Galen Vincent

Mathematical Physics

Project 2

Get the data

Here I import the data from the CSV file that holds it all:

In[1]:=
```
SetDirectory[NotebookDirectory[]]
```

Out[1]=
```
C:\Users\galen\Desktop\Fall 2017\Math Physics\Project 2\GVincent_Project2
```

In[2]:=
```
idata = Import["DATAP2.csv"];
```

This data is for the average temperature of 28 different locations around Colorado each month from January 2007 to December 2016. Most data sets have a few missing months, of course.

Delete all of the excess zeros that got imported in with the data. Each set of data is a different length, so this makes each list within the 'data' variable its own length.

In[3]:=
```
For[n = 1, n ≤ Length[idata], n ++,
 For[m = 1, m ≤ Length[idata[[n]]], m++, If[idata[[n, m]] == 0,
   idata[[n]] = Delete[idata[[n]], Table[{j}, {j, m, Length[idata[[n]]]}]]]]]
```

Break up the data so that each of the 28 sets can be accessed by calling a number 1-28 (aka break down the data in to 28 paired data sets):

In[4]:=
```
data = Table[0, {n, 1, Length[idata] / 2}];
```

In[5]:=
```
For[n = 1, n ≤ (Length[idata] / 2), n++,
  data[[n]] =
   Table[{idata[[2 * n - 1, m]], idata[[2 * n, m]]}, {m, 1, Length[idata[[2 * n]]]}]];
```

Break up the data so that data can be accessed by month (aka break down the data into 120 sets of 28ish points each):

In[6]:=
```
mdata = Table[{}, {n, 1, 120}];
```

In[7]:=
```
For[m = 1, m <= (Length[idata] / 2), m++,
 For[n = 1, n ≤ Length[data[[m]]], n++,
  j = data[[m, n, 1]];
  AppendTo[mdata[[j]], data[[m, n, 2]]]]]
```

Analysis Part 1: Large-scale features: From Linear Regression to Covariance

Define common plot options:

In[8]:=
```
plotoptions = {Frame → True, LabelStyle → {Black, FontFamily → "Helvitica", FontSize → 15}};
```

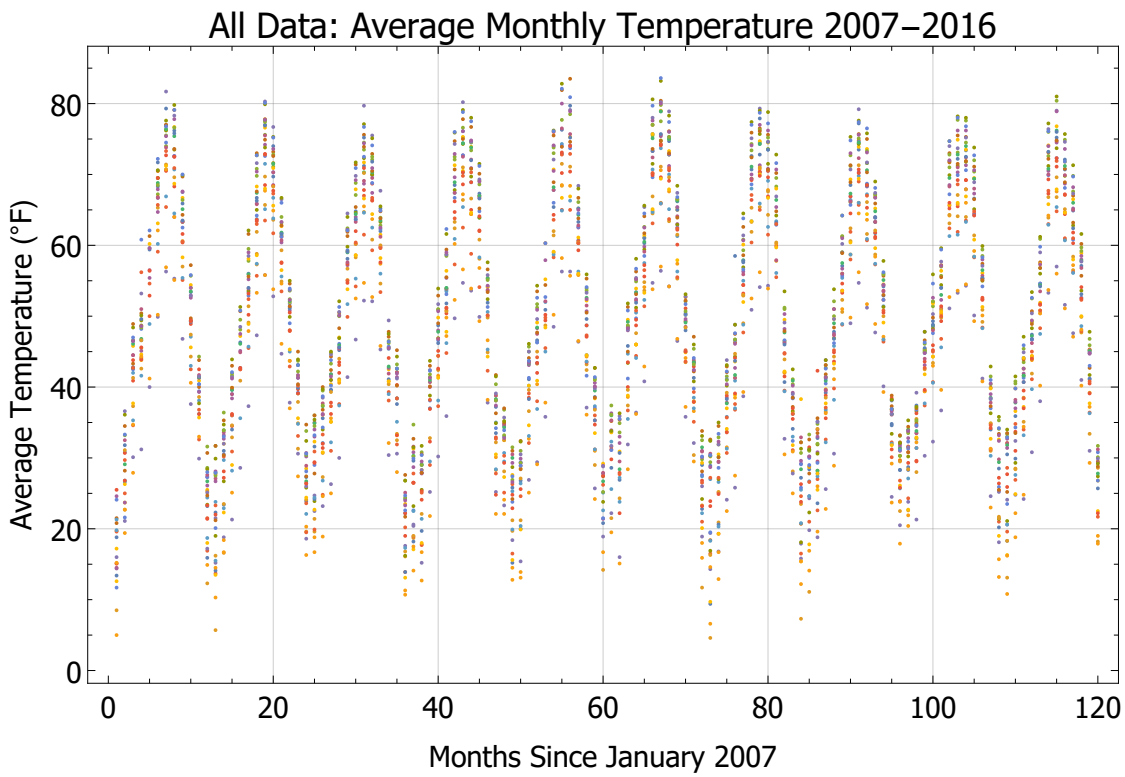
a)

Put the data into one huge set and plot it:

In[9]:=
```
bigdata = Flatten[data, 1];
```

In[10]:=
```
p2 = ListPlot[data, plotoptions,
  PlotLabel → "All Data: Average Monthly Temperature 2007-2016", FrameLabel →
   {"Months Since January 2007", "Average Temperature (°F)"}, GridLines → Automatic]
```

Out[10]=



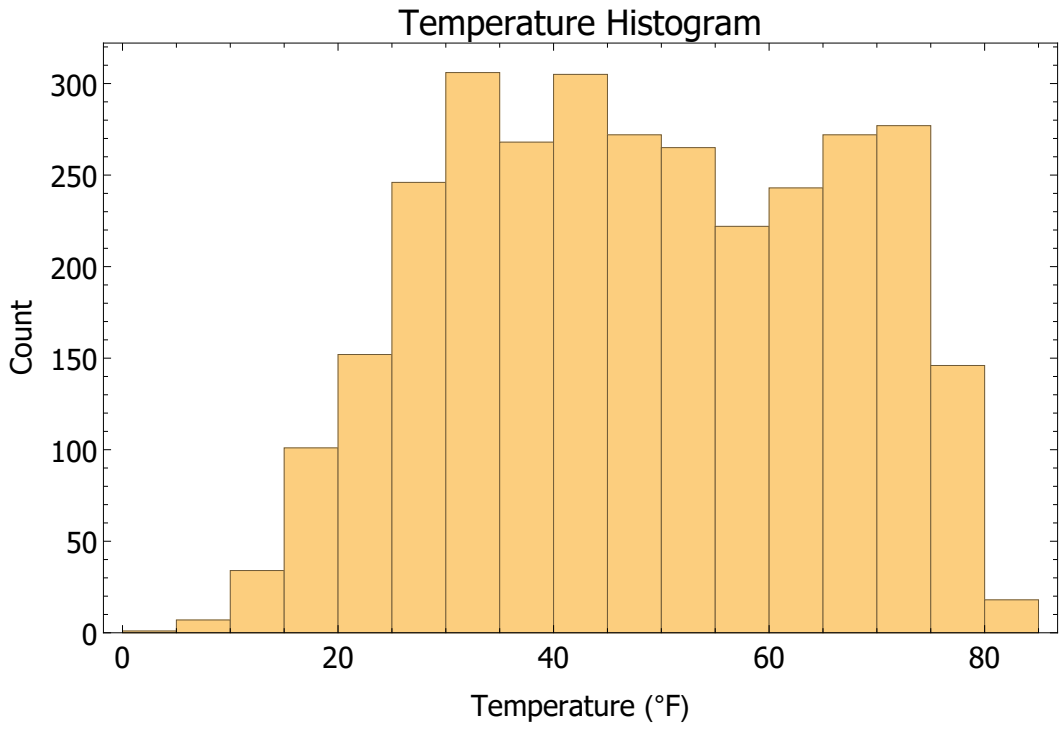
Qualitatively, a very definite oscillatory pattern can be seen, with some noise and some fluctuation year to year. This makes sense when we think about the source of this data. Average monthly temperature

will oscillate with the seasons, so the evident period is likely around a year long. The noise in the data makes sense because these average temperatures were taken from 28 different locations across Colorado. Since every location will record a somewhat different temperature, we see some noise in the data.

b)

In[11]:=
```
Histogram[Flatten[mdata], plotoptions,
 PlotLabel → "Temperature Histogram", FrameLabel → {"Temperature (°F)", "Count"}]
```

Out[11]=



Here, I bin the data set into the counts of temperature values, irregardless of time value (for now). This gives us information about how often certain temperature value are measured over the course of 10 years. We can see that the distribution is not all that telling, as we have no indication of when the temperature values were taken. I will do a different sort of binning below to get a better idea of what this data is telling us.

I now bin the data so that it is grouped into years. I justify this binning because there are fluctuations year-to-year in the temperature data, but if I am trying to look at the overall trend of temperatures as time goes on, I need to just look at the data from a year as a whole.

First I break down the data so that it is grouped into years:

In[12]:=
```
ydata = mdata;
```

In[13]:=
```
ydata = Table[Join[mdata[[(12 * n + 1)]], mdata[[(12 * n + 1) + 1]], mdata[[(12 * n + 1) + 2]],
    mdata[[(12 * n + 1) + 3]], mdata[[(12 * n + 1) + 4]], mdata[[(12 * n + 1) + 5]],
    mdata[[(12 * n + 1) + 6]], mdata[[(12 * n + 1) + 7]], mdata[[(12 * n + 1) + 8]],
    mdata[[(12 * n + 1) + 9]], mdata[[(12 * n + 1) + 10]], mdata[[(12 * n + 1) + 11]]], {n, 0, 9}];
```

Then I can find the mean temperature of each year:

In[14]:=
```
μy = Table[Mean[ydata[[n]]], {n, 1, 10}]
```

Out[14]=
```
{47.8469, 47.1997, 47.1487, 48.0429, 47.1127, 51.1906, 46.7375, 48.1912, 49.4694, 49.1775}
```

Then I can find the standard deviation of the temperatures for each year:

In[15]:=
```
σy = Table[StandardDeviation[ydata[[n]]], {n, 1, 10}]
```

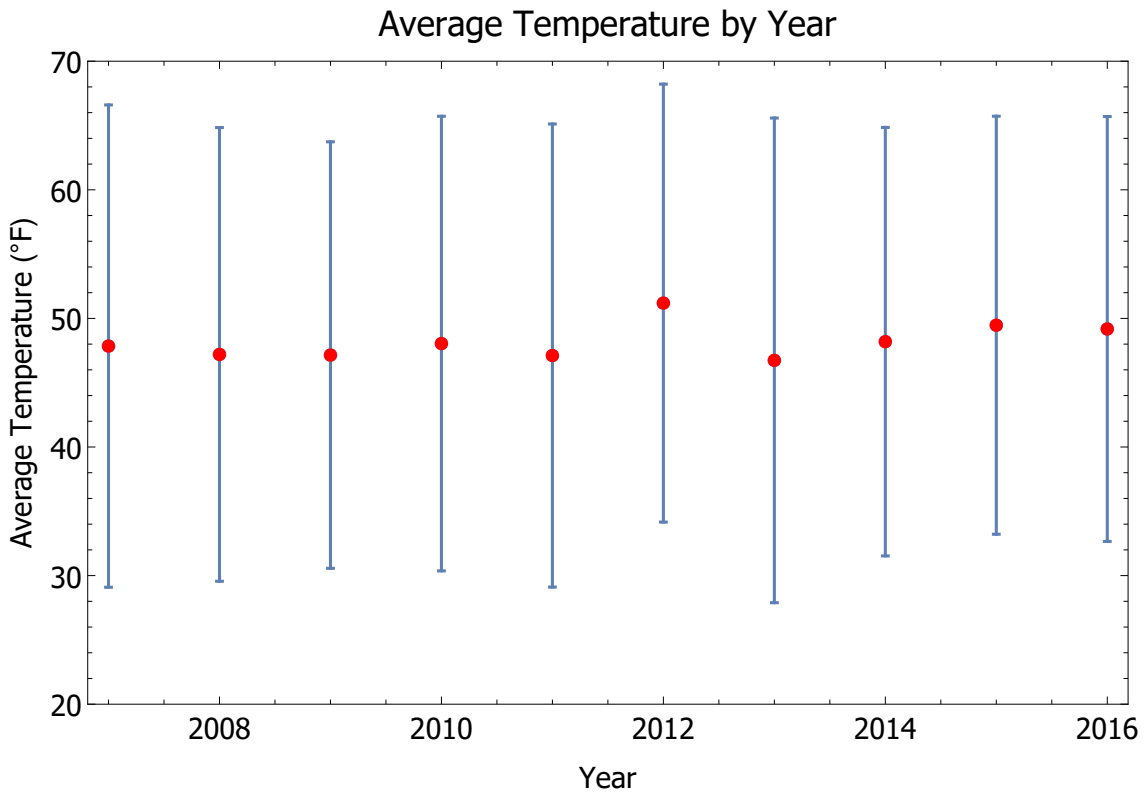
Out[15]=
```
{18.753, 17.6409, 16.5826, 17.6754, 18.0047, 17.0271, 18.8421, 16.6593, 16.252, 16.5205}
```

Now, I can visualize these bins by plotting the mean of each year along with the standard deviation of the data for each year as error bars:

In[16]:=
```
Needs["ErrorBarPlots`"]
```

In[17]:=
```
Show[ErrorListPlot[Table[{{2007 + n - 1, μy[[n]]}, ErrorBar[σy[[n]]]}, {n, 1, Length[μy]}],
  PlotRange → {20, 70}, PlotMarkers → None, plotoptions,
  PlotLabel → "Average Temperature by Year",
  FrameLabel → {"Year", "Average Temperature (°F)"}],
 ListPlot[Table[{2007 + n - 1, μy[[n]]}, {n, 1, Length[μy]}], PlotStyle → Red]]
```

Out[17]=



So this binning is much more telling. I can now see the average temperature, with error bars, for each year that I have data for. The error underneath these bins is caused by some strange statistics,

because there is a sinusoidal temperature change within each year. This is also why the error bars are so big, because there is so much variation in temperature within each year, which is a result of the data source. The data itself doesn't look to ragged, but the error bars on top of it definitely do, once again as a result of the data source.

A last option for binning, which I will actually use through this project, is binning by month. I choose to bin by month so that I can see the oscillations in temperature by month, and analyze them with the Fourier Transforms. This is also the convenient binning that I found my data in, so I will stick with it. I take the average by month, find the standard deviations, and plot the bins with error bars in part c, below.

c)

From here on out I will be using binning by month.

First I find the mean temperature of each month:

In[18]:=
```
μm = Table[Mean[mdata[[n]]], {n, 1, Length[mdata]}];
```

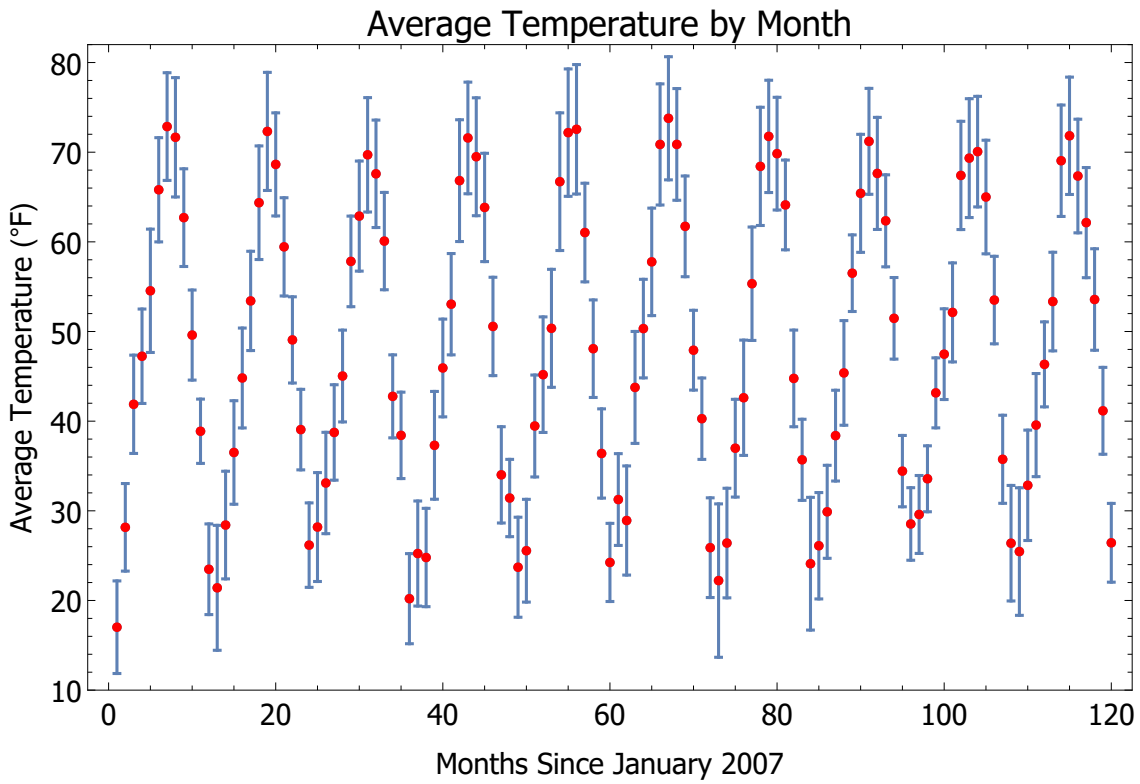
Then I find the standard deviation of the temperature of each month:

In[19]:=
```
σm = Table[StandardDeviation[mdata[[n]]], {n, 1, Length[mdata]}];
```

Now I plot the bins with error bars:

In[20]:=
```
p1 = Show[ErrorListPlot[
    Table[{{n, μm[[n]]}, ErrorBar[σm[[n]]]}, {n, 1, Length[μm]}], PlotRange → {10, 82},
    PlotMarkers → None, plotoptions, PlotLabel → "Average Temperature by Month",
    FrameLabel → {"Months Since January 2007", "Average Temperature (°F)"}],
  ListPlot[Table[{n, μm[[n]]}, {n, 1, Length[μm]}], PlotStyle → Red]]
```

Out[20]=

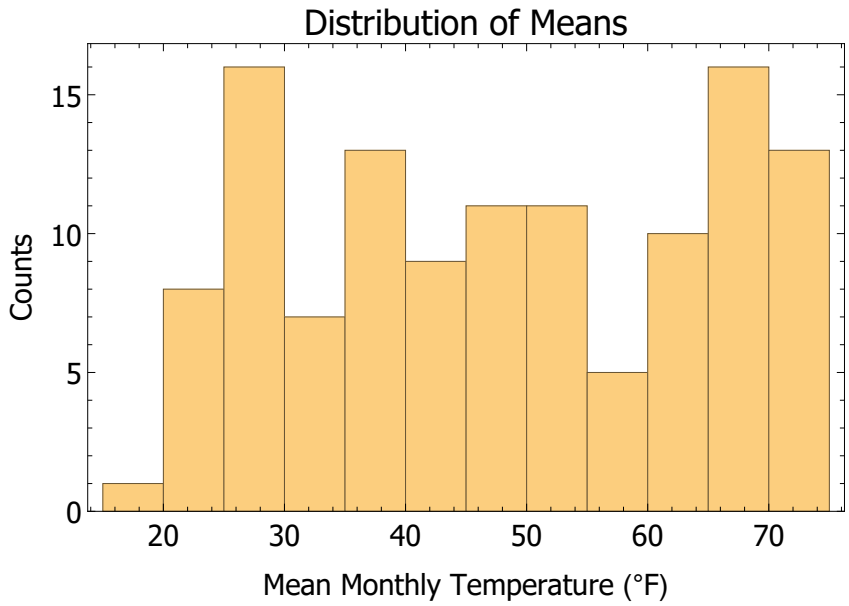


This is a plot of the means of my data. A few things to note about this plot: My data has no error bars in the x direction because the averages are by month, and there is no error in categorizing temperatures into month.

Thinking about the type of statistics behind the error in these bins, the error would be caused from different temperature readings from different locations. This data wouldn't be Gaussian, because the data points are being taken from different locations, so there would be some sort of different distribution based on the location of the 28 temperature readings.

In[21]:=
```
Histogram[μm, {5}, plotoptions, PlotLabel → "Distribution of Means",
 FrameLabel → {"Mean Monthly Temperature (°F)", "Counts"}]
```

Out[21]=



Above is the plot of the distribution of the means of each month of data. It makes sense that this distribution is kind of all over the place, becasue the means for each month are also quite all over the place. This makes sense as temperature changes so much month to month.

d)

Below I have made a 2D histogram of all of my data

First, I format my data into (x,y) pairs so I can use the Histogram 3D function:

In[22]:=
```
histdata =
  Table[Table[{n, mdata[[n, m]]}, {m, 1, Length[mdata[[n]]]}], {n, 1, Length[mdata]}];
```

In[23]:=
```
histdata = Flatten[histdata, 1];
```

Now, I create a table of the (x,y,z) coordinates of the means that I want to show with the data:

In[24]:=
```
μ3d = Table[{n, μm[[n]], 15}, {n, 1, Length[μm]}];
```

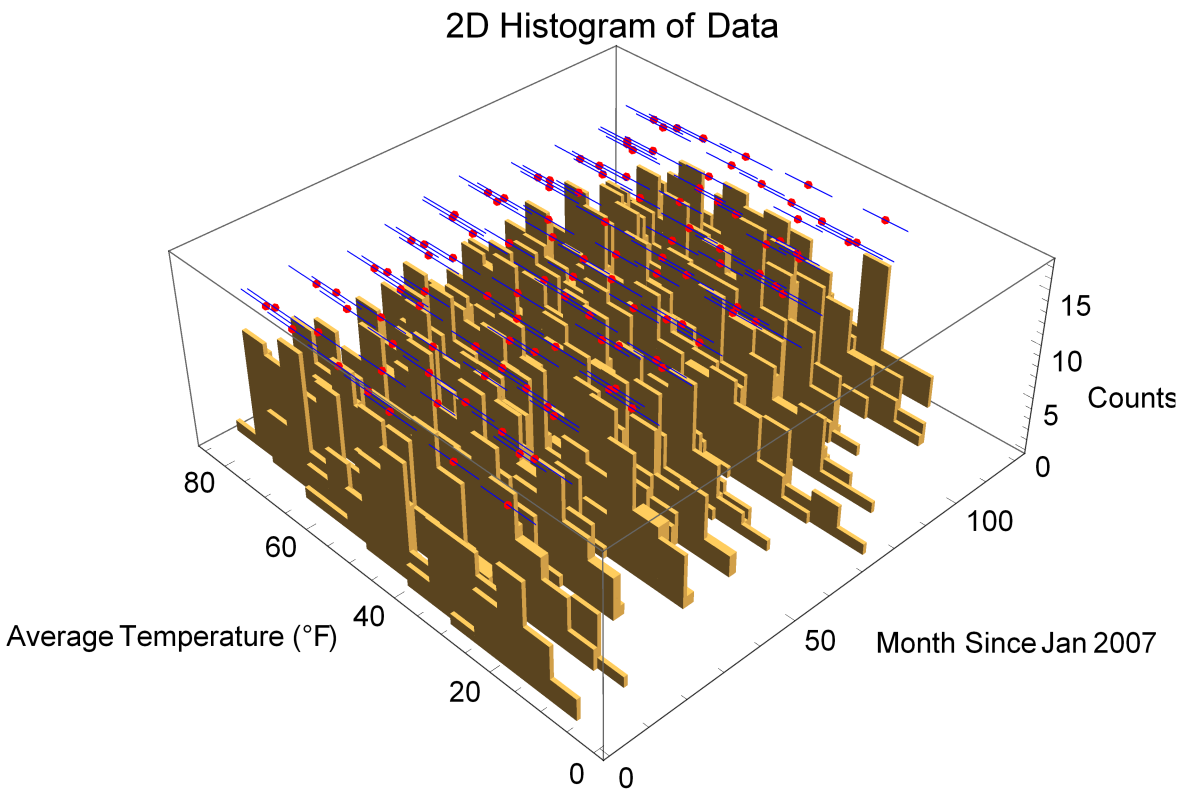
To plot the error bars, there isn't a defined error bar 3D function, so I created one of my own (with some help from the internet):

In[25]:=
```
ErrorBar3D[point_, error_] := Line[{point - {0, error, 0}, point + {0, error, 0}}];
```

Now I plot all three things and show them together. The 2D histogram, the averages, and the standard deviations. Once again, I only have standard deviations in average temperature, so there are only error bars in one direction. For the histogram, I specified bins to be by month, as they were in part c, so that the plots can be easily compared, and I specified the average temperature to be binned in 5 degree bins.

In[26]:=
```
p3 = Histogram3D[histdata, {{1}, {5}},
    LabelStyle → {Black, FontFamily → "Helvitica", FontSize → 15},
    AxesLabel → {"Month Since Jan 2007", "Average Temperature (°F)", "Counts"},
    PlotLabel → "2D Histogram of Data"];
p4 = ListPointPlot3D[Table[{n, μm[[n]], 15}, {n, 1, Length[μm]}],
    PlotStyle → {Red, PointSize[.01]}];
p5 = Graphics3D[Table[{Blue, ErrorBar3D[μ3d[[n]], σm[[n]]]}, {n, 1, Length[μ3d]}]];
Show[p3, p4, p5]
```

Out[29]=



In this plot, the histogram is in orange, the average value for each bin are the red dots, and the error bars for the bins are the blue lines. I would recommend moving around the plot in three dimensions to get some good views of what exactly it is showing.

This visualization has indeed clarified to me that this is 3D data.

e)

I can conclude that the data from month to month is not IID. This is very evident from the distribution of means in part c. If the data collected month to month was IID, then the distribution of means would be a Gaussian, but it is obviously not Gaussian. It also makes sense based on the data source, becasue the average temperature taken during different months is definitely not identically distributed, as different months will obviously have different temperatures. As a result, the distribution of means does not follow the central limit theorem. If I was to look at yearly averages, it might be a different story, as temperature

year to year does not have the obvious temperature change that month to month measurements have, but I unfortunately only have 10 years of data to work with, so it is difficult to analyze a distribution of these means. Theoretically, however, I would say that the year to year averages would be IID, and follow the central limit theorem.

f)

As a precursor to all of this linear fitting analysis, I know going in that the linear fits I am going to try aren't going to be great fits for my monthly data, as it is so oscillatory, but I am more interested in the overall trend of the data, and less so about the fit being a perfect match. I am hoping to see a slightly increasing model over the years, to show that the temperature around Colorado is increasing over time.

First, I find the best linear fit to my distributions: y(x) = A + Bx, with A & B determined by equations which we derived in class.

First, I format my monthly data averages into a form where it is {month, average temp}:

In[30]:=
```
μmpairs = Table[{n, μm[[n]]}, {n, 1, Length[μm]}];
```

Then I calculate A & B, with appropriate weights based on error under each point:

In[31]:=
```
n = Length[μmpairs];
w = 1/σm^2;
Δ = Sum[(w[[i]]), {i, 1, n}] * Sum[(w[[i]] μmpairs[[i, 1]]^2), {i, 1, n}] - (Sum[w[[i]] μmpairs[[i, 1]], {i, 1, n}])^2;
A = 1/Δ ((Sum[(w[[i]] μmpairs[[i, 1]]^2), {i, 1, n}]) * (Sum[(w[[i]] μmpairs[[i, 2]]), {i, 1, n}]) -
    (Sum[(w[[i]] μmpairs[[i, 1]]), {i, 1, n}]) * (Sum[(w[[i]] μmpairs[[i, 1]] * μmpairs[[i, 2]]), {i, 1, n}]));
B = 1/Δ ((Sum[(w[[i]]), {i, 1, n}]) * (Sum[(w[[i]] μmpairs[[i, 1]] * μmpairs[[i, 2]]), {i, 1, n}]) -
    (Sum[(w[[i]] μmpairs[[i, 1]]), {i, 1, n}]) * (Sum[(w[[i]] μmpairs[[i, 2]]), {i, 1, n}]));
linModel[
  x_] =
 A +
  B *
   x
```

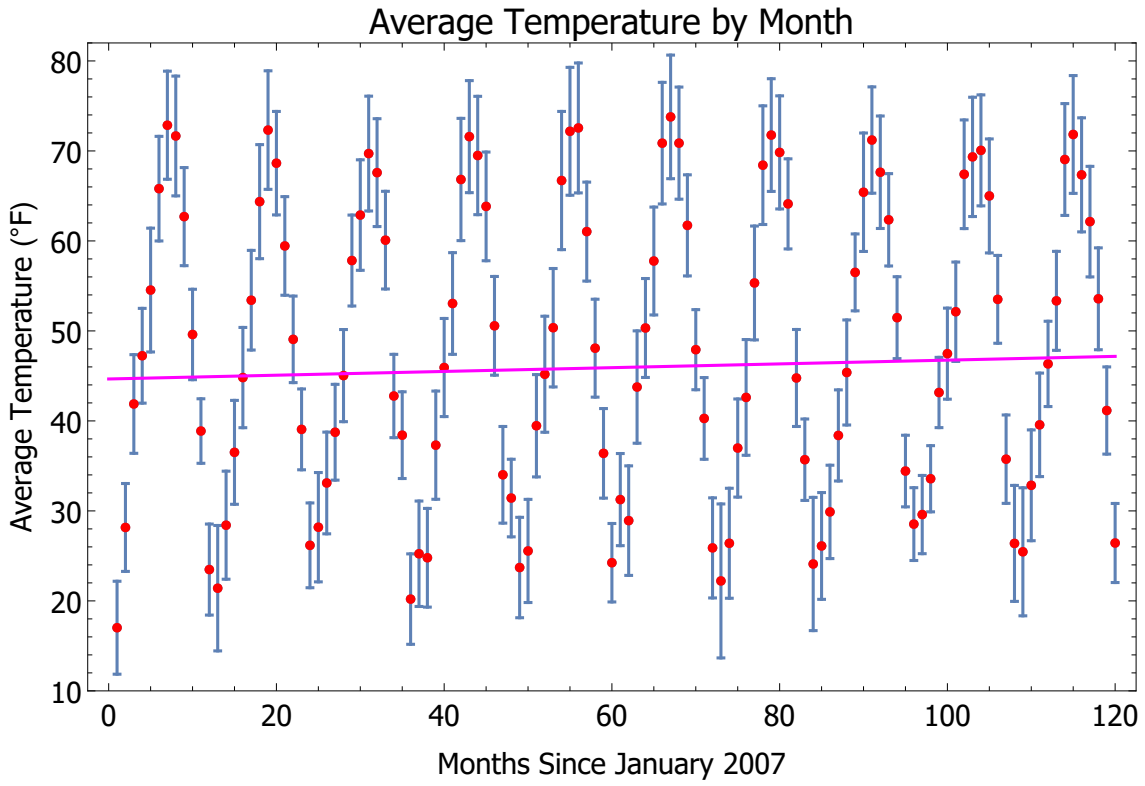
Out[36]=
$$44.6593 + 0.020938\, x$$

Now I plot this fit on top of my monthly data:

In[37]:=
```
p9 = Plot[linModel[x], {x, 0, 120}, PlotStyle → Magenta];
```

In[38]:=
```
Show[p1, p9]
```

Out[38]=



We can see that indeed there is a slight increase in the linear fit as time moves on. This is promising to show that the temperature in Colorado is indeed increasing over the years. First, we need to find errors in our fit parameters before we can come to a definite conclusion.

In class we defined the equation for error in A and B based on error propagation, with a constant σ_y for all data sets. I re-derived these formulas to include different weightings of each set of data to be:

$$\sigma_A = \sqrt{\frac{\sum_{i=1}^{n}(x_i^2/w_i)}{\Delta}}$$

$$\sigma_B = \sqrt{\frac{\sum_{i=1}^{n}(1/w_i)}{\Delta}}$$

I calculate these uncertainties for my data:

In[39]:=
```
σA = Sqrt[(Sum[w[[i]]^-1 μmpairs[[i, 1]]^2, {i, 1, n}])/Δ]
```

Out[39]=
```
30.7344
```

In[40]:=
```
σB = Sqrt[(Sum[w[[i]]^-1, {i, 1, n}])/Δ]
```

Out[40]=
```
0.442833
```

So I conclude that the fit parameters are really:

A = 44.6593 +/- 30.7344
B = 0.020938 +/- 0.442833

This doesn't bode well for the conclusion that temperature is increasing over time, becasue our positive slope in temperature does not stay positive for our entire error range. Overall, it is very difficult to make a good linear model out of this data, becasue of its sinusoidal nature.

g)

Once again, I go into this section of the project knowing that none of these fits are going to be very good for my data as a whole. However, I can still compare their χ^2 values to see which best fits the slightly increasing relationship.

First, I will try to fit a quadratic term to the data:

$y(x) = a_0 + a_1 x + a_2 x^2$

I'll solve for these coefficients using matrices.

In[41]:=
```
A2 = {{∑_{i=1}^{n} (w[[i]]), ∑_{i=1}^{n} (w[[i]] μmpairs[[i, 1]]), ∑_{i=1}^{n} (w[[i]] μmpairs[[i, 1]]^2)},
   {∑_{i=1}^{n} (w[[i]] μmpairs[[i, 1]]), ∑_{i=1}^{n} (w[[i]] μmpairs[[i, 1]]^2),
    ∑_{i=1}^{n} (w[[i]] μmpairs[[i, 1]]^3)}, {∑_{i=1}^{n} (w[[i]] μmpairs[[i, 1]]^2),
    ∑_{i=1}^{n} (w[[i]] μmpairs[[i, 1]]^3), ∑_{i=1}^{n} (w[[i]] μmpairs[[i, 1]]^4)}};
```

In[42]:=
```
MatrixForm[A2]
```

Out[42]//MatrixForm=

$$\begin{pmatrix} 4.03104 & 246.755 & 20088.3 \\ 246.755 & 20088.3 & 1.83129\times10^{6} \\ 20088.3 & 1.83129\times10^{6} & 1.77249\times10^{8} \end{pmatrix}$$

In[43]:=
```
b2 = {{∑_{i=1}^{n} (w[[i]] μmpairs[[i, 2]])}, {∑_{i=1}^{n} (w[[i]] μmpairs[[i, 1]] μmpairs[[i, 2]])},
   {∑_{i=1}^{n} (w[[i]] μmpairs[[i, 1]]^2 μmpairs[[i, 2]])}};
```

In[44]:=
```
MatrixForm[b2]
```

Out[44]//MatrixForm=

$$\begin{pmatrix} 185.19 \\ 11440.5 \\ 934824. \end{pmatrix}$$

In[45]:=
```
a = LinearSolve[A2, b2]
```

Out[45]=
```
{{44.3135}, {0.0382878}, {-0.000143717}}
```

And I now have the quadratic model:

In[46]:=
```
quadModel[x_] = a[[1, 1]] + a[[2, 1]] x + a[[3, 1]] x^2
```

Out[46]=
```
44.3135 + 0.0382878 x - 0.000143717 x^2
```

In[47]:=
```
p10 = Plot[quadModel[x], {x, 1, 120}, PlotStyle → {Dashed, Green}];
```

Next, I will try an exponential fit. This model will take the form:

$y(x) = b_0 e^{b_1 x}$

To make this model linear, we can take the natural log of each side:

$\ln(y) = \ln(b_0) + b_1 x$

Now we can apply our normal solving process, but just take the natural log of the y values, and we will solve for $\ln(b_0)$ (from which we can extract b_0) and b_1. Note that we also need to change the error of our y data now that we have taken the natural log of it. This means our weights need to be changed. If we say that z = ln(y), then $\sigma_z = \frac{1}{y}\sigma_y$ (from error propagation), so the weights become $\frac{1}{\sigma_z^2} = \frac{y^2}{\sigma_y^2}$

In[48]:=
```
wnew = μm^2/σm^2;
```

In[49]:=
```
A3 = {{∑_{i=1}^{n} (wnew[[i]])}, ∑_{i=1}^{n} (wnew[[i]] μmpairs[[i, 1]])},
   {∑_{i=1}^{n} (wnew[[i]] μmpairs[[i, 1]]), ∑_{i=1}^{n} (wnew[[i]] μmpairs[[i, 1]]^2)}};
```

In[50]:=
```
b3 = {{∑_{i=1}^{n} (wnew[[i]] Log[μmpairs[[i, 2]]])},
   {∑_{i=1}^{n} (wnew[[i]] μmpairs[[i, 1]] Log[μmpairs[[i, 2]]])}};
```

In[51]:=
```
b = LinearSolve[A3, b3]
```

Out[51]=
```
{{3.97616}, {0.0000956439}}
```

We can extract b_0 from the results

In[52]:=
```
b0 = Exp[b[[1, 1]]];
```

In[53]:=
```
b1 = b[[2, 1]];
```

And I now have the exponential model:

In[54]:=
```
expModel[x_] = b0 Exp[b1 x]
```

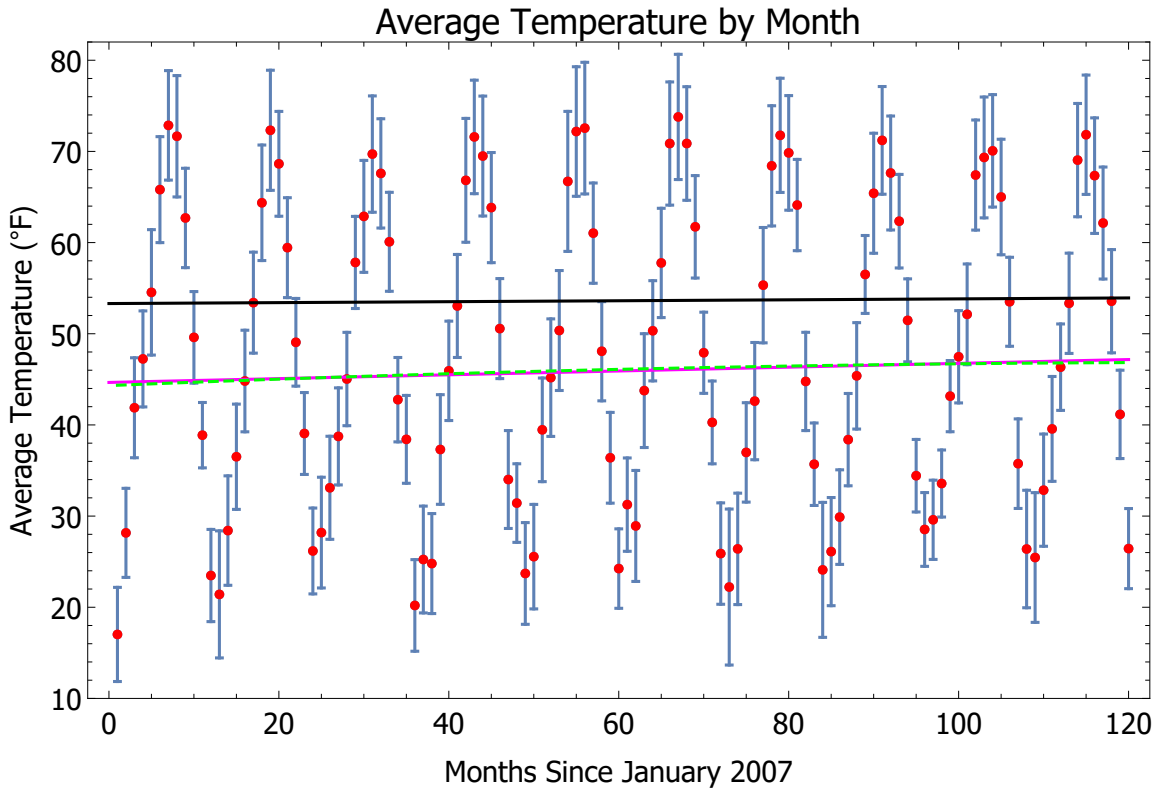
Out[54]=
```
53.3121 ℮^(0.0000956439 x)
```

In[55]:=
```
p11 = Plot[expModel[x], {x, 0, 120}, PlotStyle → Black];
```

Now I can show both of my new fits, along with my linear fit, all on one plot.

In[56]:=
```
Show[{p1, p9, p10, p11}]
```

Out[56]=



In this plot, the black line is the exponential fit, the magenta line is the linear fit, and the green dotted line is the quadratic fit. Interestingly enough the exponential fit is so much higher than the other two fits becasue when I took the natural log of my y values, the data points associated with the higher average temperatures are clumped closer together (becasue that is how a logarithm works), and as a result, the fit gets pulled up towards those values. This is why the exponential fit is so much higher that the other two. Right away, we can qualitatively tell that this fit isn't going to be the best one. Also we can see that the quadratic and linear fits are very close to one another.

To quantitatively analyze the different fits, we can look at the reduced χ^2 of each of them. I do this below:

In[57]:=
```
χ2[fit_, dataset_, σ_] :=
  Sum[(dataset[[i, 2]] - fit[dataset[[i, 1]]])^2 / σ[[i]]^2, {i, 1, Length[dataset]}];
```

First, the linear fit (2 Degrees of Freedom):

In[58]:=
```
linearχ02 = χ2[linModel, μmpairs, σm] / (Length[μmpairs] - 2)
```

Out[58]=
```
8.25572
```

Next, the quadratic fit (3 Degrees of Freedom):

```
In[59]:= quadχ02 = χ2[quadModel, μmpairs, σm] / (Length[μmpairs] - 3)
Out[59]= 8.32549
```

Lastly, the exponential fit (2 Degrees of Freedom):

```
In[60]:= expχ02 = χ2[expModel, μmpairs, σm] / (Length[μmpairs] - 2)
Out[60]= 10.2835
```

As I expected, the exponential's reduced χ^2 is much higher than either the linear or quadratic fit, so we can immediately throw it out of the race for best fit for the trend my data. The reduced χ^2 for the linear and quadratic fits aren't anywhere near being on average 1 standard deviation away from my data, but they are still informative to which is the better fit. I already knew that these fits were going to be horrible for actually modeling my very periodic data, but the slightly smaller reduced χ^2 value for the linear fit tells me the linear relationship does the best job at describing the overall trend in my data. So I choose the linear model to be the best model to describe the slightly increasing trend of my data.

h)

The linear correlation coefficient is defined as:

$$r = \frac{\sigma_{xy}}{\sigma_x \sigma_y} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}}$$

I can do these calculations with my ginormous data set:

```
In[61]:= rdata = Flatten[data, 1];

In[62]:= monthavg = N[Sum[rdata[[n, 1]], {n, 1, Length[rdata]}] / Length[rdata]]
Out[62]= 61.0625

In[63]:= tempavg = Sum[rdata[[n, 2]], {n, 1, Length[rdata]}] / Length[rdata]
Out[63]= 48.222

In[64]:= covariance =
           Sum[(rdata[[n, 1]] - monthavg) (rdata[[n, 2]] - tempavg), {n, 1, Length[rdata]}];

In[65]:= sigmay = √(Sum[(rdata[[n, 1]] - monthavg)^2, {n, 1, Length[rdata]}])
Out[65]= 1928.94

In[66]:= sigmax = √(Sum[(rdata[[n, 2]] - tempavg)^2, {n, 1, Length[rdata]}])
Out[66]= 975.702
```

In[67]:=
$$r = \frac{\text{covariance}}{\text{sigmax} * \text{sigmay}}$$

Out[67]= 0.0527055

So the linear correlation coefficient is 0.0527055. Based on this value, and the fact that there are over 3000 data points in my ginormous data set, I can come to the conclusion that there is a 0.316128 % chance that this data is uncorrelated (from an online table). Basically, even though the r value seems low, becasue it is calculated from so many data points, it tells us that the data is almost definitely correlated, in a positive linear fashion, becasue the r value is positive.

Is my data consistent with a linear relationship? According to the linear correlation coefficient, there is indeed a high likely hood that there is some positive linear relationship to all of the data. Based on the linear correlation coefficient, I can say with 99.7% confidence that there is a positive linear relationship in my ginormous data set.

What can I say about the complete data set vs the averaged one? I can say that the complete data set is definitely linearly correlated, as we showed with the correlation coefficient. However, I have a lot of uncertainty with the averaged data set's linear fit found in part (f). How do these statements compare? It seems that there is more uncertainty surrounding my averaged data set than my ginormous one, becasue of the fact that when I average my data, the uncertainty in these average values becomes very large, based on the source of the points underneath these averages.

Basically, from this section, I conclude that there is indeed a positive linear relationship in my data, meaning that as time move forward, temperatures increase. However, a linear fit is not a good encompassing fit for the data. This is quite obvious just by looking at the plot of the linear fit on the data. Also, I don't fully trust the uncertainty in my linear fit parameters. This stems from the fact that the data under the average monthly temperatures came from different locations, so that doesn't exactly make perfect sense to count as "uncertainty". I think that taking the average of each of the 28 locations into one monthly average gives a value that can be thought of as a statewide temperature average for that month. Because I don't really trust the uncertainties under the averages, I think that the error in linear fit parameters we found in part (f) is better than we quantitatively showed. Because of this, and the correlation coefficient, I come to the conclusion that there has indeed been an increasing linear relationship in average temperature in Colorado over the last 10 years.

Analysis Part 2: Small-scale features: Fourier transforms

a)

One way to get a perfect fit on my data would be to create a polynomial fit of order n-1, where n is my number of data points. This works becasue of the fact that in a n-1 order polynomial, there are n fit

parameters, which equally match the n data points, allowing for a 100% perfect fit. For example, two points can always be connected with a first order polynomial (y = m x + b). This method could be solved “by hand” using Mathematica and matrices, but it would be quite tedious, so I will just use a built in Mathematica function to find the fit.

A second way to get a perfect fit is to use a method called spline interpolation. This method often works much better than the polynomial fit described above. This method creates a piecewise function, with a different polynomial function between each of the data points. This method takes into account things like curvature between points, and is much better at capturing the trend of the data than the polynomial method. I at one point learned how to calculate these splines by hand (in MATH 307 - Introduction to Scientific Computing), but it is quite tedious, so I will use a built in Mathematica function for this as well.

First, I find the perfect polynomial fit to my monthly data:

I know that I have 120 data points, so I need an order 119 polynomial (can’t say I’ve ever used one of those before). After some playing around on Mathematica, it seems that trying to fit an order 119 polynomial doesn’t work so well. My guess is that this is because of the massive errors that can arise when taking a number to the 119th power. To show that this method is theoretically valid, I instead just fit a 17th order polynomial to my first 18 data points:

In[68]:=
```
fits = Table[x^i, {i, 1, 17}];
```

In[69]:=
```
perfectFit1 = LinearModelFit[μmpairs[[1 ;; 18]], fits, x];
```

This gives us a perfect fit function of:

In[70]:=
```
Normal[perfectFit1]
```

Out[70]=
$$113\,566. - 387\,095.\,x + 569\,903.\,x^2 - 488\,694.\,x^3 + 276\,272.\,x^4 - 110\,115.\,x^5 + 32\,231.\,x^6 - 7109.65\,x^7 + 1201.08\,x^8 - 156.782\,x^9 + 15.856\,x^{10} - 1.23823\,x^{11} + 0.0739009\,x^{12} - 0.00330608\,x^{13} + 0.000107236\,x^{14} - 2.37983\times10^{-6}\,x^{15} + 3.23061\times10^{-8}\,x^{16} - 2.02225\times10^{-10}\,x^{17}$$

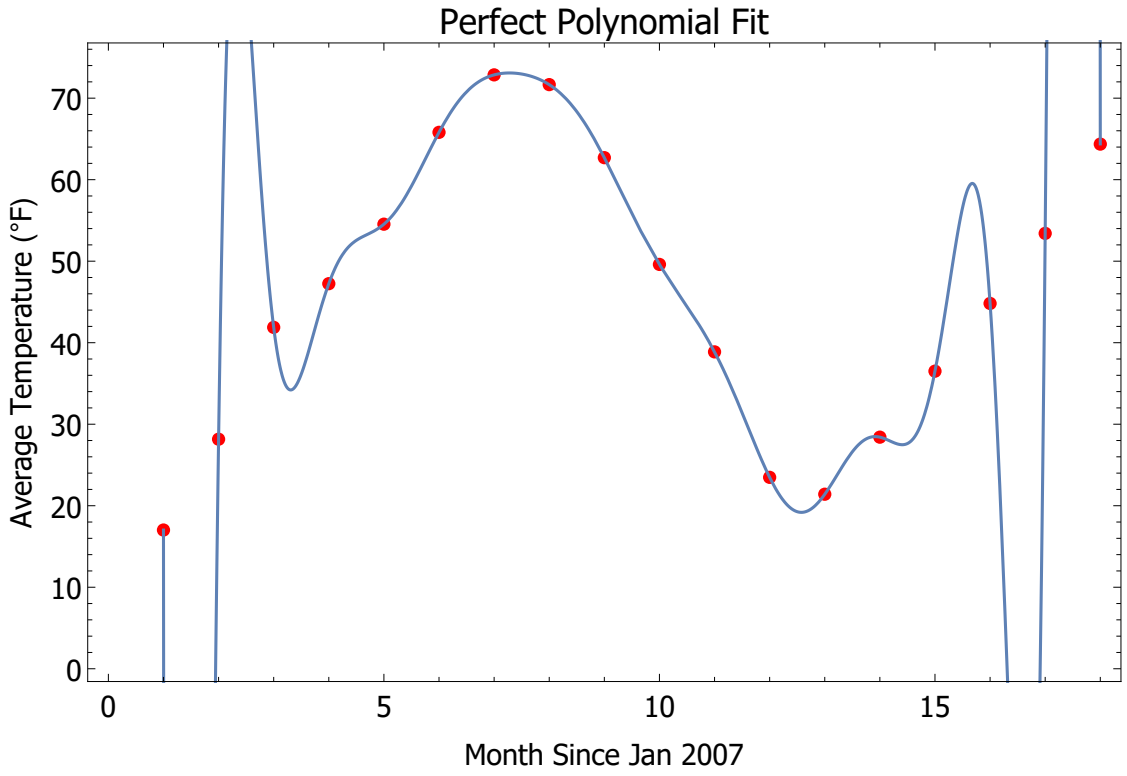
We can now plot the results to see what the fit looks like:

In[71]:=
```
p12 = Plot[perfectFit1[x], {x, 1, 18}, PlotRange → All];
```

In[72]:=
```
p13 = ListPlot[μmpairs[[1 ;; 18]], PlotStyle → Red, plotoptions,
   FrameLabel → {"Month Since Jan 2007", "Average Temperature (°F)"},
   PlotLabel → "Perfect Polynomial Fit"];
```

In[73]:=
```
Show[p13, p12]
```

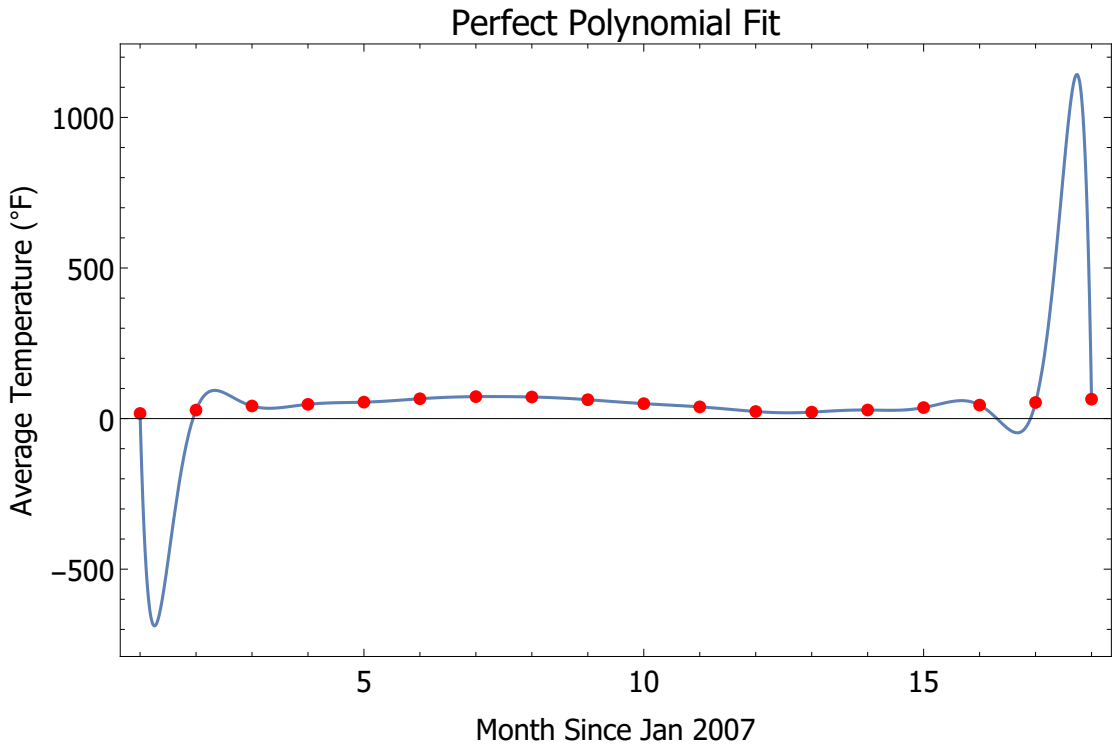
Out[73]=



With this range on the plot, we can tell that the function does indeed go through each of the data points. To get a better view of how bad this function actually is as a model of the data, we can zoom out and capture the whole range of the function:

In[74]:=
```
Show[p12, p13, plotoptions,
 FrameLabel → {"Month Since Jan 2007", "Average Temperature (°F)"},
 PlotLabel → "Perfect Polynomial Fit"]
```

Out[74]=



Now we can see just how bad of a fit the “perfect polynomial” really is. There are huge spikes on either end of the data set. This figure makes it abundantly clear that even though the fit goes through each of the points in the model, it is not a good model for telling us anything about the data.

Now I will look at the spline interpolation method for getting a perfect fit to my data:

In[75]:=
```
perfectFit2 = Interpolation[μmpairs];
```

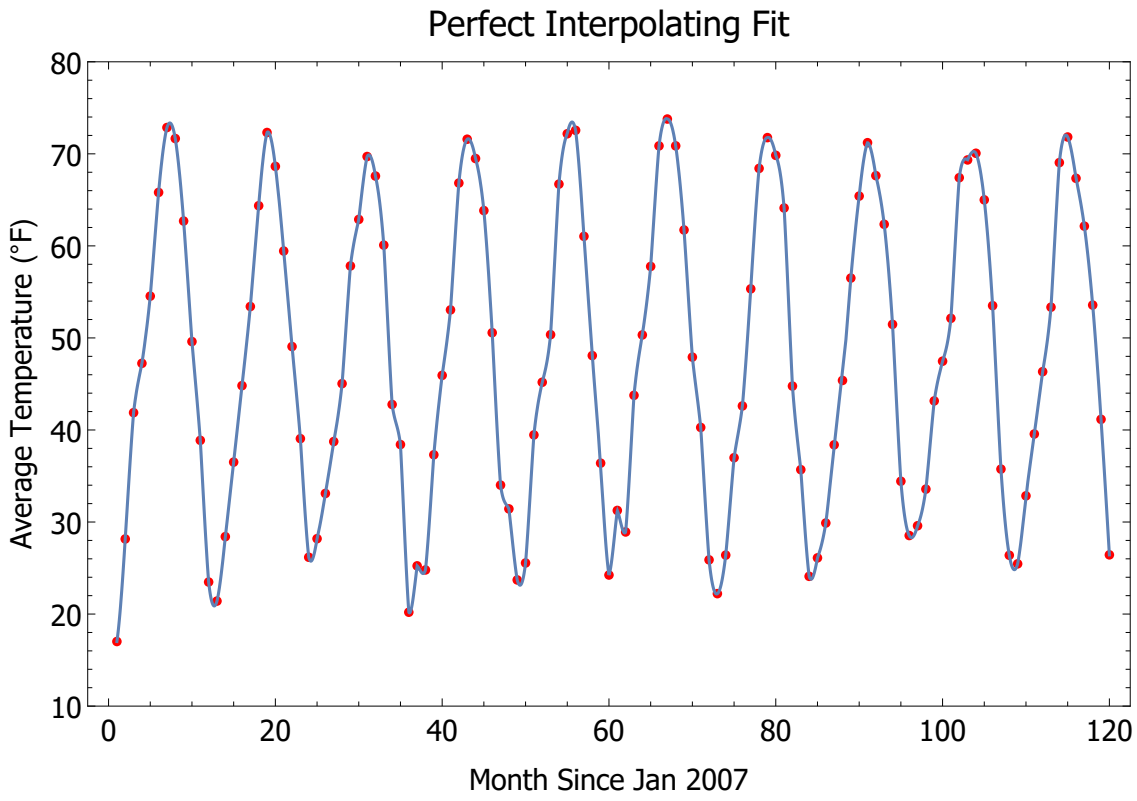
It is hard to write out an equation for this fit, as it is really a piecewise function with 119 different polynomial segments, but the model is contained within Mathematica, and we can visualize it with a plot.

In[76]:=
```
p14 = Plot[perfectFit2[x], {x, 1, 120}];
```

In[77]:=
```
p15 = ListPlot[μmpairs, PlotStyle → Red, plotoptions,
   FrameLabel → {"Month Since Jan 2007", "Average Temperature (°F)"},
   PlotLabel → "Perfect Interpolating Fit", PlotRange → {10, 80}];
```

In[78]:=
```
Show[p15, p14]
```

Out[78]=



Here we see a fit that is actually not bad. It goes through each data point, and it actually follows the trend in the data well. The only problem is that it follows the data a bit too closely. This data is not perfect, and we are looking for a model that captures the general trend over the years, not each and every specific detail of the year. This is especially evident around month 60, where we see the model follow the data perfectly, causing an unnatural spike in the fit.

How are these models different? The polynomial fit is one distinct function, while the interpolation actually consists of different functions between each of the data points. As a result, for the polynomial to have a perfect fit, it needs to go to really high and really low y values. However, for the interpolating function, as there can be different functions between each point, the fit isn't constrained to follow one function for the entire domain, and this results in a much cleaner fit. If I had to choose a perfect fit, I would choose the interpolating polynomial.

b)

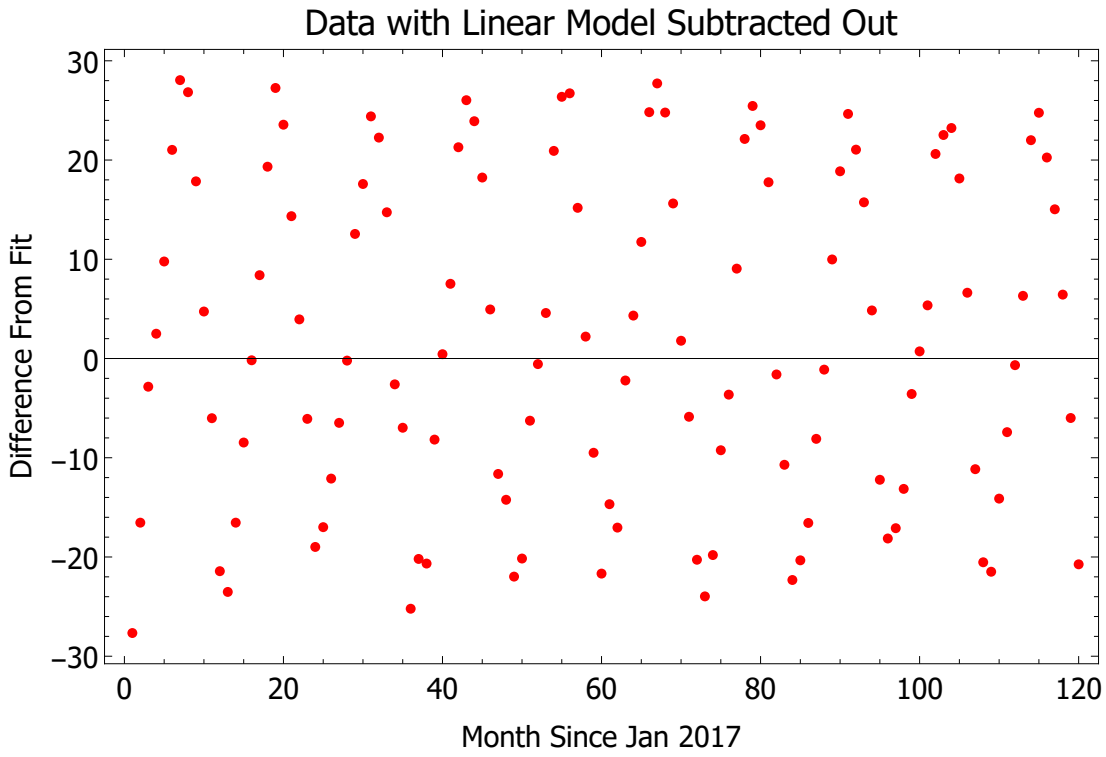
In part 2(g), I determined that the simple linear fit was the best fit for the overall trend of my data. I will now subtract this linear fit from the data to get data that varies around y = 0:

In[79]:=
```
μmpt2 = Table[{μmpairs[[i, 1]], μmpairs[[i, 2]] - linModel[μmpairs[[i, 1]]]},
   {i, 1, Length[μmpairs]}];
```

Here is a plot of the resulting data:

In[80]:=
```
p17 = ListPlot[μmpt2, plotoptions,
  FrameLabel → {"Month Since Jan 2017", "Difference From Fit"},
  PlotLabel → "Data with Linear Model Subtracted Out", PlotStyle → {Red, PointSize[0.01]}]
```

Out[80]=



Here we do indeed see data that varies around y = 0, and looks sort of noisy without any fit to connect the points. I know that there is more than noise under this data. This data is temperature by month for 10 years, so there is going to be oscillations in the values based on the season for which it was taken in. The winter will have lower temperature values, and the summer will have higher. This pattern will repeat year after year, causing an oscillating pattern.

c)

First, I define a function to take the discrete fourier transform of my binned monthly data:

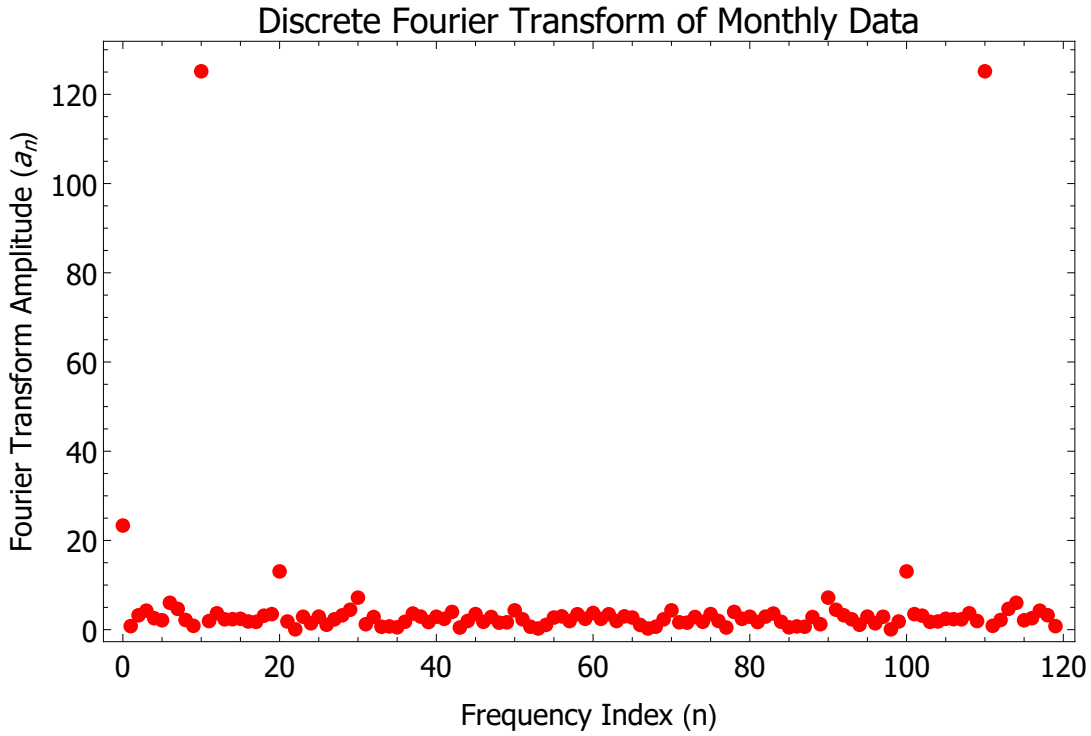
In[81]:=
```
DFT[datalist_] := Table[1/√Length[datalist] Sum[datalist[[k + 1]] Exp[-2 π I (i) (k)/Length[datalist]],
    {k, 0, Length[datalist] - 1}], {i, 0, Length[datalist] - 1}];
```

In[82]:=
```
discreteFT = DFT[μmpt2[[All, 2]]];
```

Now we can look at the magnitudes of the weights of each associated frequency:

In[83]:=
```
ListPlot[Table[{i, Abs[discreteFT[[i + 1]]]}, {i, 0, Length[discreteFT] - 1}],
 PlotRange → All, plotoptions, PlotStyle → {Red, PointSize[0.015]},
 FrameLabel → {"Frequency Index (n)", "Fourier Transform Amplitude (a_n)"},
 PlotLabel → "Discrete Fourier Transform of Monthly Data"]
```

Out[83]=



For this plot, the x axis represents a frequency index, and the y axis represents the weight of each of these frequencies. What this means is that this plot tells us about what frequency components are present in my data.

Right away from analyzing the plot above, I can see that there are two frequency components that are weighted much higher than any of the others. These frequencies occur at n values of 10 and 110, as shown below. These corresponds to Mathematica list indices of 11 and 111 (as Mathematica starts at index 1, but we start our frequency index at n = 0).

In[84]:=
```
Abs[discreteFT[[11]]]
```

Out[84]= 125.17

In[85]:=
```
Abs[discreteFT[[111]]]
```

Out[85]= 125.17

What does this mean about my data in particular? It confirms my idea that my data is very periodic. The Fourier Transform of a sine or cosine wave has two spikes, which correspond to the positive and negative values for its given frequency. I can extract the frequency out of the Fourier Transform by figuring out the frequency associated with index 10. I don't have to worry about index 110, as it just represents the negative component of the same frequency as the 10th index. To find the frequency associated with this index, I can apply a simple formula:

ω_n = n $\left(\frac{2\pi}{NT}\right)$, where n is the frequency index, N is the total number of data points, and T is the distance between each of the data points. T is one (month) for my data. N = 120, and n = 10.

In[86]:=
```
ωimportant = 10 * 2 π / 120
```

Out[86]= $\frac{\pi}{6}$

Which we can convert from angular frequency into Hz by saying f = $\frac{\omega}{2\pi}$:

In[87]:=
```
fimportant = ωimportant / (2 π)
```

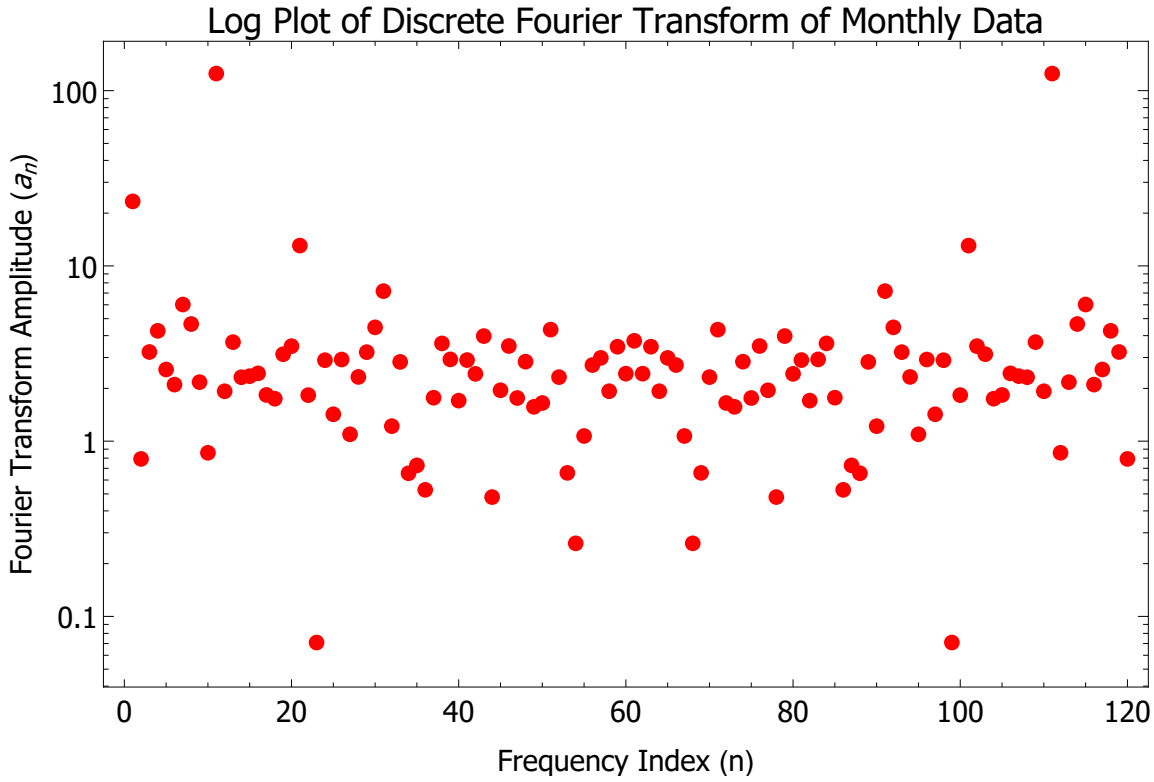
Out[87]= $\frac{1}{12}$

And we find a result that is exactly what I was expecting. The most important frequency is $\frac{1}{12\,\text{months}}$, or a period of 12 months. This makes sense as seasonal changes in temperature, because the temperature repeats itself every 12 months!

Now I will look at a log plot of the Discrete Fourier Transform, to see if there any small features I missed in the regular linear plot.

In[88]:=
```
ListLogPlot[Abs[discreteFT], PlotRange → All,
 plotoptions, PlotStyle → {Red, PointSize[0.015]},
 FrameLabel → {"Frequency Index (n)", "Fourier Transform Amplitude (a_n)"},
 PlotLabel → "Log Plot of Discrete Fourier Transform of Monthly Data"]
```

Out[88]=



There isn't anything too telling here. Most of the frequencies are still very grouped together, except for a high point at n = 0, which corresponds to a frequency of zero (or a constant), so that isn't all that telling. I also see the high points at n = 10 & 110. Other than those points, all I can see special about this plot are the two frequencies at around n = 23 and n = 98 which are not significant at all, and have amplitudes close to zero, but that isn't all too important for characterizing my data.

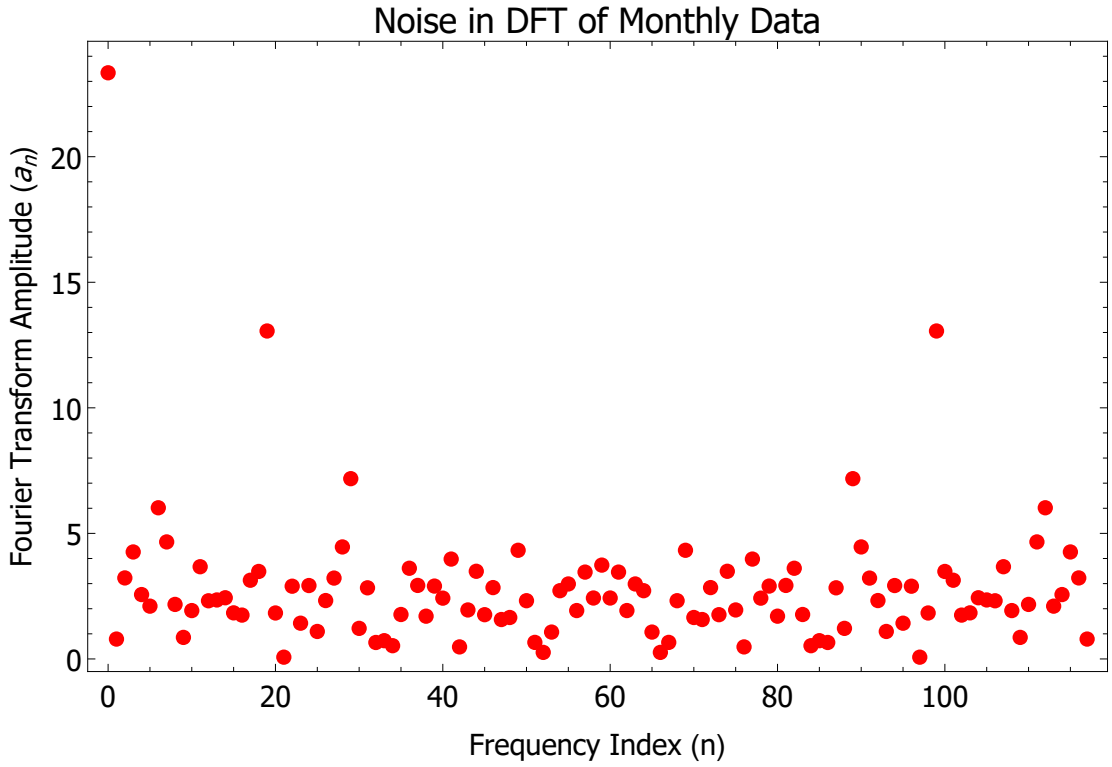
d)

To have a better look at the noise in the signal, I will get rid of the two points that are obviously significant:

In[89]:=
```
noiseDFT = Delete[discreteFT, {{11}, {111}}];
```

In[90]:=
```
ListPlot[Table[{i, Abs[noiseDFT[[i + 1]]]}, {i, 0, Length[noiseDFT] - 1}],
 PlotRange → All, plotoptions, PlotStyle → {Red, PointSize[0.015]},
 FrameLabel → {"Frequency Index (n)", "Fourier Transform Amplitude (a_n)"},
 PlotLabel → "Noise in DFT of Monthly Data"]
```

Out[90]=



After some research into different kinds of noise, I have found that the different “colors” of noise correspond to how the frequency amplitudes (a_n) act as the frequency changes. For example, in pink noise, a_n decreases as frequency goes up. In blue noise, a_n increases as frequency goes up, etc. There are many different colors of noise for different frequency responses.

For my data, I am predicting white noise, which is just a flat distribution of a_n values across the frequency range. This is evident in the graph above, as the weights seem to be evenly distributed across the whole frequency range.

It is also apparent from the graph that there are some frequency values (besides the ones I already removed), which are above the white noise boundary. One of these values is at n = 0, which as I mentioned before just corresponds to a zero frequency, so we don’t care too much about it. The other two peaks which appear to be above the noise level once again occur twice, indicating the positive and negative value of that important frequency. We can look at just the lower n value peak:

In[91]:=
```
Abs[noiseDFT[[20]]]
```

Out[91]=
```
13.0598
```

This occurs at n = 21 (Mathematica list index 20). I conclude that this may also be an important frequency component of my data, as it stands out above the white noise of the rest of the frequencies. I

can calculate the frequency associated with this index:

In[92]:=
```
ωimportant2 = 21 * 2 π / 120
```

Out[92]=
$$\frac{7\,\pi}{20}$$

In[93]:=
```
fimportant2 = ωimportant2 / (2 π)
```

Out[93]=
$$\frac{7}{40}$$

So there is a potentially important frequency of $\frac{7}{40}$ months^{-1}. I will attempt to include this in my overall model.

e)

I will develop a model for my data which includes both the linear model that I found in Analysis Part 1, and the important frequency components which I just found. To do this, I will fit some sines and cosines with my important frequency components to the data points which have my linear model subtracted out. Then I will add these sines and cosines to my linear model to get a full model for my data which correctly characterizes the linear trend, and the wiggles.

In[94]:=
```
μmpt2;
```

I will attempt to fit the following sines and cosines, which include the important frequency components that I found:

$$y(x) = c_0 \sin\left(\frac{\pi}{6} x\right) + c_1 \cos\left(\frac{\pi}{6} x\right) + c_2 \sin\left(\frac{7\pi}{20} x\right) + c_3 \cos\left(\frac{7\pi}{20} x\right)$$

I'll use Mathematica to solve for this fit:

In[95]:=
```
freqModel = LinearModelFit[μmpt2, {Sin[π/6 x], Cos[π/6 x], Sin[7 π/20 x], Cos[7 π/20 x]}, x];
```

In[96]:=
```
Normal[freqModel]
```

Out[96]=
$$2.1308 - 19.1656 \cos\left[\frac{\pi x}{6}\right] - 0.280212 \cos\left[\frac{7\,\pi x}{20}\right] - 12.4471 \sin\left[\frac{\pi x}{6}\right] - 0.182096 \sin\left[\frac{7\,\pi x}{20}\right]$$

I now add this model that I found for the wiggles to my linear model, and get out a final model for my data:

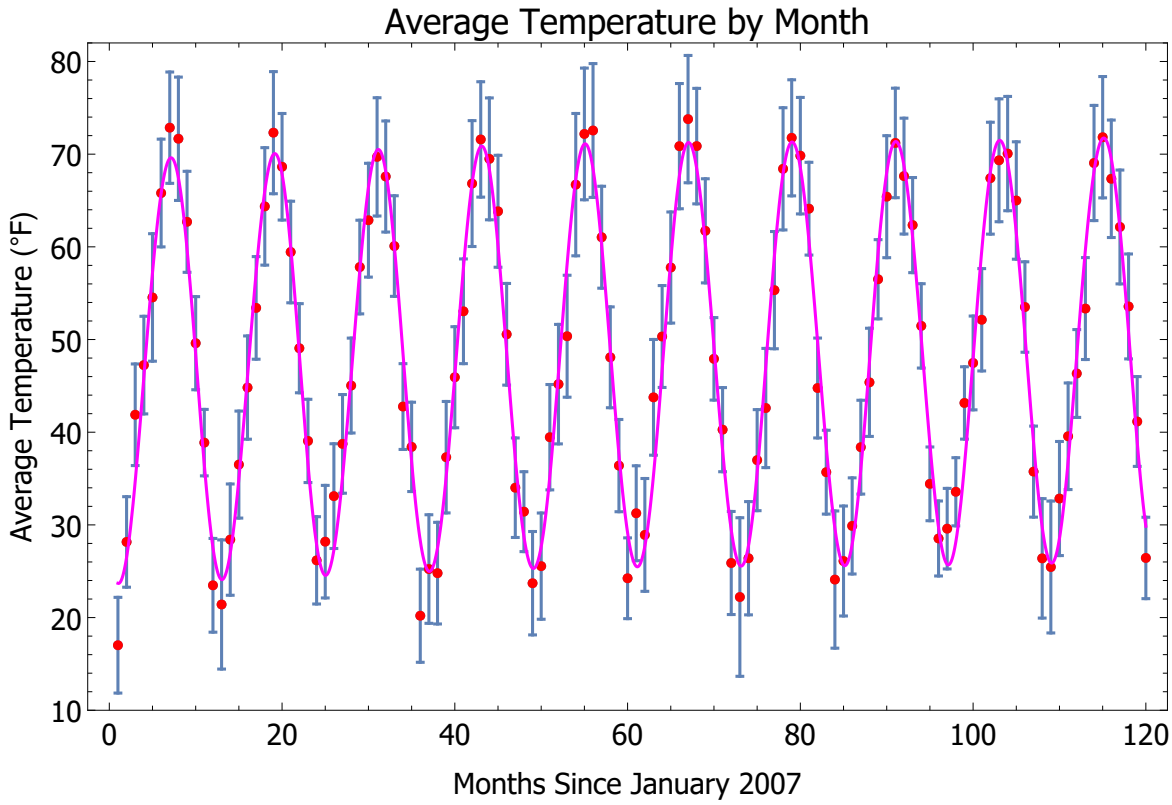
In[97]:=
```
Clear[totalModel]
```

In[98]:=
```
totalModel[y_] = freqModel[y] + linModel[y];
```

In[99]:=
```
p18 = Plot[totalModel[x], {x, 1, 120}, PlotStyle → Magenta];
```

In[100]:=
```
Show[p1, p18]
```

Out[100]=



The model looks pretty dang good on top of all the data!

The frequency content of this model matches the frequency content of the data well, becasue I designed it specifically to do so. To check this, I can take the full continuum Fourier Transform of my model and see the frequency components that I worked into it jump back out:

In[101]:=
```
FTmodel = FourierTransform[totalModel[x], x, ω]
```

Out[101]=
```
(-7.02388 - 4.56448 i) DiracDelta[7 π - 20 ω] - (144.123 + 93.6011 i) DiracDelta[π - 6 ω] +
 117.286 DiracDelta[ω] - (144.123 - 93.6011 i) DiracDelta[π + 6 ω] -
 (7.02388 - 4.56448 i) DiracDelta[7 π + 20 ω] - (0. + 0.0524838 i) DiracDelta'[ω]
```

You can see that the fourier transform of my data has a few select important frequency components. These are indicated by the Dirac Delta functions in the transform. All a_n (frequency weights) are zero except for the frequencies that make the arguments in any of the Dirac Delta functions zero. This gives us important ω values of $\frac{7\pi}{20}$, $-\frac{7\pi}{20}$, $\frac{\pi}{6}$, $\frac{-\pi}{6}$, and 0. These match up perfectly with the important frequencies we found based on the discrete Fourier Transform, and intentionally put into our model. The frequency of 0 corresponds to the linear part of the model, as it has a frequency component of zero.

I attempted to show this stuff graphically, but it turns out it is really hard to plot Dirac Delta functions (since they go to infinity). Symbolically, however it makes perfect sense how we recover the important frequencies.

f)

To assume that the time resolution of our data goes to zero, I will just use the model that I found in the last part in between the domain of my data (1 to 120 months). I do this becasue to take the semi-discrete Fourier Transform, I need a continuous function, and I need to define a period for that function. The semi-discrete FT is then defined by:

$f(x) = \frac{1}{2} a_0 + \sum_{n=1}^{\infty} a_n \cos\left(\frac{n\pi x}{L}\right) + b_n \sin\left(\frac{n\pi x}{L}\right)$, where a_0, a_n, and b_n are defined by:

$a_0 = \frac{1}{L} \int_0^{2L} f(x)\, dx$

$a_n = \frac{1}{L} \int_0^{2L} f(x) \cos\left(\frac{n\pi x}{L}\right) dx$

$b_n = \frac{1}{L} \int_0^{2L} f(x) \sin\left(\frac{n\pi x}{L}\right) dx$

Where 2L = 120, based on my data. I calculate the formulas for these coefficients, and create tables of the first 120 coefficients below:

In[102]:=
```
L = 60;
```

In[103]:=
```
a0 = 1/L Integrate[totalModel[x], {x, 0, 2 L}]
```

Out[103]=
```
96.0929
```

In[104]:=
```
Clear[n];
an[n_] := 1/L NIntegrate[totalModel[x] Cos[n π x/L], {x, 0, 2 L}]
```

In[106]:=
```
bn[n_] := 1/L NIntegrate[totalModel[x] Sin[n π x/L], {x, 0, 2 L}]
```

In[107]:=
```
ans[[1]] = {0, a0};
```

... Set: Symbol ans in part assignment does not have an immediate value.

In[108]:=
```
bns[[1]] = {0, 0};
```

... Set: Symbol bns in part assignment does not have an immediate value.

In[109]:=
```
Clear[i]
```

In[110]:=
```
ans = Table[{i, an[i]}, {i, 1, 119}];
bns = Table[{i, bn[i]}, {i, 1, 119}];
ans = Prepend[ans, {0, a0}];
bns = Prepend[bns, {0, 0}];
```

NIntegrate: Numerical integration converging too slowly; suspect one of the following: singularity, value of the integration is 0, highly oscillatory integrand, or WorkingPrecision too small.

NIntegrate: NIntegrate failed to converge to prescribed accuracy after 9 recursive bisections in x near {x} = {63.2808}. NIntegrate obtained -4.61853×10^{-13} and 4.5155492605991`*^-13 for the integral and error estimates.

NIntegrate: Numerical integration converging too slowly; suspect one of the following: singularity, value of the integration is 0, highly oscillatory integrand, or WorkingPrecision too small.

NIntegrate: NIntegrate failed to converge to prescribed accuracy after 9 recursive bisections in x near {x} = $\{5.38055 \times 10^{-7}\}$. NIntegrate obtained -4.54747×10^{-13} and 1.7832939944238014`*^-10 for the integral and error estimates.

NIntegrate: Numerical integration converging too slowly; suspect one of the following: singularity, value of the integration is 0, highly oscillatory integrand, or WorkingPrecision too small.

General: Further output of NIntegrate::slwcon will be suppressed during this calculation.

NIntegrate: NIntegrate failed to converge to prescribed accuracy after 9 recursive bisections in x near {x} = {119.99999946194453520555001255437719415164465885936806444078683853 1}. NIntegrate obtained -2.45137×10^{-13} and 2.041625549217191`*^-10 for the integral and error estimates.

General: Further output of NIntegrate::ncvb will be suppressed during this calculation.

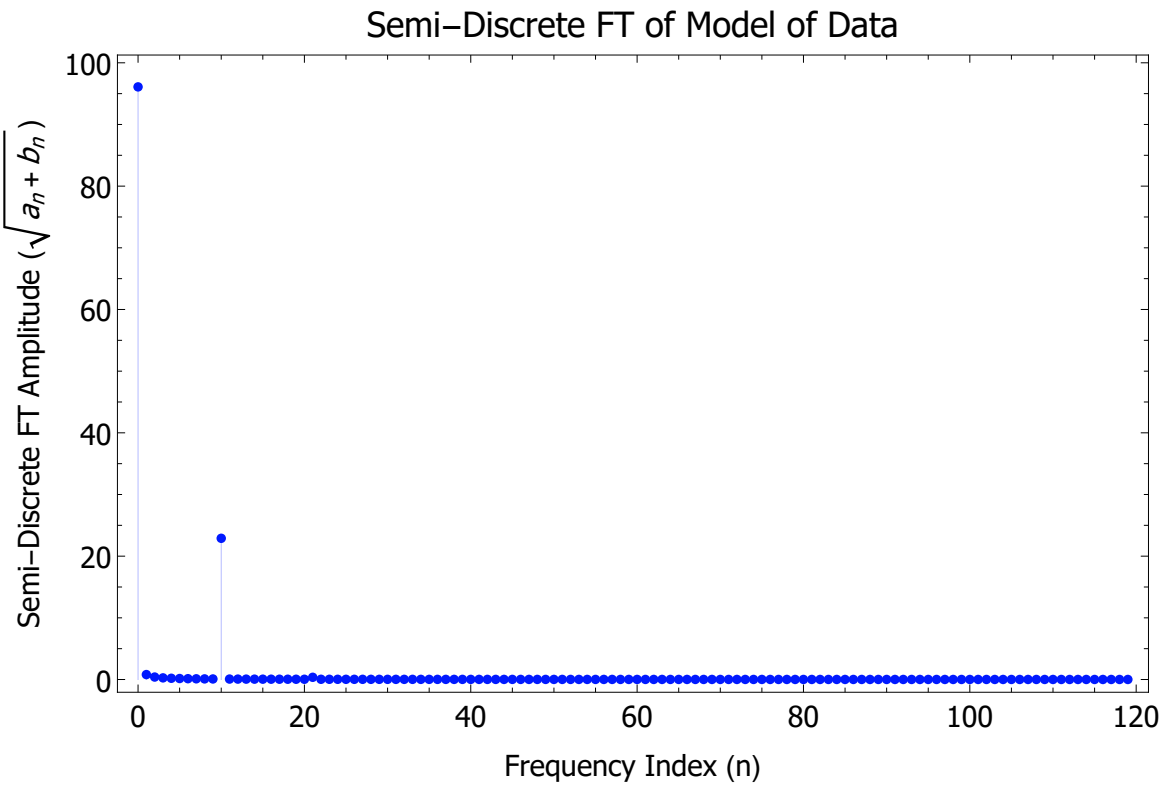
I can ignore all of these errors, as it is just telling me that NIntegrate doesn't like getting so close to zero (but its fine, I like the zeros).

To look at the total weights for each of the frequencies, I will look at the combination of a_n and b_n for each weight value, to get a better picture of the total weight associated with each frequency. I will do this by taking $\sqrt{a_n^2 + b_n^2}$.

In[114]:=
```
aplusb = Table[{i, Sqrt[ans[[i + 1, 2]]^2 + bns[[i + 1, 2]]^2]}, {i, 0, Length[ans] - 1}];
```

In[115]:=
```
ListPlot[aplusb, Filling → Axis, PlotRange → All, plotoptions,
 FrameLabel → {"Frequency Index (n)", "Semi-Discrete FT Amplitude (√(a_n + b_n))"},
 PlotLabel → "Semi-Discrete FT of Model of Data", PlotStyle → Hue[.65]]
```

Out[115]=



Here I can see graphically that the coefficients of my semi-discrete Fourier Transform do indeed re-create the Discrete Fourier Transform of my data. This is evident by the spike in frequency amplitudes at n = 0, n = 10, and a small spike at n = 21. These correspond to the exact n values for which I saw frequency amplitude spikes in my Discrete Fourier Transform. I show the values for these spikes below.

In[116]:=
```
aplusb[[1]]
```

Out[116]=
```
{0, 96.0929}
```

In[117]:=
```
aplusb[[11]]
```

Out[117]=
```
{10, 22.8965}
```

In[118]:=
```
aplusb[[22]]
```

Out[118]=
```
{21, 0.356369}
```

There are ∞ coefficients in the semi-discrete Fourier Transform, but if we take the first 120 of these coefficients, we can recover the 120 points obtained by the Discrete Fourier Transform. So, what I have discovered is that the coefficients of the Fourier Series of the model for my data are directly related to the Discrete Fourier Transform of my data set, when I take the period for my Fourier Series to be the length of my data set and take the first 120 coefficients.

g)

Here, I add 120 zeros to the end of my 120 point data set.

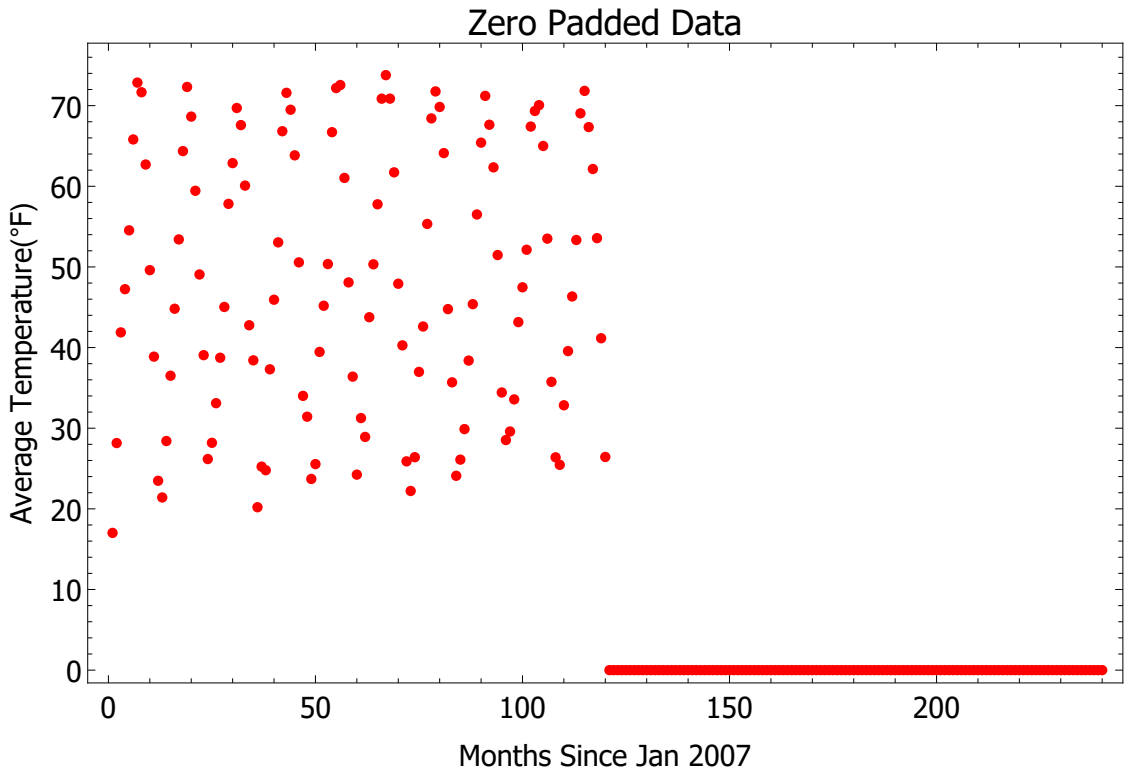
In[119]:=
```
zerosToAdd = Table[{120 + i, 0}, {i, 1, 120}];
```

In[120]:=
```
zeroPaddedData = Join[μmpairs, zerosToAdd];
```

Here is what it looks like graphically:

In[121]:=
```
ListPlot[zeroPaddedData, PlotRange → All, plotoptions, PlotStyle → {Red, PointSize[0.01]},
 FrameLabel → {"Months Since Jan 2007", "Average Temperature(°F)"},
 PlotLabel → "Zero Padded Data"]
```

Out[121]=



Indeed, this data with zero padding more closely represents a continuous function, for which a continuum Fourier Transform could be performed on. This is because a continuous function is essentially just a set of data points where there are an infinite number of points in a certain interval. Another way to think of it is to say that a continuous function is just a set of data points where the time step between points is infinitely small. So, when I zero pad by 120 data points with 120 more data points, we zoom out to a domain of [0,240] to be able to view all of the points, and as a result, the points all appear to be closer together, and appear to more closely represents a continuous function.

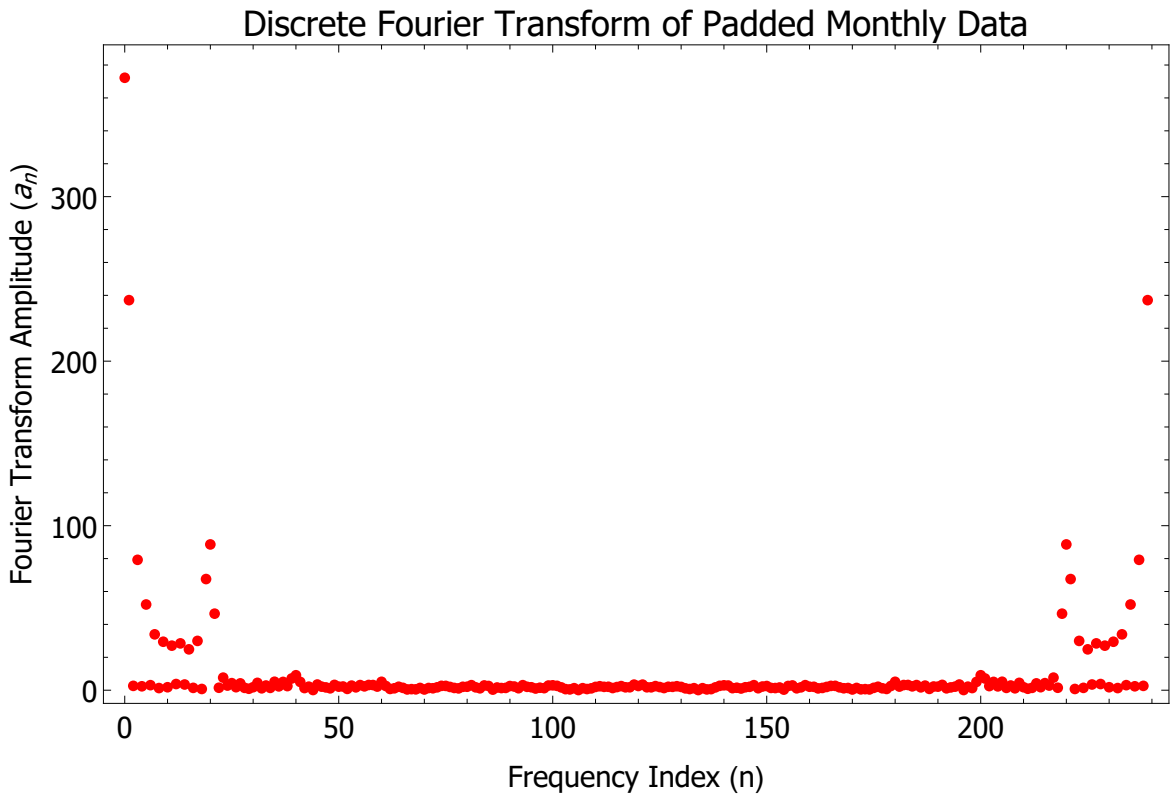
So, when I take the discrete Fourier Transform of this new set of data, there are twice as many frequencies to find significances for, becasue there are the same number of frequencies as there are data points. This means that as the number of points in my set increases, the number of frequencies to look at increase. A continuum Fourier Transform has an infinitely many frequencies to look at, so it is easy to

infer that as I increase the number of data points in my set, the discrete Fourier Transform of my data more closely approaches the continuum Fourier Transform.

In[122]:=
```
paddedFT = Abs[DFT[zeroPaddedData[[All, 2]]]];
```

In[123]:=
```
ListPlot[Table[{i, paddedFT[[i + 1]]}, {i, 0, Length[paddedFT] - 1}],
 PlotRange → All, plotoptions, PlotStyle → {Red, PointSize[0.01]},
 FrameLabel → {"Frequency Index (n)", "Fourier Transform Amplitude (a_n)"},
 PlotLabel → "Discrete Fourier Transform of Padded Monthly Data"]
```

Out[123]=



Here I have shown the DFT of my zero-padded data set. It is apparent just from looking at this plot that the values appear to be more “continuous” than they did for my 120 data point set. There are more frequencies to look at, so this plot more closely represents a continuum Fourier Transform. The actual values of amplitudes aren’t as important here, becasue my data set now contains a bunch of zeros, which throws off the nice frequencies that I saw when analyzing my regular data set.