We are forecasting the future of global warming if current trends hold. To do this we used the site <http://berkeleyearth.lbl.gov/auto/Global/Land_and_Ocean_summary.txt> and we used their most accurate data which was Annual Anomaly. This data spans between 1850 and 2018. Using this data, we hope to find the annual anomaly of the global temperature in 2100.

Using the least square method I found a best fit for a linear equation, a quadratic equation, a cubic equation and an exponential equation. My final equations were:

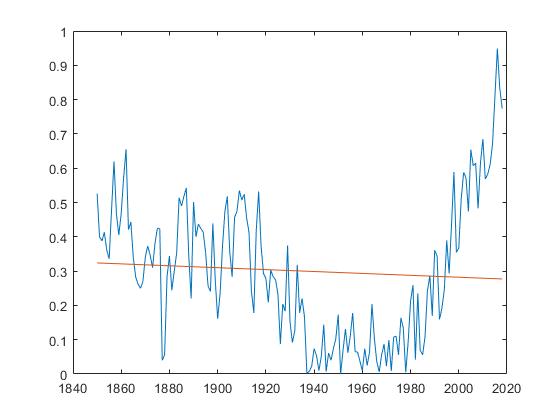
0.32384 – 0.00028x, (1)

0.63839 – 0.01138x + 0.00007x2, (2)

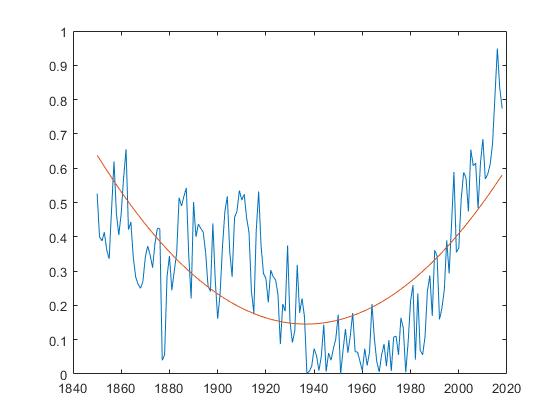
0.36221 + 0.00794x – 0.00022x2 + 0.000001x3, (3)

0.26175 + 0.23061e-0.03771x, (4)

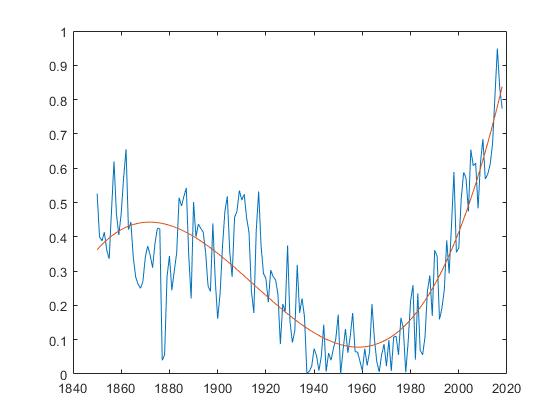
Of which my cubic fitted equation, equation (3), was the closest to the data as seen below.



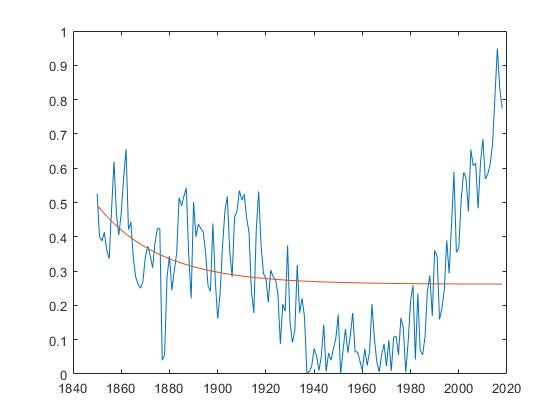
**Graph 1:** Linear



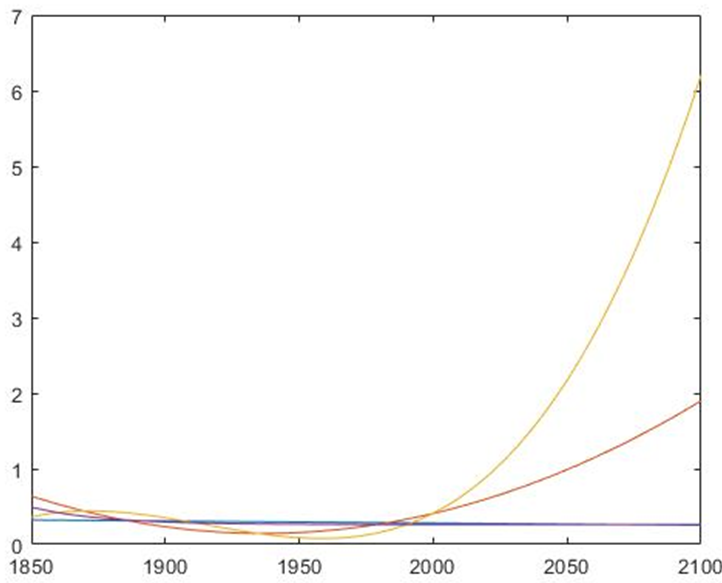
**Graph 2:** Quadratic



**Graph 3:** Cubic



**Graph 4:**  Exponential

Plugging in x = 250 (2100 – 1850 = 250) for my equations gave results of:

Linear: 0.25429

Quadratic: 1.89910

Cubic: 6.20818

Exponential: 0.26177

**Graph 5:** Previous graphs extrapolated to 2100

Given that these are temperature abnormalities in degrees Celsius it becomes abundantly clear with the given data set that Earth will experience a temperature hike of 6.2 degrees Celsius by the year 2100. Please note that this result is based off a cubic function of best fit to data already provided and it is only one way of interpreting said data. With a fitted trigonometric function or a fitted multivariable exponential equation, we would see our predictions as either a much lower number or a much higher number with similar relative accuracy.

My MATLAB code:

% globalwarming\_project.m

%

% by Nathan Pelletier

% AUCSC 340

% April 5 2019

%

% Program uses least square method on a large data set

% from http://berkeleyearth.lbl.gov/auto/Global/Land\_and\_Ocean\_summary.txt

% (of which we are only using annual anomaly) to calculate a fitted linear,

% quadratic and cubic function. We are also using lsqcurvefit() to find

% a fitted exponential function for our data.

% Using said equations the program then graphs the results and

% is used to find data for 2100

% please note: I used a separate java program to separate annual anomaly

% from the rest of the data for typing purposes

annual\_anomaly =[0.526 0.400 0.388 0.413 0.361 0.336 0.480 0.620...

0.468 0.405 0.469 0.573 0.655 0.421 0.443 0.340 0.283 0.262...

0.250 0.270 0.342 0.373 0.346 0.310 0.381 0.425 0.424 0.040...

0.056 0.284 0.344 0.244 0.300 0.353 0.514 0.490 0.518 0.542...

0.346 0.220 0.501 0.400 0.437 0.424 0.414 0.354 0.257 0.241...

0.439 0.273 0.161 0.233 0.372 0.474 0.518 0.357 0.283 0.456...

0.474 0.535 0.507 0.524 0.455 0.413 0.241 0.178 0.419 0.532...

0.374 0.293 0.279 0.209 0.303 0.283 0.275 0.232 0.087 0.204...

0.183 0.375 0.152 0.092 0.127 0.318 0.178 0.220 0.167 0.002...

0.008 0.023 0.074 0.053 0.010 0.052 0.144 0.007 0.061 0.040...

0.074 0.103 0.173 0.000 0.070 0.131 0.062 0.111 0.178 0.065...

0.064 0.038 0.009 0.074 0.025 0.064 0.204 0.103 0.035 0.006...

0.056 0.087 0.023 0.099 0.009 0.108 0.110 0.056 0.164 0.133...

0.005 0.093 0.212 0.259 0.042 0.235 0.069 0.056 0.108 0.242...

0.287 0.169 0.361 0.344 0.159 0.191 0.246 0.390 0.293 0.437...

0.589 0.355 0.367 0.509 0.588 0.572 0.474 0.654 0.607 0.615...

0.483 0.616 0.685 0.569 0.582 0.610 0.671 0.810 0.949 0.833...

0.774];

x1 = linear\_least\_square(annual\_anomaly)

x2 = quadratic\_least\_square(annual\_anomaly)

x3 = cubic\_least\_square(annual\_anomaly)

x4 = exponential\_least\_square(annual\_anomaly)

x = 250;

% x = 0:168;

% x = 0:250;

linear = x1(1) + x1(2) \* x

quadratic = x2(1) + x2(2) \* x + x2(3) \* x.^2

cubic = x3(1) + x3(2) \* x + x3(3) \* x.^2 + x3(4) \* x.^3

exponential = x4(1) + x4(2) \* exp(x4(3) \* x)

% THIS WAS TO SEE WHICH FUNCTION SET BEST FIT THE DATA

% CUBIC FIT IT BEST

%==============

% year = 1850:2018;

% plot(year, annual\_annomaly, year, exponential)

% year = 1850:2100;

% length(x)

% length(year)

% plot(year, linear, year, quadratic, year, cubic, year, exponential)

% linear\_least\_square(array[n]) --> array[2]

%

% takes an array of any size and returns the coefficents of a fitted

% linear equation. The answer is returned in an array of size 2 where

% a + bx ==> [a]

% [b]

function answer = linear\_least\_square(point)

x1 = 0;

x2 = 0;

y = 0;

yx1 = 0;

n = length(point) - 1; % returns a count of all elements starting at 1

%calculates out all summation elements

for i = 0:n

x1 = i + x1;

x2 = i^2 + x2;

%point(i + 1) because arrays start at 1 not 0 and end at n + 1 not n

y = point(i + 1) + y;

yx1 = point(i + 1) \* i + yx1;

end %for

A = [n x1; x1 x2];

B = [y; yx1];

answer = A\B;

end %linear\_least\_square

% quadratic\_least\_square(array[n]) --> array[3]

%

% takes an array of any size and returns the coefficents of a fitted

% quadratic equation. The answer is returned in an array of size 3 where

% a + bx + cx^2 ==> [a]

% |b|

% [c]

function answer = quadratic\_least\_square(point)

x1 = 0;

x2 = 0;

x3 = 0;

x4 = 0;

y = 0;

yx1 = 0;

yx2 = 0;

n = length(point) - 1;

%for loop to calculate individual elements of summation notation

for i = 0:n

x1 = i + x1;

x2 = i^2 + x2;

x3 = i^3 + x3;

x4 = i^4 + x4;

% i + 1 becuase matlab indexes into arrays at 1 not 0

y = point(i + 1) + y;

yx1 = point(i + 1) \* i + yx1;

yx2 = point(i + 1) \* i^2 + yx2;

end %for

A = [n x1 x2; x1 x2 x3; x2 x3 x4];

B = [y; yx1; yx2];

answer = A\B;

end %quadratic\_least\_square

% cubic\_least\_square(array[n]) --> array[4]

%

% takes an array of any size and returns the coefficents of a fitted

% cubic equation. The answer is returned in an array of size 4 where

% a + bx + cx^2 + d^3 ==> [a]

% |b|

% |c|

% [d]

function answer = cubic\_least\_square(point)

x1 = 0;

x2 = 0;

x3 = 0;

x4 = 0;

x5 = 0;

x6 = 0;

y = 0;

yx1 = 0;

yx2 = 0;

yx3 = 0;

n = length(point) - 1;

%for loop to calculate individual summation elements of final matrix

for i = 0:n

x1 = i + x1;

x2 = i^2 + x2;

x3 = i^3 + x3;

x4 = i^4 + x4;

x5 = i^5 + x5;

x6 = i^6 + x6;

% i + 1 because matlab starts indexes into arrays at 1 not 0

y = point(i + 1) + y;

yx1 = point(i + 1) \* i + yx1;

yx2 = point(i + 1) \* i^2 + yx2;

yx3 = point(i + 1) \* i^3 + yx3;

end %for

A = [n x1 x2 x3; x1 x2 x3 x4; x2 x3 x4 x5; x3 x4 x5 x6];

B = [y; yx1; yx2; yx3];

answer = A\B;

end %cubic\_least\_square

% exponential\_least\_square(array[n]) ==> array[3]

%

% takes an array of any size and returns the coefficents of the fitted

% function a + be^(xc)

% and returns it in the form

% [a]

% [b]

% [c]

function answer = exponential\_least\_square(point)

n = 2018 - 1850;

t = [0:n];

F = @(x,xdata) x(1) + x(2)\*exp(x(3)\*xdata);

x0 = [1 1 0];

[answer,resnorm,~,exitflag,output] = lsqcurvefit(F,x0,t,point)

end %exponential\_least\_square