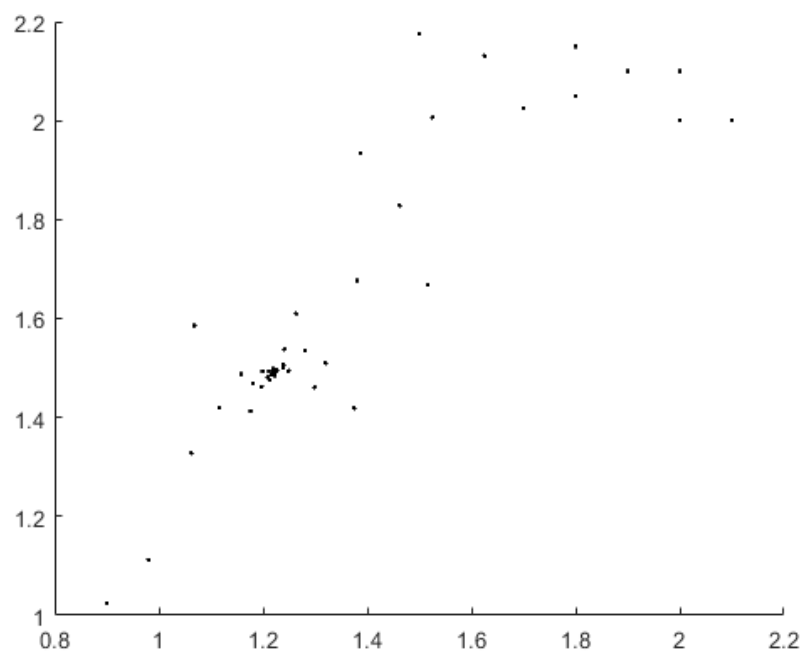
-0.724491959045917 0.524888598633890



```
>> fminsearch(@f, [2, 2], opt)
```

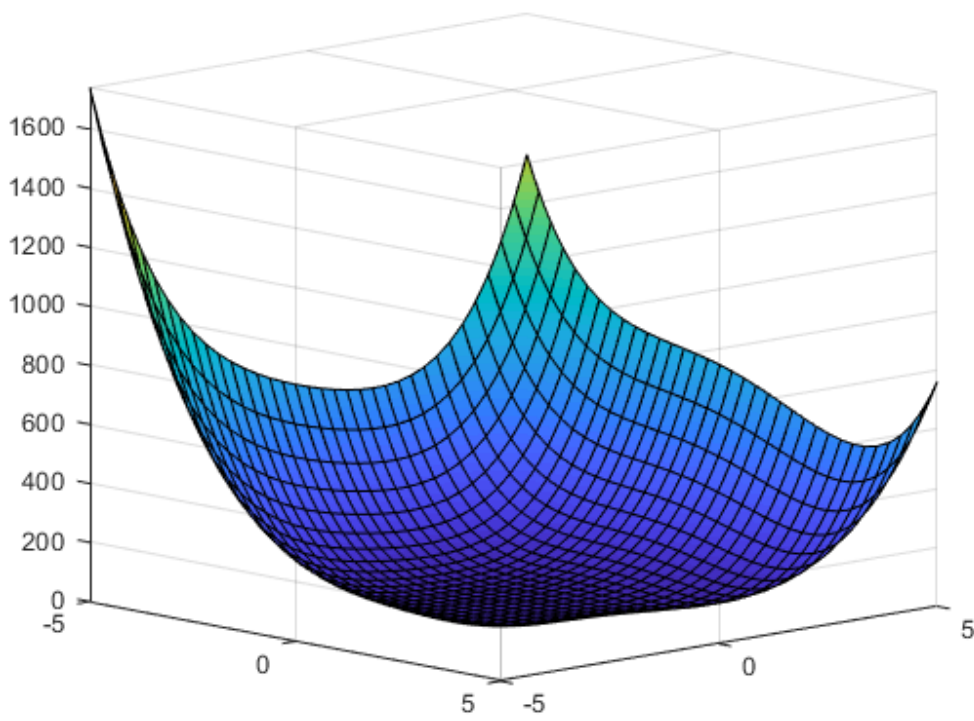
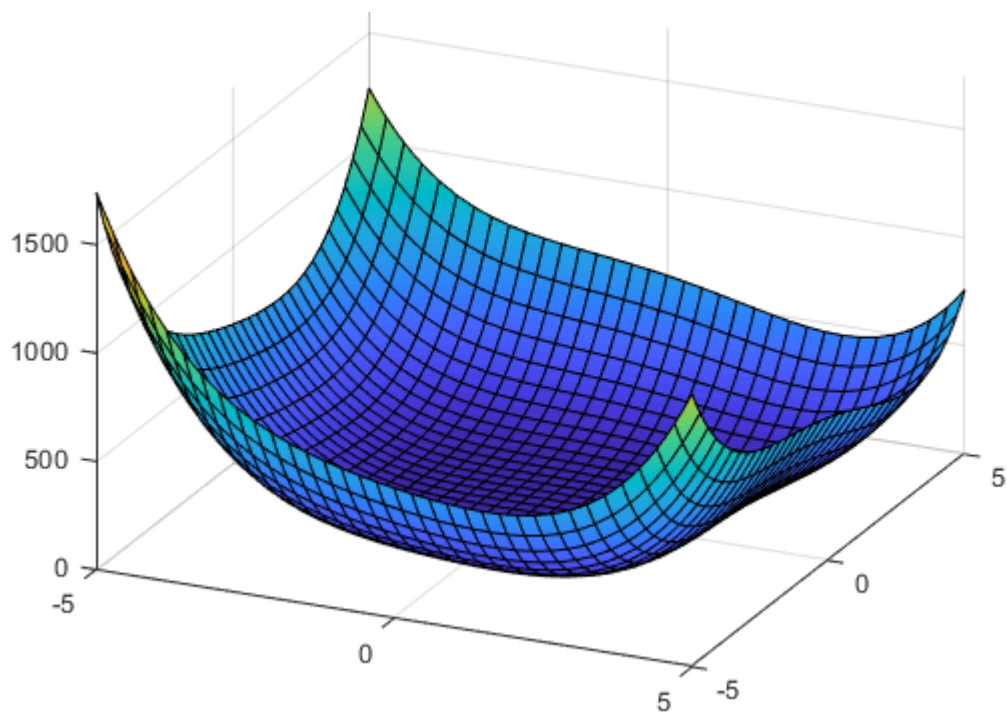
ans =

1.220744084620451 1.490216120151177



The two real solutions came out using fminsearch as expected. The points are converging to the minimum. Method is good since there is a long flat valley.

```
>> fsurf(f2)
```



(e)

% looking for the 1st root using Steepest Descent

```
>> x=[3,3];
>> steepDescent
a =
    0.0625000000000000
x =
    1.3750000000000000    0
>> steepDescent
a =
    0.186509957852831
x =
   -0.480628369731098    0.705240778131018
>> steepDescent
a =
```

```

0.303357691656985
x =
-0.632270146828918  0.436346598390595
>> steepDescent
a =
0.2500000000000000
x =
-0.732500452482564  0.495434064963469
>> steepDescent
a =
0.247199937904922
x =
-0.728506754597176  0.526564533556525

```

% now looking for the the next root

```

>> x=[1,1];
>> steepDescent
a =
0.0625000000000000
x =
0.875000000000000  1.250000000000000
>> steepDescent
a =
0.181661484165570
x =
0.915448064833740  1.357861506223307
>> steepDescent
a =
0.087680117595272
x =
0.986329491533330  1.300833111479657
>> steepDescent
a =
0.073275439859802
x =
0.990629409471990  1.364923611954040
>> steepDescent
a =
0.201004008981486
x =
1.092084052032483  1.350767644581195
>> steepDescent
a =
0.048667475087922
x =
1.082853738026191  1.405720023680523
>> steepDescent
a =
0.467994011420385
x =
1.219189772050151  1.468550949557925
>> steepDescent
a =
0.039645925042302
x =

```

```

1.212502443670108 1.484534779074316
>> steepDescent
a =
0.249229839560836
x =
1.216872879854062 1.490185503604595
>> steepDescent
a =
0.045150306798016
x =
1.218247773278352 1.488318848414849
>> steepDescent
a =
0.111386967258575
x =
1.218682439781968 1.489477144106549
>> steepDescent
a =
0.062370609640723
x =
1.219317145752969 1.488994099152432

```

3.

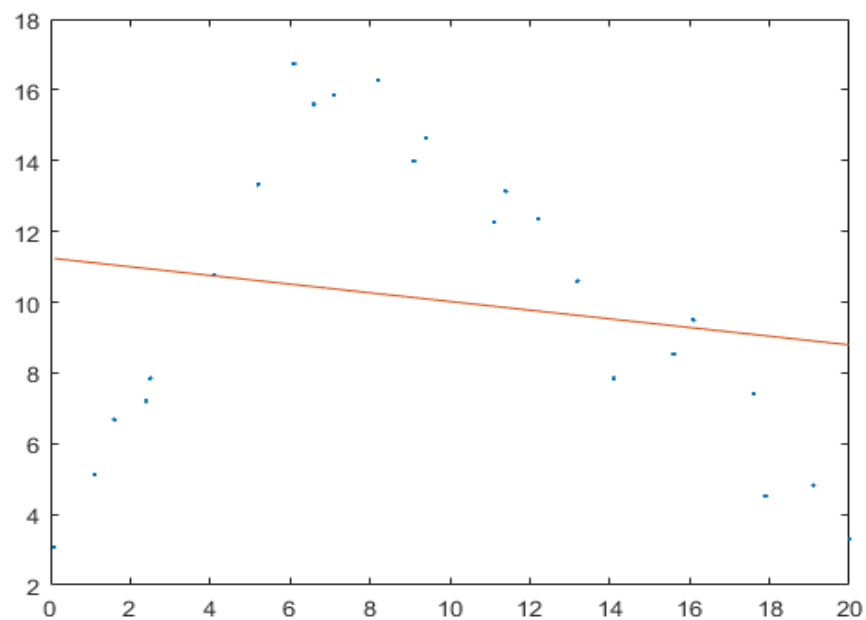
% Linear

```

>> A=[length(x), sum(x);
sum(x), sum(x.^2)];
>> b=[sum(y);
sum(x.*y)]
>> c=A\b
c =
11.244642801208141
-0.122699427217409
>> sum((y-(x*c(2)+c(1))).^2)
S =
4.172865272092502e+02

```

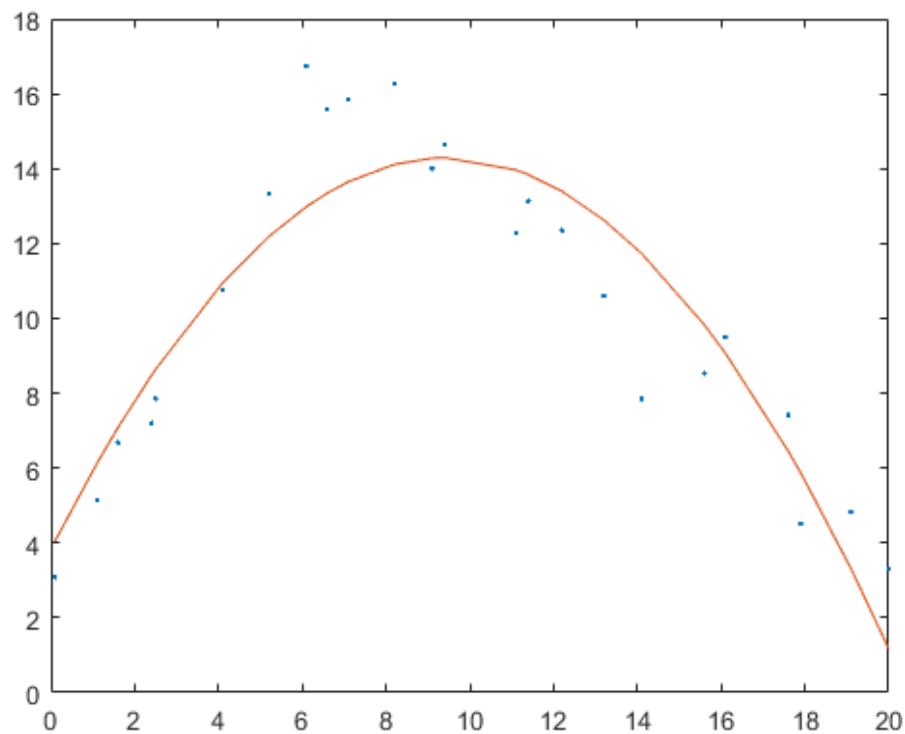
So $y = 11.2446 - 0.1227x$



% quadratic

```
>> A=[length(x), sum(x), sum(x.^2); sum(x), sum(x.^2), sum(x.^3); sum(x.^2), sum(x.^3), sum(x.^4)]
>> b=[sum(y);
    sum(x.*y);
    sum(y.*x.^2)];
>> c=A\b
c =
    3.830554635074556
    2.217397324655864
   -0.117509707495235
>> sum((y-(x.^2*c(3)+x*c(2)+c(1))).^2)
S =
    69.606048786689456
```

So $y = 3.83055 + 2.2173x - 0.1175x^2$



%cubic

```
>> A=[length(x), sum(x), sum(x.^2), sum(x.^3); sum(x), sum(x.^2), sum(x.^3), sum(x.^4); sum(x.^2), sum(x.^3), sum(x.^4),
sum(x.^5); sum(x.^3), sum(x.^4), sum(x.^5), sum(x.^6)];
>> b=[sum(y);
    sum(x.*y);
    sum(y.*x.^2);
    sum(y.*x.^3)];
>> c=A\b
c =
    1.135817111223791
    4.007623065260419
   -0.346696821555106
    0.007648732850660
```

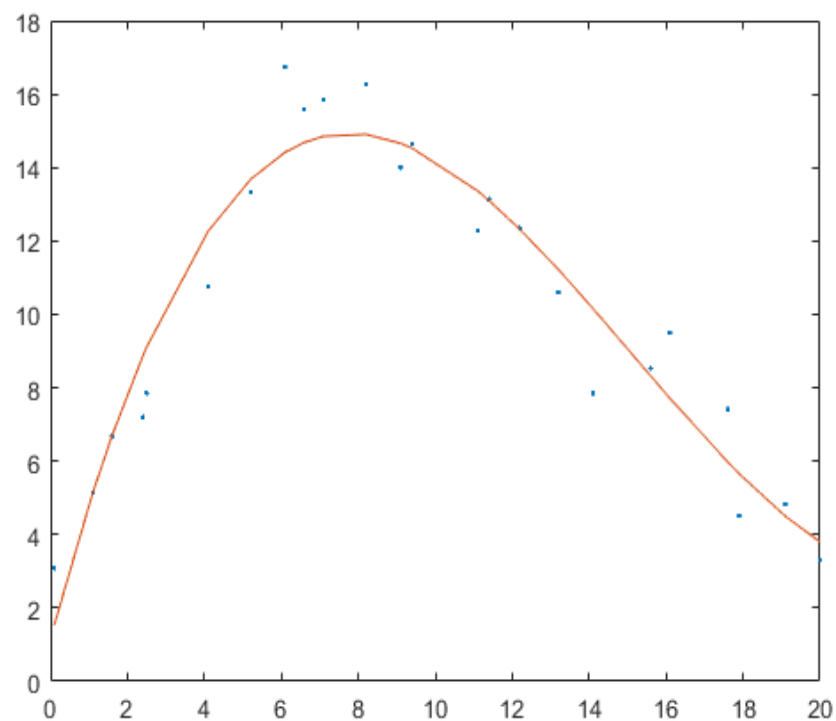


```
>> sum((y-(x.^3*c(4)+x.^2*c(3)+x*c(2)+c(1))).^2)
```

S =

32.475350445810221

So $y = 1.135 + 4.007x - 0.3466x^2 + 0.0076x^3$



These regressions are not that great. The corner at the top is pretty sharp. The curve does not capture this very well.

4.

See attached code that I wrote to solve this.

Here is a plot of the best fit. See last page for proof/derivation of piecewise linear fit.

```
>> piecewisefit
```

The Best Fit is row 10:

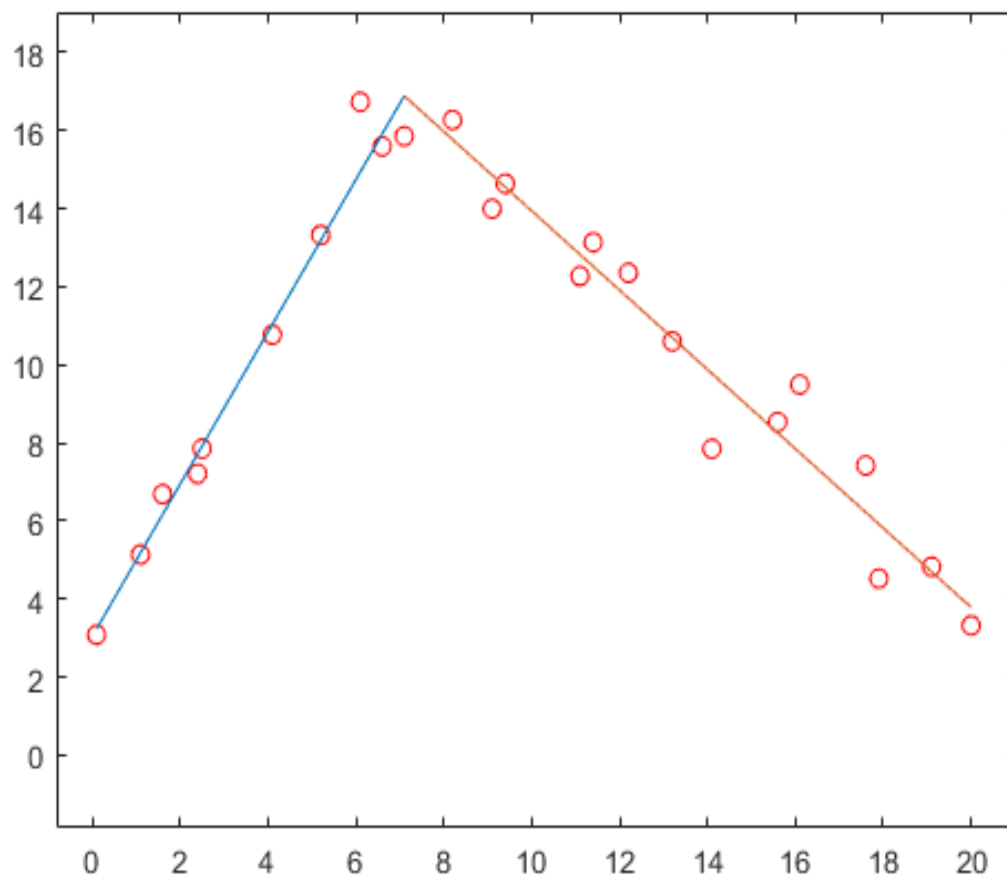
Sq.Error = 17.654541

Slope 1 = 1.951240

Intercept 1 = 3.034009

Slope 2 = -1.016165

Intercept 2 = 24.102580



See next page for derivation #4

$$S = \sum_{i=1}^j (y_i - c_1 - c_2 x_i)^2 + \sum_{i=j+1}^n (y_i - c_1 - \epsilon c_2 + \epsilon c_4 - c_4 x_i)^2$$

Sam Sharf

$$\frac{\partial S}{\partial c_1} = 2 \sum_{i=1}^j (y_i - c_1 - c_2 x_i) \cdot (-1) + 2 \sum_{i=j+1}^n (y_i - c_1 - \epsilon c_2 + \epsilon c_4 - c_4 x_i) \cdot (-1)$$

4-25-19

$$-2 \sum_{i=1}^j (y_i - c_1) + 2 \sum_{i=1}^j (c_2 x_i) + 2 \sum_{i=j+1}^n (\epsilon c_2 - \epsilon c_4 + c_4 x_i) = 0$$

pr # 4

$$\sum_{i=1}^j (c_1 - y_i) + \sum_{i=1}^j (c_2 x_i) + \sum_{i=j+1}^n (\epsilon c_2 - \epsilon c_4 + c_4 x_i) = 0$$

$$n c_1 - \sum_{i=1}^j (y_i) + \sum_{i=1}^j (c_2 x_i) + (n-j) \epsilon c_2 - \sum_{i=j+1}^n c_4 \epsilon + \sum_{i=j+1}^n (c_4 x_i) = 0$$

$$n c_1 - \sum_{i=1}^j (y_i) + \sum_{i=1}^j (c_2 x_i) + (n-j) \epsilon c_2 - (n-j) c_4 \epsilon + c_4 \sum_{i=j+1}^n x_i = 0$$

$$\textcircled{1} \quad n c_1 + c_2 \left[\sum_{i=1}^j (x_i) + (n-j) \epsilon \right] + c_4 \left[\sum_{i=j+1}^n (x_i) - (n-j) \epsilon \right] = \sum_{i=1}^j (y_i)$$

$$\frac{\partial S}{\partial c_2} = 2 \sum_{i=1}^j (y_i - c_1 - c_2 x_i) (-x_i) + 2 \sum_{i=j+1}^n (y_i - c_1 - \epsilon c_2 + \epsilon c_4 - c_4 x_i) (-\epsilon) = 0$$

$$\sum_{i=1}^j (-x_i y_i + c_1 x_i + c_2 x_i^2) - \epsilon \sum_{i=j+1}^n (y_i) + \epsilon c_1 (n-j) + \epsilon^2 c_2 (n-j) - \epsilon^2 c_4 (n-j) + \epsilon c_4 \sum_{i=j+1}^n (x_i) = 0$$

$$\textcircled{2} \quad c_1 \left[\sum_{i=1}^j (x_i) + (n-j) \epsilon \right] + c_2 \left[\sum_{i=1}^j (x_i^2) + (n-j) \epsilon^2 \right] + c_4 \left[\epsilon \sum_{i=j+1}^n (x_i) - \epsilon^2 (n-j) \right] = \sum_{i=1}^j x_i y_i + \epsilon \sum_{i=j+1}^n (y_i)$$

$$\frac{\partial S}{\partial c_4} = 2 \sum_{i=1}^j (y_i - c_1 - c_2 x_i) (0) + 2 \sum_{i=j+1}^n (y_i - c_1 - \epsilon c_2 + \epsilon c_4 - c_4 x_i) (\epsilon - x_i) = 0$$

$$\sum_{i=j+1}^n (\epsilon y_i - \epsilon c_1 - \epsilon^2 c_2 + \epsilon^2 c_4 - \epsilon c_4 x_i) - \sum_{i=j+1}^n (x_i y_i - c_1 x_i - \epsilon c_2 x_i + \epsilon c_4 x_i - c_4 x_i^2) = 0$$

$$\epsilon \sum_{i=j+1}^n y_i - (n-j) \epsilon c_1 - (n-j) \epsilon^2 c_2 + (n-j) \epsilon^2 c_4 - \epsilon c_4 \sum_{i=j+1}^n x_i - \sum_{i=j+1}^n (x_i y_i) + \sum_{i=j+1}^n c_1 x_i + \epsilon c_2 \sum_{i=j+1}^n x_i$$

$$- \epsilon c_4 \sum_{i=j+1}^n (x_i) + c_4 \sum_{i=j+1}^n x_i^2 = 0$$

$$\textcircled{3} \quad c_1 \left[\sum_{i=j+1}^n (x_i) - (n-j) \epsilon \right] + c_2 \left[\epsilon \sum_{i=j+1}^n (x_i) - (n-j) \epsilon^2 \right] + c_4 \left[\sum_{i=j+1}^n (x_i^2 - 2 \epsilon x_i + \epsilon^2) \right] = \sum_{i=j+1}^n (x_i y_i - \epsilon)$$

Now this system of Eqns can be solved using MATLAB!

(They are equivalent to the ones in the test)