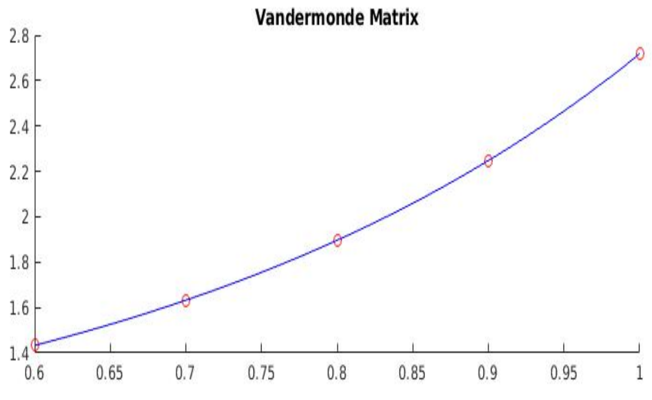
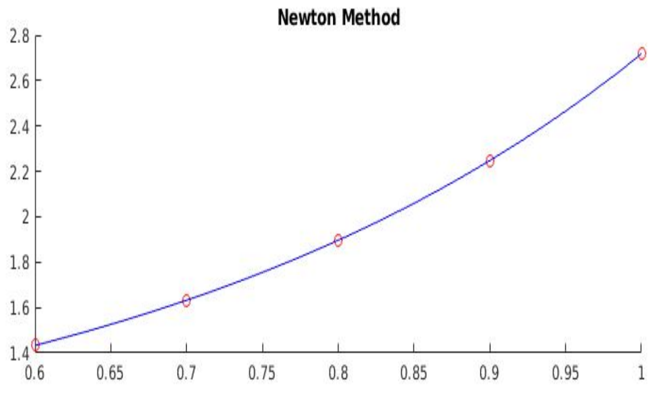
EECS 639 Project

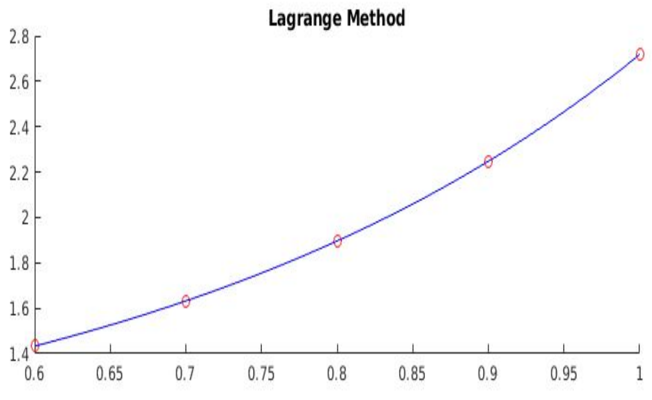
Justin Roderman & Harrison Luo

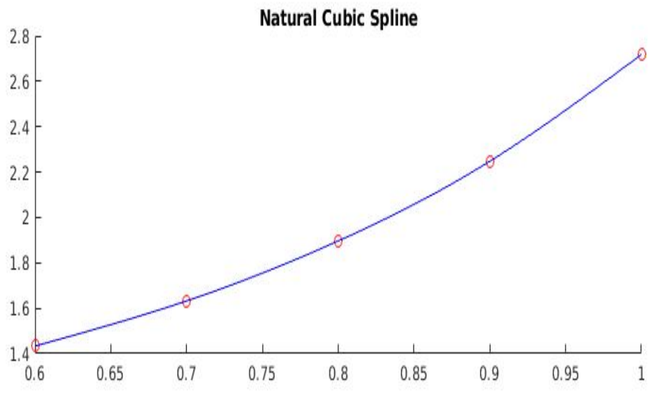
Part B

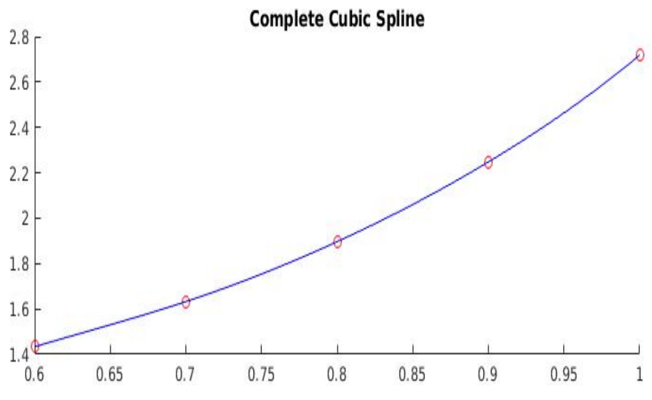
Dataset 1

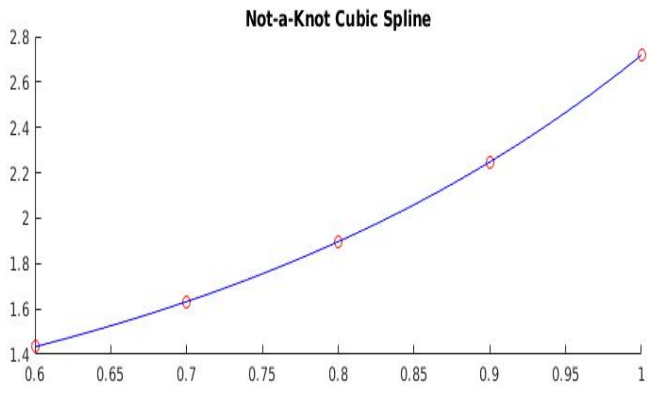






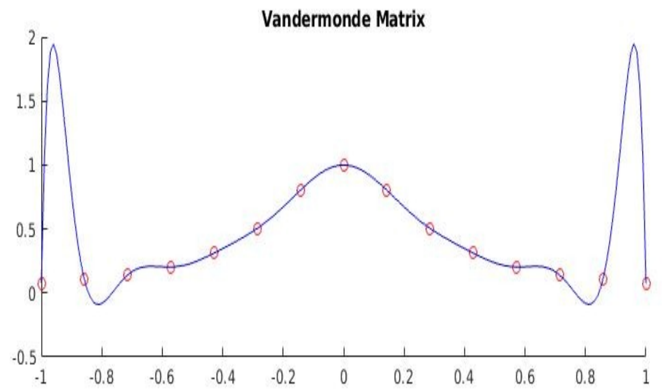


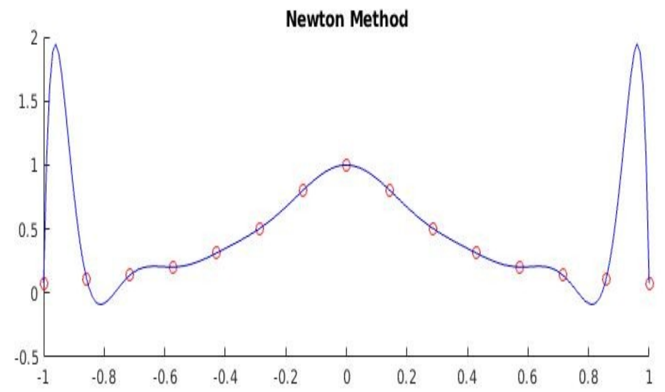


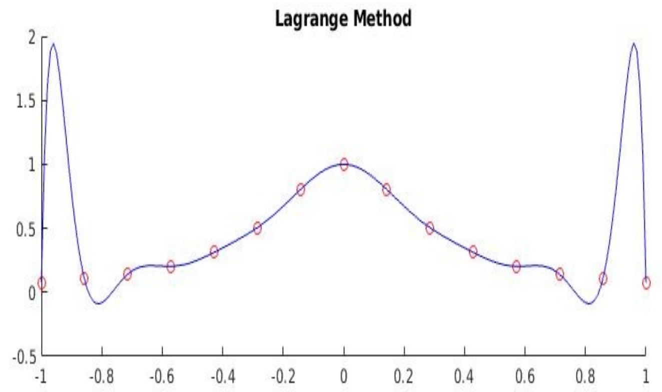


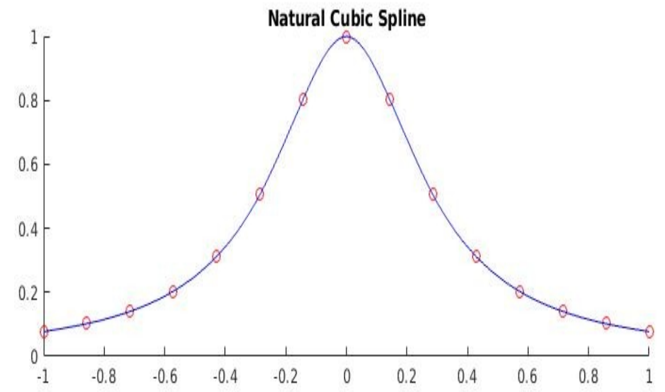
Part B

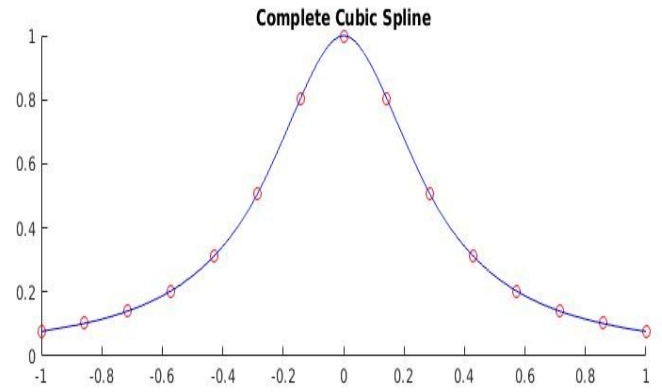
Dataset 2

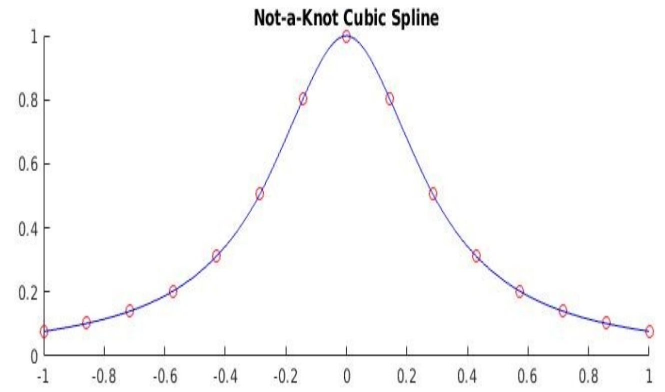






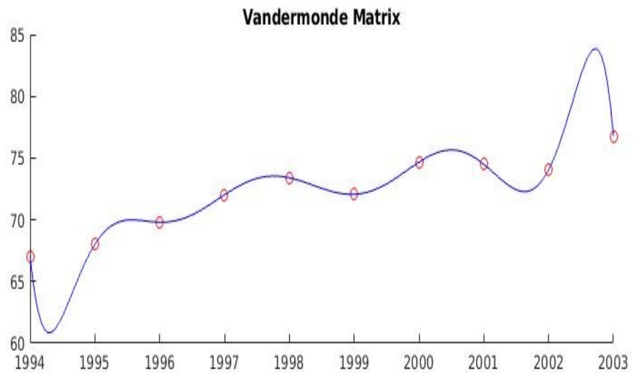


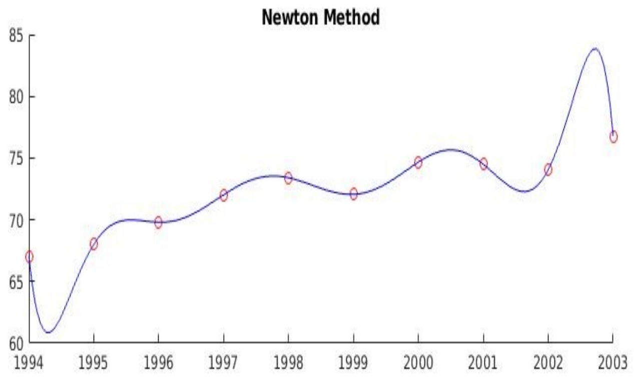


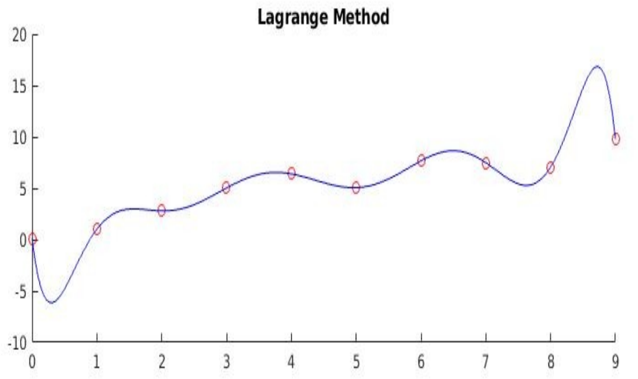


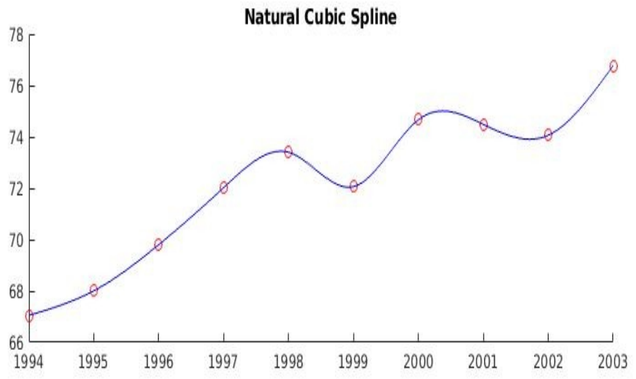
Part B

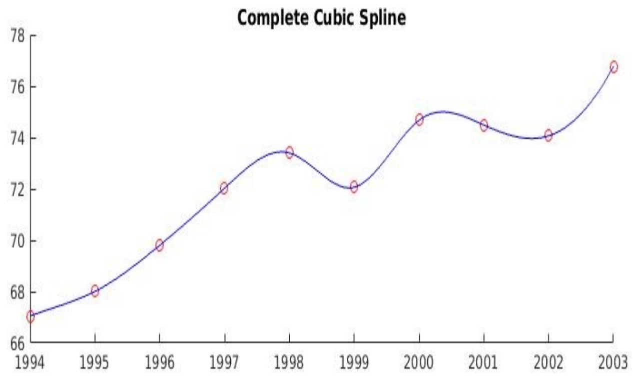
Dataset 3

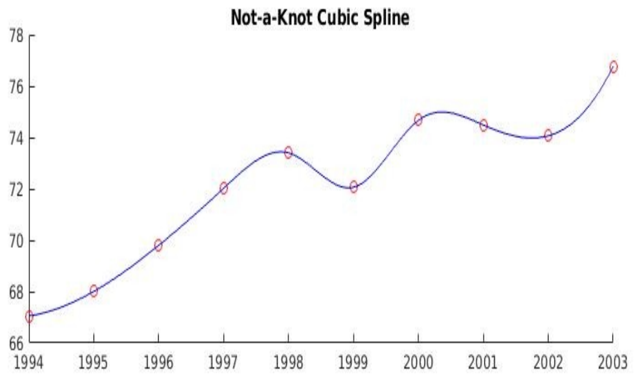




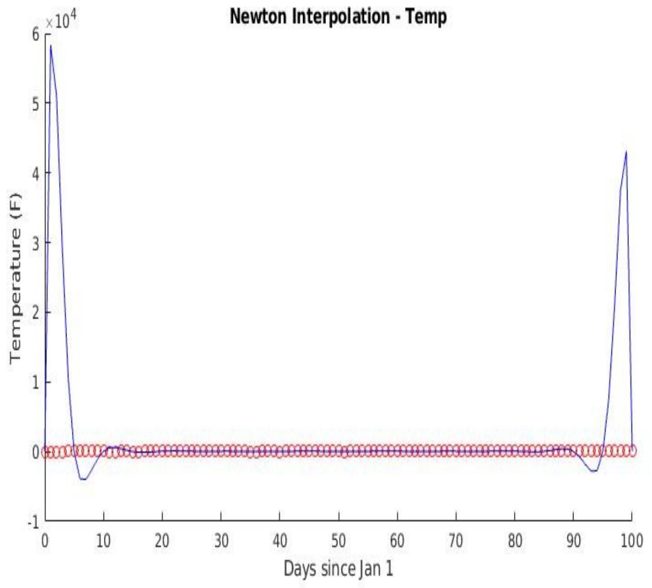


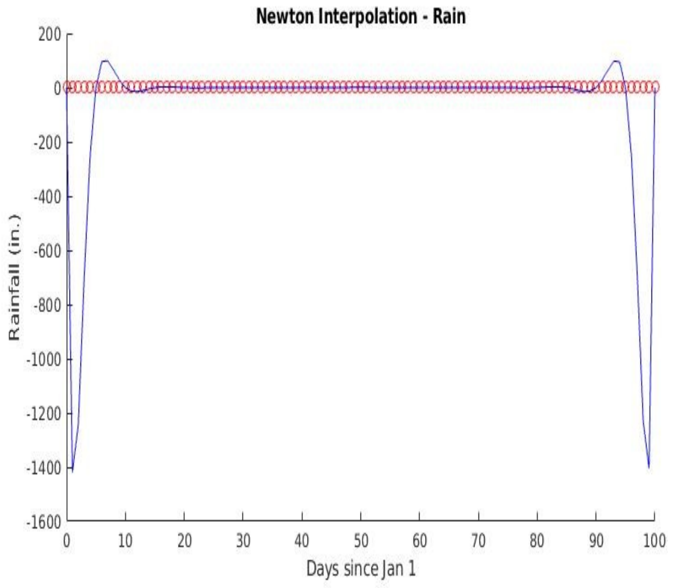


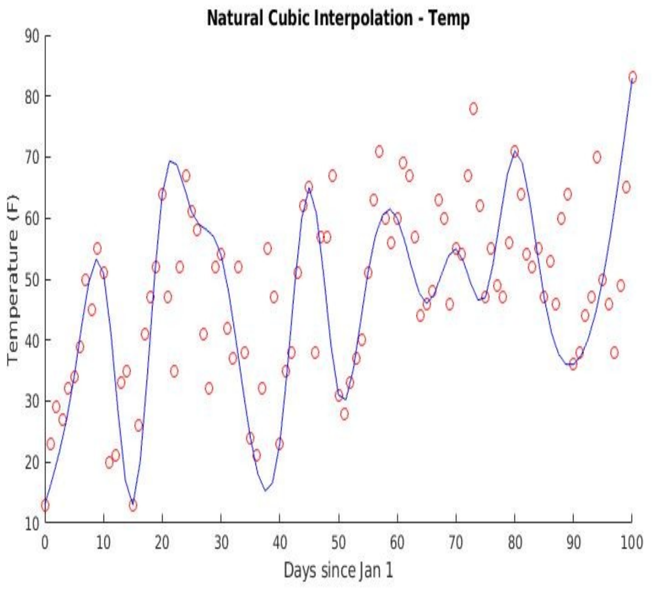


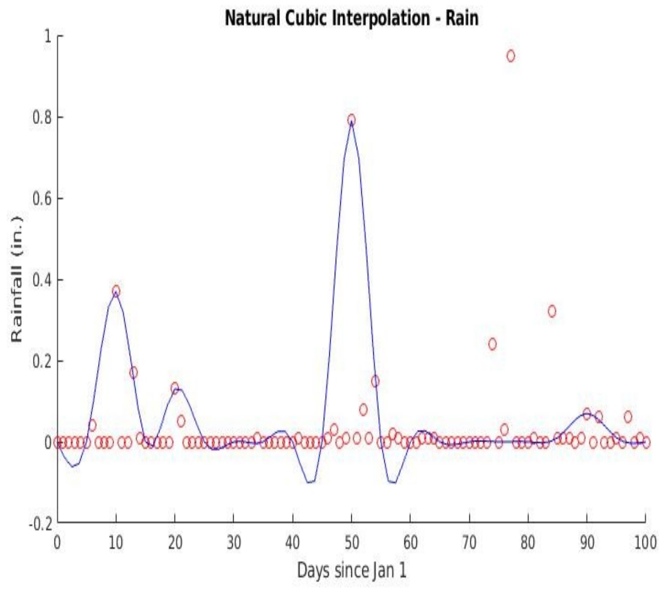


Part C


Newton Interpolation - Temp
Temperature (F)
Days since Jan 1
×10^4


Newton Interpolation - Rain
Rainfall (in.)
Days since Jan 1


Natural Cubic Interpolation - Temp
Temperature (F)
Days since Jan 1


Natural Cubic Interpolation - Rain
Rainfall (in.)
Days since Jan 1

Data Set 1
f(x) = e^x^2 on [.6,1] with 5 Equally Spaced Points

Initial Analysis - Very few points, and a very simple function. I should expect little problems with interpreting this data.

Vandermonde:

Accuracy - With only 5 points of data Vandermonde will be pretty accurate, as the degree is still low so there is little effect yet… on the conditionality of the matrix.
Efficiency - Costly Operations uses a complexity of O(N^2), Vandermonde will always be a high cost.

Newton:

Accuracy - Newton on the other hand benefits from having a larger pool of data for higher accuracy.
Efficiency - Newton tends to be more efficient than Vandermonde. Still has a complexity of O(n^2)

Lagrange:

Accuracy - Pretty accurate, but would be more accurate needs with more than 5 data points, however with the identity matrix it results in very little error. Could have a moment where it is divided by zero.
Efficiency - Extremely Expensive to solve the Lagrange Matrix no matter the size

Natural Cubic Spline:

Accuracy - zero values are being set to the second derivative values which doesn't match the values that the function may be measuring. S
Efficiency - Generally pretty efficient, comparatively more efficient when compared to other algorithms when in terms of larger data sets

Complete Cubic Spline:

Accuracy - Pretty accurate approximations, with a small data set it could be more reliable with more data points
Efficiency - Generally pretty efficient, comparatively more efficient when compared to other algorithms when in terms of larger data sets

Not-A-Knot Spline:

Accuracy - Better than the natural cubic spline in terms of accuracy.
Efficiency - Generally pretty efficient, comparatively more efficient when compared to other algorithms when in terms of larger data sets

Data Set 2
f(x) = 1/(1+12X^2) on [-1,1] with 15 Equally Spaced Points

Initial Analysis - More points, and a more complicated function. I should expect some more problems with interpreting this data.

Vandermonde:

Accuracy - Becomes increasingly ill-conditioned with more points of data. This is due to the residual of the linear system
Efficiency - The computational cost also increases significantly. As the Vandermonde Matrix becomes more ill-conditioned as degree is increasing

Newton:

Accuracy - When compared to Vandermonde and Lagrange, Newton is still the most stable choice. Newton is pretty accurate with more points.
Efficiency - Cost is increased because more operations are needed but is a 1 for 1 relationship and is balanced.

Lagrange:

Accuracy - Identity Matrix still makes it relatively accurate except at the end of the matrix. Falls into the risk of a divide by zero.
Efficiency - Absolutely off the charts, as more points are added Lagrange becomes intense to calculate.

Natural Cubic Spline:

Accuracy - More accurate with more data points given.
Efficiency - Cubic Splines are still efficient of an option to go with to solve. However, it does cost quite a significant amount more to solve

Complete Cubic Spline:

Accuracy - Accurate at the endpoints at an arbitrary point
Efficiency - Cubic Splines are still efficient of an option to go with to solve. However, it does cost quite a significant amount more to solve

Not-A-Knot Spline:

Accuracy - More accurate than Natural, but captures the third derivative of the spline to be continuous at the X.
Efficiency - Cubic Splines are still efficient of an option to go with to solve. However, it does cost quite a significant amount more to solve

Data Set 3
Years - 1994 - 2003

Initial Analysis - Consistent number of points however, the points can vary a little more should see some deviations in the accuracy.

Vandermonde:

Accuracy - Becomes increasingly ill-conditioned with more points of data. End points are less accurate

Efficiency - The computational cost also increases significantly. As the Vandermonde Matrix becomes more ill-conditioned as degree is increasing. Getting the squares of so many of these large numbers and square roots can result in some serious approximation issues.

Newton:

Accuracy - However when compared to Vandermonde and Lagrange, Newton is still the most stable choice. It still lacks in accuracy at the two endpoints as well.

Efficiency - Cost is incremental and is moderately balanced. Less chance of overflow however.

Lagrange:

Accuracy – Lagrange accuracy is still extremely high, however never the forget that it comes at a great cost in particular.

Efficiency - Absolutely off the charts, as more points are added Lagrange becomes intense to calculate. Especially considering the larger numbers there will be more added problems when multiplying these numbers together.

Natural Cubic Spline:

Accuracy - More accurate with closer data points given. With a smaller deviation. With the data points with yearly differences there could be more fluctuation in the cubic splines.

Efficiency - Cubic Splines are still efficient of an option to go with to solve. However, it does cost quite a significant amount more to solve

Complete Cubic Spline:

Accuracy - More accurate with closer data points given. With a smaller deviation. With the data points with yearly differences there could be more fluctuation in the cubic splines.

Efficiency - Cubic Splines are still efficient of an option to go with to solve. Clamped will approach the end points at a better solution

Not-A-Knot Spline:

Accuracy - More accurate with closer data points given. With a smaller deviation. With the data points with yearly differences there could be more fluctuation in the cubic splines.

Efficiency - Cubic Splines are still efficient of an option to go with to solve. However, it does cost quite a significant amount more to solve

Common Patterns I observed:
Vandermonde would be a better fit for a small data set and a quick solution but inaccurate solution. Newton is the best all around decision with a incremental cost for efficiency but accuracy remains relatively stable due to its reliance on previous inputs and lower triangular solution.

Lagrange has the most accurate solution, but its just so costly and there are risks associated like a divide by zero it should remain as just a theoretical concept.
Cubic Splines are all around good choices certain ice cream flavors makes slight differences between the three, but hey… its Ice Cream so who’s going to complain? Unless its banana, banana sucks.

Part C
Daily Temperatures and Rainfalls

1. Daily Temperatures:
   a) Maximum Interpolation Error: **5.8319e+04** F
   The Runge phenomenon is definitely evident in this plot - the data is fairly consistent until it gets to the ends, where it proceeds to dip into wildly massive numbers.
   b) Maximum Interpolation Error: **38.4449** in.
   The Runge phenomenon does not appear in this plot - cubic spline mitigates this issue by having multiple equations, leading to the error not being bunched up at the ends.
   c) Overall, it seems as though cubic spline is the better approach here. In cubic spline, the data is a lot more well-fitted to the data than Newton. Even if you ignore the two ends of the Newton Interpolant, the amount of error is still much more than Cubic Spline.
2. Daily Rainfall:
   a) Maximum Interpolation Error: **1.4190e+03** F
   The Runge phenomenon is definitely evident again in this plot - the data is fairly consistent until it gets to the ends, where it proceeds to dip into wildly massive numbers.
   b) Maximum Interpolation Error: **0.9495** in.
   The Runge phenomenon does not appear again in this plot - cubic spline mitigates this issue by having multiple equations, leading to the error not being bunched up at the ends.
   c) The cubic spline is still superior in this data analysis. The Runge phenomenon still creates a massive error in the Newton Interpolant, but not in the Cubic Spline.
3. If you chose a city that is possibly closer to the coast like San Francisco, it would probably create a nicer data interpolant. This is because the weather would be a lot more consistent than Kansas, where the weather changes every 2 seconds. This creates a data set that looks random, leading to an inaccurate interpolant.
4. From this application, I have learned that although Cubic Spline can take a longer time than Newton Interpolant, but in the end, the interpolant works much better at representing the data set.

vandermonde.m

```
function [Xout, Yout] = vandermonde(Xin, Yin, num)
% Creates the Vandermonde Matrix and evaluates the data.

%   Inputs : Xin - The inputs for the X points
%            Yin - The Inputs for the Y points
%            num - The amount of points for the output
%   Output : Xout - The x-values of the computed data
%            Yout - The y-values of the computed data
%

n = length(Xin);
% Creates the initial Vandermonde matrix
van = zeros(n);

%Sets the first column
van(:,1) = 1;

%For loop to fill the vandermonde
for k = 2:n
    van(:,k) = Xin(:).*van(:,(k-1));
end

%Solves for coeffecients by using GEPP with back-sub
coef = linsolve(van, Yin');

% Use Horner's Nested Theorem to calculate data points
Xout = linspace(Xin(1), Xin(n), num);
Yout = ones(1, num) * coef(n);

% Yout = coef1 + x1(coef2 + x2(...coef(n-1) + x(n-1)(coefn))...))
for i=(n-1):-1:1
    Yout = coef(i) + (Xout .* Yout);
end

end
```

newton.m

```
function [Xout, Yout] = newton(Xin,Yin, num)
% Creates the Newton Matrix and evaluates the data.

%   Inputs : Xin - The inputs for the X points
%            Yin - The Inputs for the Y points
%            num - The amount of points for the output
%   Output : Xout - The x-values of the computed data
%            Yout - The y-values of the computed data
%

%Calculates the size of the input
n = length(Xin);

%Creates a matrix with just ones
A = ones(n);

% So we don't have to take the transpose every single time
Xin = Xin';

% Multiplies the last column by (t - tk)
for k = 2:n
    A(k:n,k) = A(k:n,(k-1)) .* (Xin(k:n) - Xin(k-1));
end
% A has ones in the top right corner, but if we tell our linsolve
%   that the matrix is Lower-Triangular, it will ignore those 1s
%   and solve solely using forward-sub
opts.LT = true;

% Calculate the coeffecients by using forward-sub
coef = linsolve(A, Yin', opts);

% Use Horner's Nested Theorem to calculate data points
Xout = linspace(Xin(1), Xin(n), num);
Yout = ones(1, num) * coef(n);

% Yout = coef1 + x1(coef2 + x2(...coef(n-1) + x(n-1)(coefn))...))
%   where xi = (xi - Xini)
for i=(n-1):-1:1
    Yout = coef(i) + ((Xout - Xin(i)) .* Yout);
end


end
```

lagrange.m

```
function [Xout, Yout] = lagrange(Xin,Yin, num)
% Creates the Lagrange Equation and evaluates the data.

%   Inputs : Xin - The inputs for the X points
%            Yin - The Inputs for the Y points
%            num - The amount of points for the output
%   Output : Xout - The x-values of the computed data
%            Yout - The y-values of the computed data
%

%Calculates the size of the input
n = length(Xin);

% Set up output X and Y
Xout = linspace(Xin(1), Xin(n), num);
Yout = zeros(n, 1);

% Create all of the equations
for j=1:n
    % Calculate the numerator
    %   (t - t1) * (t - t2) * ... * (t - tn)
    %           (Skipping tj)
    % And calculate the denominator
    %   (tj - t1) * (tj - t2) * ... * (tj - tn)
    %           (Skipping tj)
    numer = 1;
    denom = 1;
    for k=1:n
        if k ~= j % Skip tj
            numer = numer .* (Xout - Xin(k));
            denom = denom * (Xin(j) - Xin(k));
        end
    end

    % Calculate Y step by step
    %   y1 * l1(t) + y2 * l2(t) + ... + yn * ln(t)
    Yout = Yout + Yin(j) * (numer / denom);
end

end
```

cubicSpline.m

```
function [Xout,Yout] = cubicSpline(Xin,Yin,splineType, num)
% Creates the cubic spline from the given data
%
%   Input: Xin - x-values of the data
%          Yin - corresponding y-values of the data
%          splineType - type of cubic spline to use
%               + 'natural' for natural cubic spline
%               + 'complete' for complete cubic spline
%               + 'not-a-knot' for not-a-knot cubic spline
%          num - The amount of points for the output
%
%   Output : Xout - The x-values of the computed data
%            Yout - The y-values of the computed data
%
% Calculate the coeffecients for the spline equations
%

n = length(Xin); % Length of the data
A = zeros((n-1)*4); % Holds the system of equations
b = zeros((n-1)*4,1); % Holds the y values

% Populate A with P1(t), P2(t), ..., Pn-1(t)
for i=1:(n-1)
    P = [1 Xin(i) Xin(i).^2 Xin(i).^3
         1 Xin(i+1) Xin(i+1).^2 Xin(i+1).^3]; % Pi(t)
    A(i*2-1:i*2,i*4-3:i*4) = P;
end

% Populate A with first & second derivative of P(t)
off1 = (n-1)*2; % Offset for P'(t)
off2 = off1 + (n-2); % Offset for P''(t)
for i=1:(n-2)
    P1 = [0 1 2*Xin(i+1) 3*Xin(i+1).^2]; % First derivative of Pi(t)
    P2 = [0 0 2 6*Xin(i+1)]; % Second derivative of Pi(t)
    A(i+off1,i*4-3:i*4) = P1;
    A(i+off1,(i+1)*4-3:(i+1)*4) = -P1;
    A(i+off2,i*4-3:i*4) = P2;
    A(i+off2,(i+1)*4-3:(i+1)*4) = -P2;
end

% Populate b with its y values
for i=1:(n-1)
    b(i*2-1) = Yin(i);
    b(i*2) = Yin(i+1);
end



% Continued on next page...
% SPLINE CALCULATIONS:

splineCalcRow = (n-1)*4;
if isequal(splineType, 'natural') % Natural Cubic Spline
    A(splineCalcRow-1,3:4) = [2 6*Xin(1)];
    A(splineCalcRow,(n-1)*4-1:(n-1)*4) = [2 6*Xin(n)];
    % Keep b values equal to 0
elseif isequal(splineType, 'complete') % Complete Cubic Spline
    A(splineCalcRow-1,2:4) = [1 2*Xin(1) 3*Xin(1).^2];
    A(splineCalcRow,(n-1)*4-2:(n-1)*4) = [1 2*Xin(1) 3*Xin(1).^2];
    % Sets first derivate equal to the slope at the endpoints
    b(splineCalcRow-1) = ((Yin(2)-Yin(1))/(Xin(2)-Xin(1)));
    b(splineCalcRow) = ((Yin(n)-Yin(n-1))/(Xin(n)-Xin(n-1)));
elseif isequal(splineType, 'not-a-knot') % Not-a-Knot Cubic Spline
    A(splineCalcRow-1,4) = 1;
    A(splineCalcRow-1,8) = -1;
    A(splineCalcRow,(n-1)*4) = 1;
    A(splineCalcRow,(n-2)*4) = -1;
else % Error
    error("splineType is not recognized!")
end

% Calculate coeffecients using Gaussian Elmination w/ Partial Pivoting
%   and Backwards Substitution
coef = linsolve(A, b);

%
% Calculate the spline
%

% step tells how many points in each p(x)
step = fix(num / (n-1)); % Need an integer value,
Xout = zeros((n-1) * step, 1);
Yout = zeros((n-1) * step, 1);
% Calculate values of Yout from pi(x)
for i=1:(n-1)
    x = linspace(Xin(i),Xin(i+1),step);
    y = coef(i*4-3) + coef(i*4-2)*x + coef(i*4-1)*(x.^2) + coef(i*4)*(x.^3);
    Xout((i-1) * step + 1:i*step) = x;
    Yout((i-1) * step + 1:i*step) = y;
end

end
```

testing.m

```
%First Evaluation of all the functions e^x^2
figure
X1 =  linspace(.6,1,5);
Y1 = exp(X1.^2);

% Vandermonde Interpolation
subplot(3,2,1);
hold on
[Xout, Yout] = vandermonde(X1, Y1, 200);
title("Vandermonde Matrix");
plot(X1, Y1, 'ro');
plot(Xout, Yout, 'b');

% Newton Interpolation
subplot(3,2,2);
hold on
[Xout, Yout] = newton(X1, Y1, 200);
title("Newton Method");
plot(X1, Y1, 'ro');
plot(Xout, Yout, 'b');

% Lagrange Interpolation
subplot(3,2,3);
hold on
[Xout, Yout] = lagrange(X1, Y1, 200);
title("Lagrange Method");
plot(X1, Y1, 'ro');
plot(Xout, Yout, 'b');

% Natural Cubic Spline
subplot(3,2,4);
hold on
[Xout, Yout] = cubicSpline(X1, Y1, 'natural', 200);
title("Natural Cubic Spline");
plot(X1, Y1, 'ro');
plot(Xout, Yout, 'b');

% Complete Cubic Spline
subplot(3,2,5);
hold on
[Xout, Yout] = cubicSpline(X1, Y1, 'complete', 200);
title("Complete Cubic Spline");
plot(X1, Y1, 'ro');
plot(Xout, Yout, 'b');

% Not-a-Knot Cubic Spline
subplot(3,2,6);
hold on
[Xout, Yout] = cubicSpline(X1, Y1, 'not-a-knot', 200);
title("Not-a-Knot Cubic Spline");
plot(X1, Y1, 'ro');
plot(Xout, Yout, 'b');
```

```
%Second Evaluation of the functions. 1/(1+12*(x1*^2)
figure
X2 = linspace(-1,1,15);
Y2 = (1+12*(X2.^2)).^(-1);

% Vandermonde Interpolation
subplot(3,2,1);
hold on
[Xout, Yout] = vandermonde(X2, Y2, 200);
title("Vandermonde Matrix");
plot(X2, Y2, 'ro');
plot(Xout, Yout, 'b');

% Newton Interpolation
subplot(3,2,2);
hold on
[Xout, Yout] = newton(X2, Y2, 200);
title("Newton Method");
plot(X2, Y2, 'ro');
plot(Xout, Yout, 'b');

% Lagrange Interpolation
subplot(3,2,3);
hold on
[Xout, Yout] = lagrange(X2, Y2, 200);
title("Lagrange Method");
plot(X2, Y2, 'ro');
plot(Xout, Yout, 'b');

% Natural Cubic Spline
subplot(3,2,4);
hold on
[Xout, Yout] = cubicSpline(X2, Y2, 'natural', 200);
title("Natural Cubic Spline");
plot(X2, Y2, 'ro');
plot(Xout, Yout, 'b');

% Complete Cubic Spline
subplot(3,2,5);
hold on
[Xout, Yout] = cubicSpline(X2, Y2, 'complete', 200);
title("Complete Cubic Spline");
plot(X2, Y2, 'ro');
plot(Xout, Yout, 'b');

% Not-a-Knot Cubic Spline
subplot(3,2,6);
hold on
[Xout, Yout] = cubicSpline(X2, Y2, 'not-a-knot', 200);
title("Not-a-Knot Cubic Spline");
plot(X2, Y2, 'ro');
plot(Xout, Yout, 'b');
```

```
%Third Evaluations
figure()
X3base = 1994; % Use a base to decrease error coming from computing on large
numbers
X3= [0 1 2 3 4 5 6 7 8 9];
Y3base = 67;
Y3= [0.052 1.008 2.803 5.024 6.400 5.063 7.669 7.487 7.065 9.777];

% Vandermonde Interpolation
subplot(3,2,1);
hold on
[Xout, Yout] = vandermonde(X3, Y3, 200);
title("Vandermonde Matrix");
plot(X3 + X3base, Y3 + Y3base, 'ro');
plot(Xout + X3base, Yout + Y3base, 'b');

% Newton Interpolation
subplot(3,2,2);
hold on
[Xout, Yout] = newton(X3, Y3, 200);
title("Newton Method");
plot(X3 + X3base, Y3 + Y3base, 'ro');
plot(Xout + X3base, Yout + Y3base, 'b');

% Lagrange Interpolation
subplot(3,2,3);
hold on
[Xout, Yout] = lagrange(X3, Y3, 200);
title("Lagrange Method");
plot(X3, Y3, 'ro');
plot(Xout, Yout, 'b');
% Natural Cubic Spline
subplot(3,2,4);
hold on
[Xout, Yout] = cubicSpline(X3, Y3, 'natural', 200);
title("Natural Cubic Spline");
plot(X3 + X3base, Y3 + Y3base, 'ro');
plot(Xout + X3base, Yout + Y3base, 'b');

% Complete Cubic Spline
subplot(3,2,5);
hold on
[Xout, Yout] = cubicSpline(X3, Y3, 'complete', 200);
title("Complete Cubic Spline");
plot(X3 + X3base, Y3 + Y3base, 'ro');
plot(Xout + X3base, Yout + Y3base, 'b');

% Not-a-Knot Cubic Spline
subplot(3,2,6);
hold on
[Xout, Yout] = cubicSpline(X3, Y3, 'not-a-knot', 200);
title("Not-a-Knot Cubic Spline");
plot(X3 + X3base, Y3 + Y3base, 'ro');
plot(Xout + X3base, Yout + Y3base, 'b');
```

```
% Part C Evaluations
figure()

XCFive = 0:5:100;
XCDaily = 0:1:100;

YCTempFive = [13 34 51 13 64 61 54 24 23 65 31 51 60 46 55 47 71 47 36 50 83];
YCTempDaily = [13 ... 83];

YCRainFive = [0 0 0.37 0 0.13 0 0 0 0 0 0.79 0 0 0 0 0 0 0.01 0.07 0.01 0];
YCRainDaily = [0 ... 0.1];

% Newton Interpolation - Temp
subplot(2,2,1)
hold on
[Xout, Yout] = newton(XCFive, YCTempFive, 101);
title("Newton Interpolation - Temp");
xlabel("Days since Jan 1");
ylabel("Temperature (F)");
plot(XCDaily, YCTempDaily, 'ro');
plot(Xout, Yout, 'b');
max(abs(Yout - YCTempDaily))

% Newton Interpolation - Rain
subplot(2,2,2)
hold on
[Xout, Yout] = newton(XCFive, YCRainFive, 101);
title("Newton Interpolation - Rain");
xlabel("Days since Jan 1");
ylabel("Rainfall (in.)");
plot(XCDaily, YCRainDaily, 'ro');
plot(Xout, Yout, 'b');
max(abs(Yout - YCRainDaily))

% Natural Cubic Spline - Temp
subplot(2,2,3)
hold on
[Xout, Yout] = cubicSpline(XCFive, YCTempFive, 'natural', 101);
title("Natural Cubic Interpolation - Temp");
xlabel("Days since Jan 1");
ylabel("Temperature (F)");
plot(XCDaily, YCTempDaily, 'ro');
plot(Xout, Yout, 'b');
max(abs(YCTempDaily - Yout))

% Natural Cubic Spline - Rain
subplot(2,2,4)
hold on
[Xout, Yout] = cubicSpline(XCFive, YCRainFive, 'natural', 101);
title("Natural Cubic Interpolation - Rain");
xlabel("Days since Jan 1");
ylabel("Rainfall (in.)");
plot(XCDaily, YCRainDaily, 'ro');
plot(Xout, Yout, 'b');
max(abs(YCRainDaily - Yout))
```