Detrend_data

October 10, 2019

1 Remove Climatological Mean Annual Cycle and Detrend Data

This tutorial shows how to use CDAT to remove the climatological mean annual cycle and detrend data - a commonly-used technique prior to detailed climate data analysis of monthly anomalies.

The data considered in this notebook are monthly-mean surface air temperatures gridded over the United States and spanning the years 1850 - 1990. The original dataset is complete, but it is artificially modified in this notebook by "masking" some values, yielding an incomplete dataset with some values "missing" (which is a common problem in analysis of climate data). The analysis procedure involves three major steps:

1) Remove the climatological annual cycle yielding monthly-mean departures
2) Spatially average over the domain.
3) Remove the time-mean and the linear trend.

When there are missing values in the dataset (as in the sample calculations below), the final detrended time-series will depend on the order in which these steps are executed. Here we examine options for detrending the data, and we show that slightly different results are generated depending on the order in which operations are performed. More sophisticated treatments (not discussed here) involving appropriately weighting samples and collections of samples should be considered for datasets that only sparsely cover the time and space domains.

To download this Jupyter Notebook, right click on the link and choose "Download Linked File As..." or "Save Link as...". Remember where you saved the downloaded file, which should have an .ipynb extension. (You'll need to launch the Jupyter notebook or JupyterLab instance from the location where you store the notebook file.)

Consider Two Options

Processing Option A: Spatially average the fields over the domain, then remove the trend from the resulting time-series.

Processing Option B: Remove the trend at each grid cell, then spatially average the results.

Option A Corrected

2 Prepare Notebook and Data

Section **??**

[1]:
```
from __future__ import print_function
import cdms2
import MV2
import genutil
import cdutil
import vcs
import os
import requests
```

2.1 Download Data

The following CMIP5 NetCDF file contains Near-Surface Air Temperature data in degrees Kelvin (K) over North America. It is downloaded to the directory where this notebook is stored.

[2]:
```
filename = 'tas_Amon_IPSL-CM5A-LR_1pctCO2_r1i1p1_185001-198912.nc'
if not os.path.exists(filename):
    r = requests.get("https://cdat.llnl.gov/cdat/sample_data/notebooks/{}".
 ↪format(filename), stream=True)
    with open(filename,"wb") as f:
        for chunk in r.iter_content(chunk_size=1024):
            if chunk:  # filter local_filename keep-alive new chunks
                f.write(chunk)
```

2.2 Open Data File, Extract Variable

The following two lines of code open the file just downloaded to your local computer (`filename`), extract data for the Near-Surface Air Temperature (`tas`), and assign it to the variable `data`.

[3]:
```
f = cdms2.open(filename)

data = f("tas")
```

The following line of code uses the `.info()` method to allow us to take a quick look at the structure of the temperature data stored in the `data` variable.

There are 1680 different time values which are measured as "days since 1850-01-01 00:00:00". The range of the time values is the difference between the last value (51084.5) and the first value (15.5) which equals 51069 days. If we divide this range (51069) by the number of intervals in the dataset (1680-1), we get (51069/(1680-1)) = 30.416 days, which is the average time duration for each data point. This tells us that we are working with monthly data.

Since this file is a subset of the globe, the data cover 13 latitude bands and 16 longitude values. The first and last values of latitude (~25.6 to ~48.3) and longitude (-123.75 to -67.5) tell us the data cover the continental United States.

[4]:
```
data.info()
```

```
*** Description of Slab tas ***
id: tas
shape: (1680, 13, 16)
filename:
missing_value: 1e+20
comments:
grid_name: <None>
grid_type: generic
time_statistic:
long_name: Near-Surface Air Temperature
units: K
tileIndex: None
original_name: t2m
associated_files: baseURL: http://cmip-pcmdi.llnl.gov/CMIP5/dataLocation
gridspecFile: gridspec_atmos_fx_IPSL-CM5A-LR_1pctCO2_r0i0p0.nc areacella:
areacella_fx_IPSL-CM5A-LR_1pctCO2_r0i0p0.nc
coordinates: height
standard_name: air_temperature
cell_methods: time: mean (interval: 30 minutes)
cell_measures: area: areacella
history: 2011-03-07T11:45:34Z altered by CMOR: Treated scalar dimension:
'height'. 2011-03-07T11:45:34Z altered by CMOR: replaced missing value flag
(9.96921e+36) with standard missing value (1e+20). 2011-03-07T11:45:34Z altered
by CMOR: Inverted axis: lat.
Grid has Python id 0x120ed86a0.
Gridtype: generic
Grid shape: (13, 16)
Order: yx
** Dimension 1 **
   id: time
   Designated a time axis.
   units:  days since 1850-01-01 00:00:00
   Length: 1680
   First:  15.5
   Last:   51084.5
   Other axis attributes:
      axis: T
      calendar: noleap
      realtopology: linear
      long_name: time
      standard_name: time
   Python id:  0x121e16208
** Dimension 2 **
```

```
   id: lat
   Designated a latitude axis.
   units:  degrees_north
   Length: 13
   First:  25.578947067260742
   Last:   48.31578826904297
   Other axis attributes:
      axis: Y
      realtopology: linear
      long_name: latitude
      standard_name: latitude
   Python id:  0x120ec5668
** Dimension 3 **
   id: lon
   Designated a longitude axis.
   units:  degrees_east
   Length: 16
   First:  -123.75
   Last:   -67.5
   Other axis attributes:
      axis: X
      modulo: 360.0
      topology: circular
      realtopology: circular
      long_name: longitude
      standard_name: longitude
   Python id:  0x120ff12b0
*** End of description for tas ***
```

3 Data Exploration

Section **??**

First, to get a feel for the data, let's spatially average the data over the entire domain to create a time series of the mean temperature for the region. In creating this time series, the `averager` function will take the temperature data for the entire region and spatially average it to yield a single temperature value as a function of time (i.e., the latitude and longitude dimensions are removed by this action, as shown with the `.shape` method.)

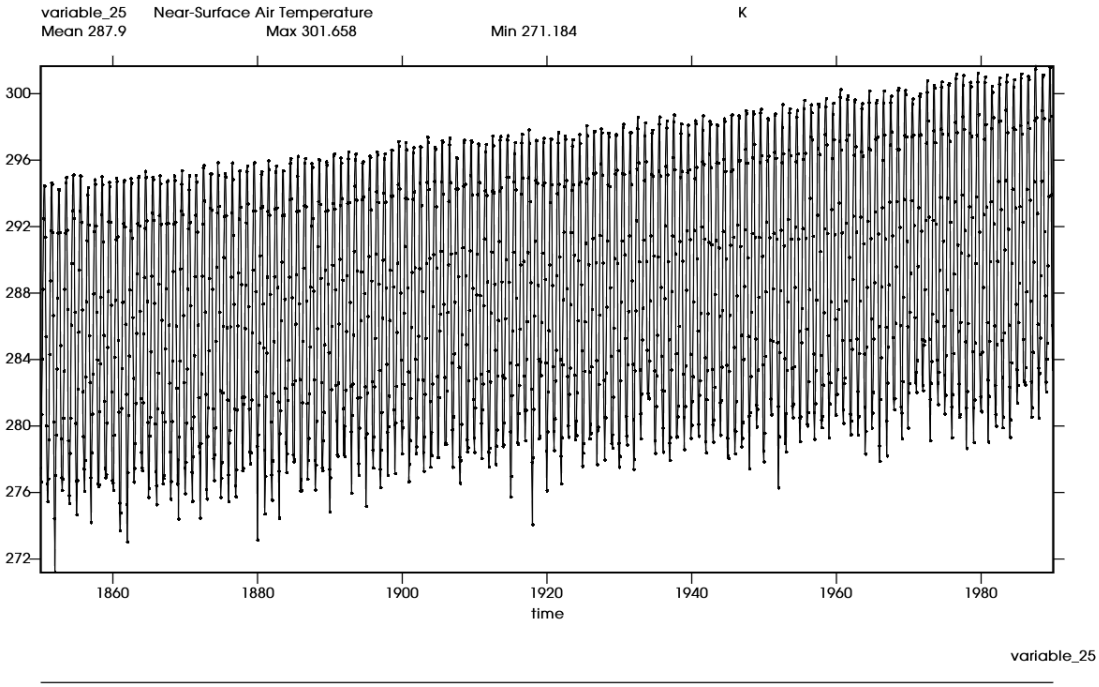
[5]:
```
data_ts = genutil.averager(data, axis='xy')
data_ts.shape
```

[5]:
```
(1680,)
```

In the next line of code, let's plot this time series.

[6]:
```
x = vcs.init(bg=True, geometry=(1200,900))
line = vcs.create1d()
line.markersize = .5
x.plot(data_ts, line)
```

[6]:



The figure above shows that the surface temperature averaged over the U.S. is characterized by a pronounced annual cycle and a long term trend with some year to year variability. The goal of the remainder of this analysis is remove both the climatological mean annual cycle and the long term trend. This procedure leads to a filtered time series representing monthly temperature anomalies characterizing variability that cannot be explained by annual cycle forcing or any long-term changes in climate forcing.

But first, let's look at a numerical example, which illustrates that the order in which you perform operations can make a difference when a dataset is incomplete (i.e., includes "masked" or "missing" data).

4 Order of Operations Matters

4.1 Numerical Example

Section **??**

If data are missing from a dataset, the order of operations can matter. The following is a numerical example of averaging values two different ways.

Let's say we have the following dataset, which is a function of X and Y:

	Y1	Y2	Y3	Y4
X1	*3*	*4*	-	*7*
X2	-	*5*	-	-
X3	*1*	*2*	*5*	*5*
X4	-	-	*6*	*4*

Creating this dataset using a numpy array we have:

[7]:
```
import numpy
array = numpy.array([[3,4,999,7], [999,5,999,999],[1,2,5,5],[999,999,6,4.]])
array
```

[7]:
```
array([[  3.,   4., 999.,   7.],
       [999.,   5., 999., 999.],
       [  1.,   2.,   5.,   5.],
       [999., 999.,   6.,   4.]])
```

Masking the 999 values leads to the following:

[8]:
```
masked = numpy.ma.masked_equal(array, 999.)
masked
```

[8]:
```
masked_array(
  data=[[3.0, 4.0, --, 7.0],
        [--, 5.0, --, --],
        [1.0, 2.0, 5.0, 5.0],
        [--, --, 6.0, 4.0]],
  mask=[[False, False,  True, False],
        [ True, False,  True,  True],
        [False, False, False, False],
        [ True,  True, False, False]],
  fill_value=999.0)
```

If we average over X first, then average over Y, we get (4.667 + 5.000 + 3.250 + 5.000) / 4 = 4.479

	Y1	Y2	Y3	Y4	Average
X1	*3*	*4*	-	*7*	**4.667**
X2	-	*5*	-	-	**5.000**
X3	*1*	*2*	*5*	*5*	**3.250**
X4	-	-	*6*	*4*	**5.000**
Average					**4.479**

Verifying this with code gives:

[9]:
```
print("X, Y:", numpy.ma.average(numpy.ma.average(masked, axis=-1)))
```

```
X, Y: 4.479166666666667
```

If we average over Y first, then over X, we get (2.000 + 3.667 + 5.500 + 5.333) / 4 = 4.125

	Y1	Y2	Y3	Y4	Average
X1	*3*	*4*	-	*7*	
X2	-	*5*	-	-	
X3	*1*	*2*	*5*	*5*	
X4	-	-	*6*	*4*	
Average	**2.000**	**3.667**	**5.500**	**5.333**	**4.125**

Verifying this with code yields:

[10]:
```
print("Y, X:", numpy.ma.average(numpy.ma.average(masked, axis=0)))
```

```
Y, X: 4.125
```

Finally, if we average using all 16 cells at once (but, of course, exclude those with missing data), we get (3 + 4 + 7 + 5 + 1 + 2 + 5 + 5 + 6 + 4) / 10 = 4.200

[11]:
```
print("All:", numpy.ma.average(masked))
```

```
All: 4.2
```

We get three different overall means (4.479, 4.125, or 4.200) depending on our processing choices (specifically the order of operations). (Note that with appropriate weighting, which is *not* done here, a consistent mean *can* be obtained, independent of the order of operations. From the first example of averaging over X, then Y, since the total number of values in the dataset is 10, the proper weighting would be: 4.667 * 3/10 + 5.000 * 1/10 + 3.250 * 4/10 + 5.000 * 2/10 = 4.200.)

When additional processing steps are involved, ordering can affect results in more complex ways, as in the next example.

4.2 Removing the Climatological Annual Cycle

Section **??**

Before detrending a time series, it is often best to filter it by removing the climatological mean annual cycle; in fact, this may be a requirement for particular types of analyses. In the case of a time series that does not span a representative number of years, an accurate determination of the trend of interest may require removal of the climatological mean annual cycle. To see why, consider a temperature time series starting in January and ending in July a year and a half later. Over Northern Hemisphere continents with a large annual cycle, the ending temperature would be much higher than the beginning temperature simply due to the usual seasonal changes in temperature. A linear fit to the time-series would result in a trend that would not reflect any real change in climate but just the particulars of the time period treated. Thus, before computing a trend it is best to remove the climatological annual cycle.

The surface temperature data considered earlier has no missing data, but more often than not observational data sets are incomplete. For purposes of illustration, we therefore will first modify the surface temperature dataset by designating certain values as "missing". Specifically, we'll treat as "missing" (i.e., delete or mask) all data values that are within 2 degrees of the maximum temperature (`data.max()-2`) and store the result in `datamskd`.

[12]:
```
datamskd = MV2.masked_greater(data, data.max()-2)
```

Karl thinks it would be good to plot here (as a function of time) the number of grid cells with missing data. Would that be difficult? This would help in interpreting the later time-series.

[13]:
```
# code for plotting number of grid cells w/ missing data as a function of time.
```

We now consider what order to carry out the operations needed to remove the climatological mean annual cycle:

1. Remove the climatological annual cycle yielding monthly-mean departures
2. Spatially average over the domain.

We examine the two ordering options, with steps performed in the order shown above and then in the reverse order.

4.2.1 Processing Option 1: Remove the annual cycle, then spatially average to create a single time series

Section **??**

First at each location (grid cell) we will remove the climatological mean annual cycle to produce monthly mean departures or anomalies relative to the climatological annual cycle. Then we will spatially average the anomalies to produce a time-series of regional-mean anomalies.

In the next line of code, the `ANNUALCYCLE.departures` calculates the average monthly temperature value for each of the 12 months in a year over the complete time period for each location in the input data file and determines the departure of each temperature (at each time and location) from this average or climatological monthly value.

For example, once an average January value for the entire dataset has been calculated, each January value in the dataset is subtracted from that average January value to yield a series of January departures for each year in the data set. Since there are 1680 months in the dataset, there are 1680/12 = 140 years of data, and therefore 140 January departures. Since there are 140 Februaries, 140 Marches, etc. there are 140 departures x 12 months = 1680 departures for each location in the dataset, as the `.shape` method shows (1680 departure values by 13 latitude bands, by 16 longitude values).

[14]:
```
datamskd_departures = cdutil.times.ANNUALCYCLE.departures(datamskd) # extract␣
↪the departures of the masked data.
```

[15]:
```
datamskd_departures.shape
```

[15]:
```
(1680, 13, 16)
```

Now that we have a time series of monthly departures at each grid cell, we can spatially average them over the entire domain to obtain a single time-series for the regional-mean monthly anomalies:

[16]:
```
datamskd_departures_ts = genutil.averager(datamskd_departures, axis='xy') #␣
↪create time series of the masked data departures.
```

Using the `.shape` method below verifies that the resulting spatially averaged data no longer have latitude and longitude information.

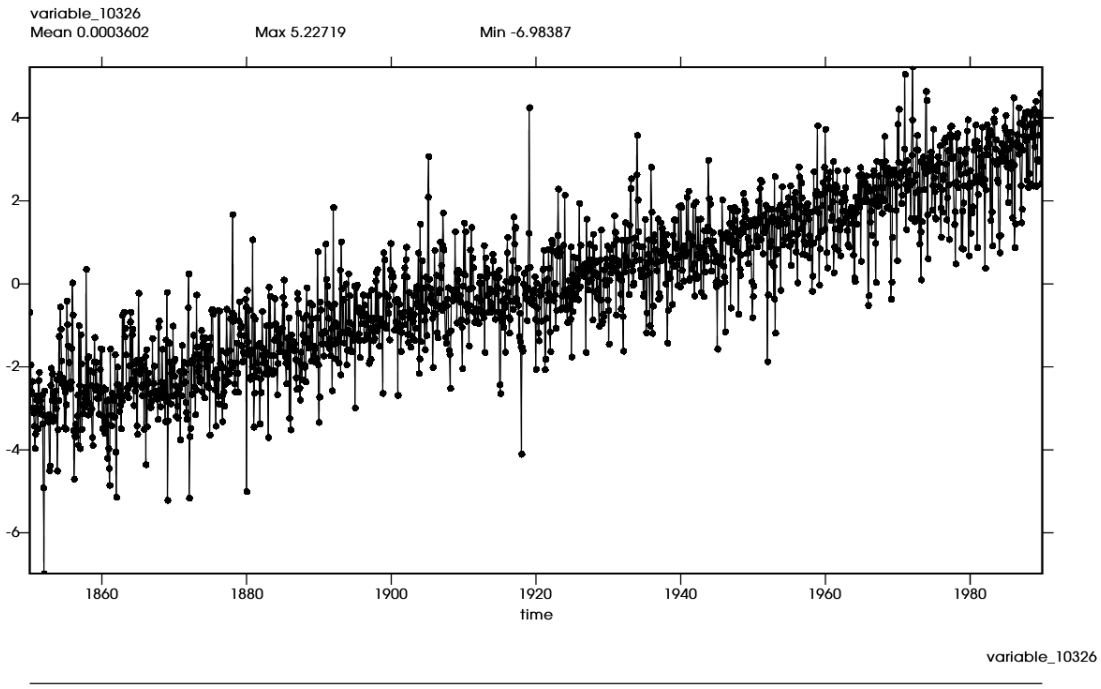
[17]:
```
datamskd_departures_ts.shape
```

[17]:
```
(1680,)
```

Let's plot the resulting time series of the departures (i.e. the result of removing the annual cycle before creating the time series).

[18]:
```
x.clear()
x.plot(datamskd_departures_ts)
```

[18]:



Notice how, with the annual cycle removed, it is easier to see the trend and which months are anomalously warm or cold (compared to the climatological mean temperature).

It should be noted that with the order of operations executed under this option, we cannot expect the mean of the anomalies for a given month of the year summed over all years to be exactly zero. Although the anomalies at the individual cells do indeed have zero means, this cannot be guaranteed when they are averaged spatially if there are missing data. This is because the order of averaging data sequentially over two dimensions when there are missing values depends on the order it is done, as demonstrated earlier. This means that under Processing Option 1, additional care must be taken. Although applying appropriate weighting during the averaging procedures (over time and over space) can remedy the problem, an easier solution is to remove a mean value (over all years for a given month of the year) from that month's temperature anomaly (`datmskd_departures_ts`) to obtain anomalies (`datmskd_departures_ts_corrected`) with means of zero:

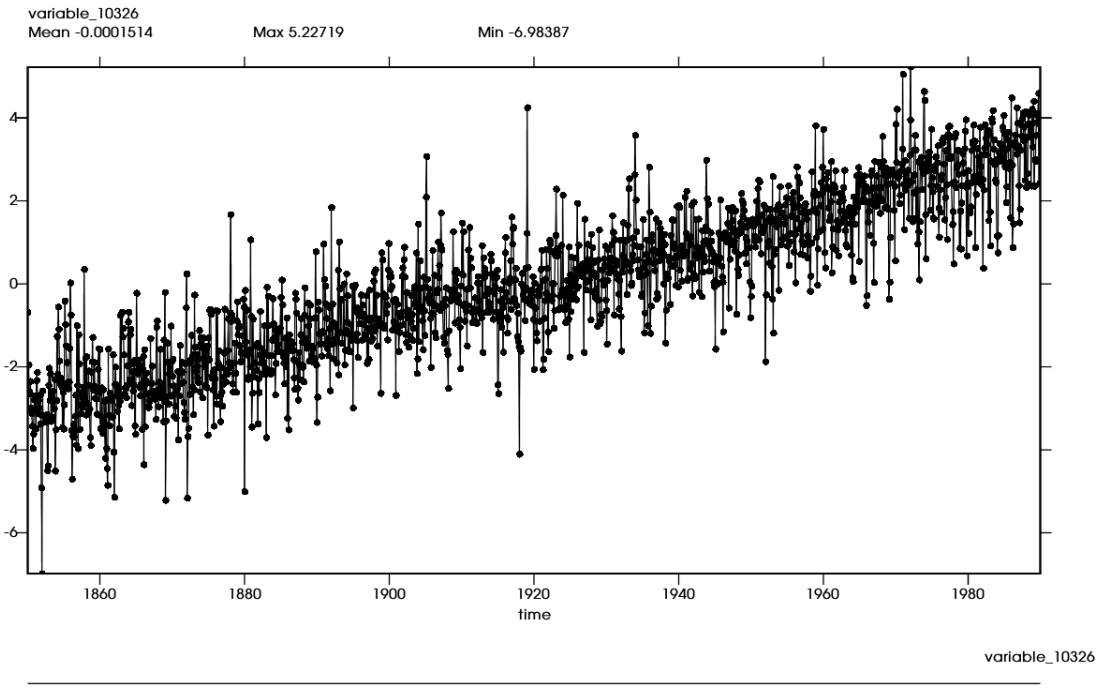
[19]:
```
datamskd_departures_ts_corrected = cdutil.times.ANNUALCYCLE.
↪departures(datamskd_departures_ts)
```

[20]:
```
datamskd_departures_ts_corrected.shape
```

[20]:
```
(1680,)
```

[21]:
```
x.clear()
x.plot(datamskd_departures_ts_corrected)
```

[21]:



4.2.2 Processing Option 2: Spatially average data to obtain a single time-series, then remove the annual cycle

Section **??**

Now let's reverse the order of performing the operations. We first calculate the spatially-averaged time series and then remove the annual cycle.

We calculate the time series characterizing area-mean temperature for the masked dataset by spatially averaging the temperature values over all latitude and longitude locations to give a single global temperature time series. (Again, the `.shape` method, shows we are looking at 1680 values with no latitude or longitude, as expected.)

[22]:
```
datamskd_ts = genutil.averager(datamskd, axis='xy')
```

[23]:
```
datamskd_ts.shape
```

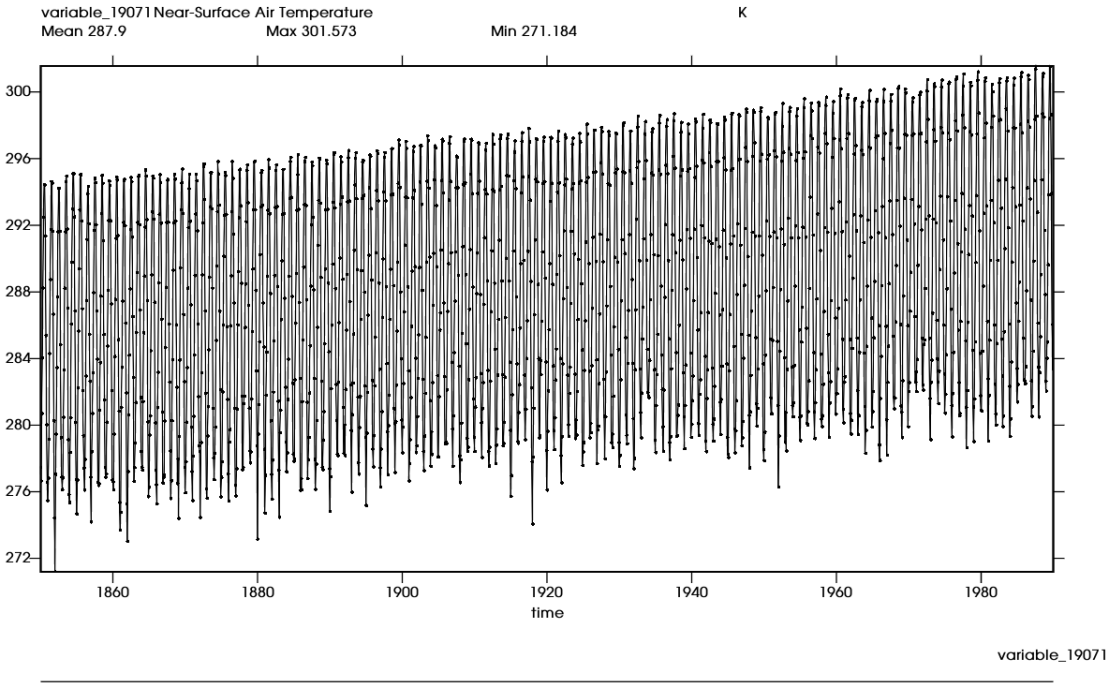
[23]:
```
(1680,)
```

Let's take a look at this time series. Note that the trend and the annual cycle are similar but not identical to what we saw with the unmasked data. In particular the maximum value of the

spatial mean (considering all times) is lower than before (301.573 now compared to 301.658 for the unmasked data). This is because temperatures at individual grid cells that are within 2 K of the maximum temperature have been eliminated from consideration ("masked").

[24]:
```
x = vcs.init(bg=True, geometry=(1200,900))
line = vcs.create1d()
line.markersize = .5
x.plot(datamskd_ts, line)
```

[24]:



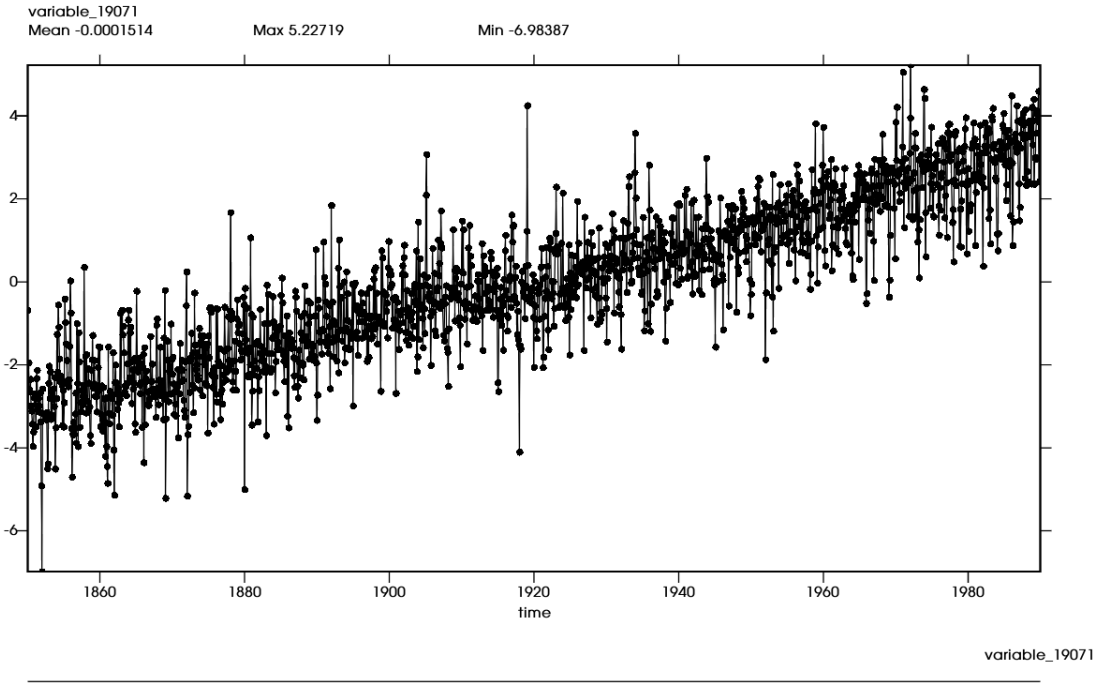
Next we'll remove the annual cycle using the `ANNUALCYCLE.departures` method as we did in executing Option 1 above. Again, the method calculates an average temperature value for each month of the year and determines the departure of the temperature at each time value from the average monthly temperature, effectively removing the annual cycle from the data.

[25]:
```
datamskd_ts_departures = cdutil.times.ANNUALCYCLE.departures(datamskd_ts)
datamskd_ts_departures.shape
```

[25]:
```
(1680,)
```

[26]:
```
x.clear()
x.plot(datamskd_ts_departures)
```

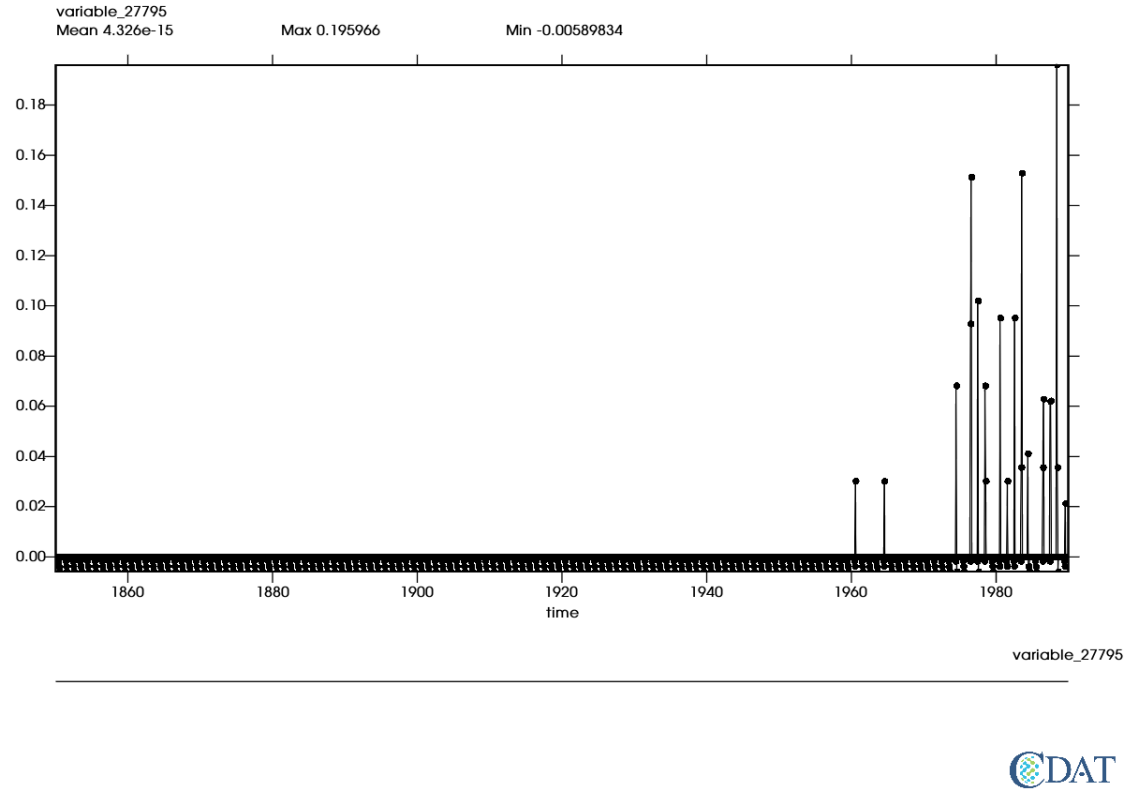
[26]:



It is difficult to discern a difference between the time series produced by the two different procedures, so let's plot their difference: `datamskd_departures_ts_corrected` (remove annual cycle at each grid cell, then spatially average to create a single time series) minus `datamskd_ts_departures` (spatially average to create a single time series, then remove annual cycle). This is the difference between Processing Option 1 and Option 2.

[27]:
```
x.clear()
x.plot(datamskd_departures_ts_corrected - datamskd_ts_departures)
```

[27]:



Note that the order of operations matters because there are missing data. The two time series are slightly different in some of the warmer (i.e., later) years, where data values may be "missing" because they exceed the temperature threshold (2 degrees cooler than the maximum temperature). If the order of operations did not matter (as in datasets without missing data), the two time series would be identical.

For the case considered above, computing the area mean time-series before removing the climatological annual cycle ("Processing Option 2") can be misleading. Recall that the missing data occur in the warmest part of the domain during the warmest part of the year. When these values are "missing", we reduce the regional mean (for months with missing data), leading to an artificially cool month for that time of year (i.e., the regional-mean anomaly can become negative solely because the warm temperatures are missing over some of the region during that month). Note that if the values were "missing" for a given grid cell every year at the same time of year, then they would not affect the regional mean and would not lead to unrealistic anomalies, but the values are only missing in the later part of the period considered.

In contrast, "Processing Option 1" first removes the climatological annual cycle at each grid cell. Now the departures reflect true anomalies from the normal monthly temperature everywhere, and when they are spatially averaged, the result usually better represents the true regional mean anomalies.

5 Detrend Data

Section **??**

5.1 Consider Two Options

We apply standard linear regression formulas to compute the slope (a) and intercept (b) of the trend line: T = a*t + b where T = temperature departures (from the climatological annual cycle), and t = time.

Starting with the modified temperature dataset that includes "missing" values, we shall again consider how the order of operations affects the result of removing the trend. **Under both options below, the first step is to remove the climatological mean annual cycle from the time series of each grid cell (as described above, `datamskd_departures`).** Then we must decide which of the following processing procedures is most appropriate for a particular application:

- Processing Option A: Spatially average the fields over the domain and remove the trend from the resulting time-series.
- Processing Option B: Remove the trend at each grid cell and then spatially average the results.

Under Option B, the resulting time-series may include a residual trend (and a residual non-zero mean), so an additional step can be applied, as discussed below, to obtain a true anomaly field that has been fully detrended.

5.1.1 Processing Option A: Spatially average the fields over the domain, then remove the trend from the resulting time-series.

The first step under this option is to spatially average the departures (from which the climatological mean annual cycle has already been removed – `datamskd_departures`). This was already done under Processing Option 1 in the above section, so we can use the result stored in `datamskd_departures_ts`, which is a single time-series of regionally-averaged anomalies (with the annual cycle removed).

Now to detrend this time series we calculate the regression coefficients (slope and intercept) and subtract that linear trend from `datamskd_departures_ts`. First we calculate the slope and intercept:

Karl: Shouldn't we be using `datamskd_departures_ts_corrected`? I added some lines at the bottom of the notebook to calculate an "Option A Corrected" which uses `datamskd_departures_ts_corrected`.

[28]:
```
slope_ts, intercept_ts = genutil.statistics.
↪linearregression(datamskd_departures_ts, axis="t")
```

Now we can remove the trend from the `datamskd_departures_ts` by subtracting the trend line after extracting the vector of times constituting the time axis for the data.

First extract the times:

[29]:
```
times = MV2.array(datamskd.getTime()[:])
times.setAxis(0, datamskd.getTime())
times.shape
```

[29]:
```
(1680,)
```

Now remove the trend:

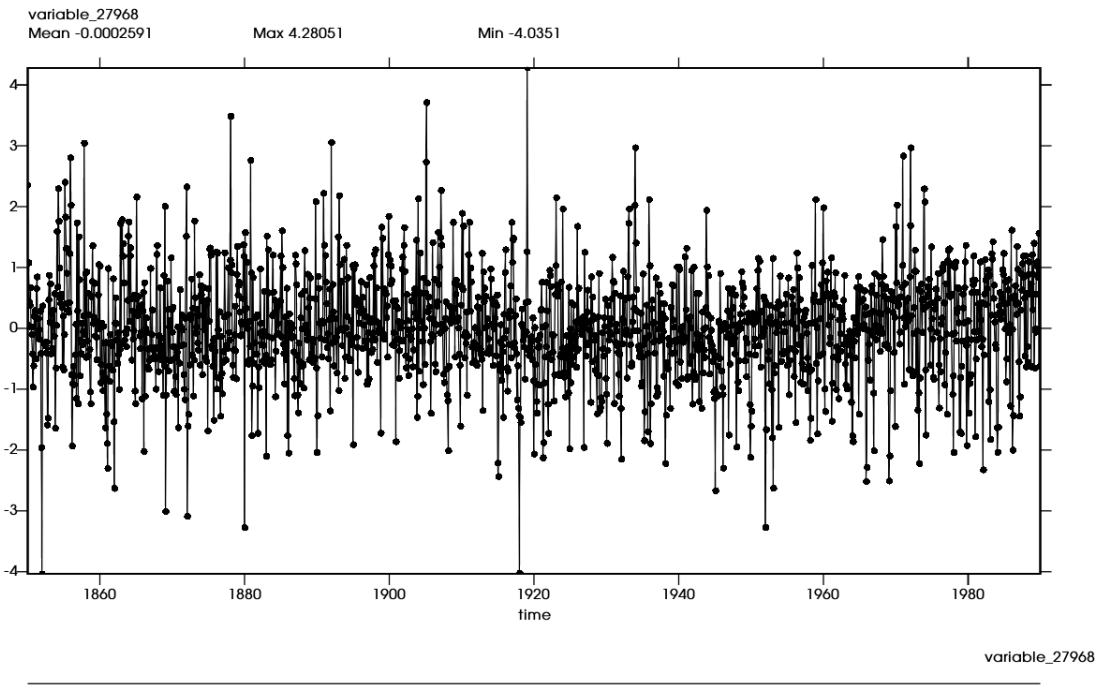
[30]:
```
datamskd_departures_ts_detrend = datamskd_departures_ts - times * slope_ts -␣
↪intercept_ts
datamskd_departures_ts_detrend.shape
```

[30]:
```
(1680,)
```

Plotting the resulting time series yields:

[31]:
```
x.clear()
x.plot(datamskd_departures_ts_detrend)
```

[31]:



5.1.2 Processing Option B: Remove the trend at each grid cell, then spatially average the results.

Section **??**

We now reverse the order of operations performed under Option A. First, for each grid cell we compute the regression coefficients (slopes and intercepts). We then remove the trends at each cell before computing the regional mean anomaly time series.

We calculate the regression coefficients for each grid cell:

[32]:
```
slope, intercept = genutil.statistics.linearregression(datamskd_departures,␣
↪axis="t")
```

```
print("Shapes: slope {}, intercept {}".format(slope.shape, intercept.shape))
```

```
Shapes: slope (13, 16), intercept (13, 16)
```

Next, for each grid cell we want to subtract the regression line from the departure time series on which it is based (`datamskd_departures`) to obtain a detrended time series of anomalies. In general, however, we cannot simply subtract `slope*times + intercept` because `slope` is a function of latitude and longitude, whereas `times` is a function of time. We can only do element-wise array multiplication if the two arrays are the same shape. Fortunately, genutil has a "`grower`" function that can replicate elements of an array to fill the dimensions that are missing. We will need to first "grow" the one-dimensional `times` array, replicating the times across the longitude and latitude dimensions:

[33]:
```
tmp, full_times = genutil.grower(datamskd_departures, times)
print("Shapes: tmp {}, full_times {}".format(tmp.shape, full_times.shape))
```

```
Shapes: tmp (1680, 13, 16), full_times (1680, 13, 16)
```

NEED TO REWRITE THE FOLLOWING TO EXPLAIN WHAT THE GROWER HAS DONE - or is Karl's explanation above the line of code sufficient? Since time is not necessarily ***on index 0***, we need to use the grower function to expand the `times` variable to cover the entire area of the dataset which will add the latitude and longitude dimensions back in. The `.grower` method takes the values in the second argument (the 1-dimensional `times` variable here) and copies (or grows) the values to match the dimensions of the first argument (the 3-dimensional `datamskd_departures` variable here) to yield variables (`tmp` and `full_times`) with the same dimensions as the first argument).

We use data as the first argument to ensure the same order.

QUESTIONS FOR CHARLES: 1. WHAT DOES "on index 0" mean? IS IT "Since time is not necessarily the 0th dimension"? AND IF SO, HOW IS THAT POSSIBLE? OR DO YOU MEAN "Since `times` is not a 3-dimensional variable, but it needs to be for future calculations, we need to use the grower function…"?

2. WHEN YOU SAY "We use data as the first argument to ensure the same order." DO YOU MEAN THE SAME ORDER OF THE DIMENSIONS (13, 16) (BUT HOW COULD THE DIMENSIONS BE IN A DIFFERENT ORDER LIKE (16, 13)? OR DO YOU MEAN SOMETHING LIKE "We use `datamskd_departures` as the first argument to ensure the resulting data variable has the same dimensions as `datamskd_departures`."

3. WHY ARE YOU USING `datamskd_departures` TO CREATE full_times, BUT `datamskd` TO GROW `coeff` AND `intercept`? COULD WE USE `datamskd_departures` FOR ALL THREE GROWER COMMANDS - THAT 1ST ARGUMENT IS JUST TO SUPPLY THE DIMENSIONS, CORRECT? NO ACTUAL VALUES FROM THE 1ST ARGUMENT ARE USED IN THE .grower METHOD, ARE THEY?

4. WHAT IS THE `tmp` VARIABLE USED FOR? IT DOESN'T SEEM TO BE USED ANYWHERE AFTER IT IS CREATED.

Now we apply the same `grower` function to the two-dimensional `slope` and `intercept` arrays to replicate across the time dimension.

[34]:
```
tmp, slope_full = genutil.grower(datamskd, slope)
print("Shapes: tmp {}, slope_full {}".format(tmp.shape, slope_full.shape))
tmp, intercept_full = genutil.grower(datamskd, intercept)
print("Shapes: tmp {}, intercept_full {}".format(tmp.shape, intercept_full.
 ↪shape))
```

```
Shapes: tmp (1680, 13, 16), slope_full (1680, 13, 16)
Shapes: tmp (1680, 13, 16), intercept_full (1680, 13, 16)
```

Now we can remove the linear trend (since all arrays share the same three dimensions) to obtain an anomaly time series at each grid cell that has no trend.

[35]:
```
datamskd_departures_detrend = datamskd_departures - full_times * slope_full -␣
 ↪intercept_full
datamskd_departures_detrend.shape
```

[35]:
```
(1680, 13, 16)
```

The `averager` method can now be applied to obtain a spatial mean anomaly time series, completing the steps called for in Processing Option B:

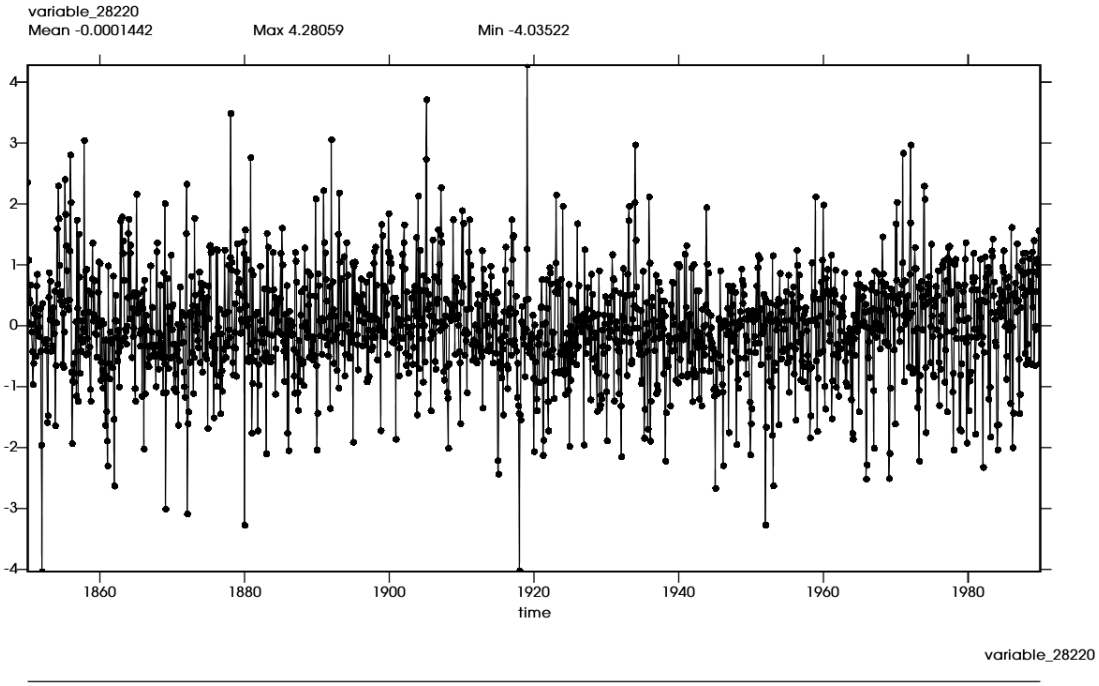
[36]:
```
datamskd_departures_detrend_ts = genutil.averager(datamskd_departures_detrend,␣
 ↪axis='xy')
datamskd_departures_detrend_ts.shape
```

[36]:
```
(1680,)
```

Plotting this, Option B, time series yields:

[37]:
```
x.clear()
x.plot(datamskd_departures_detrend_ts)
```

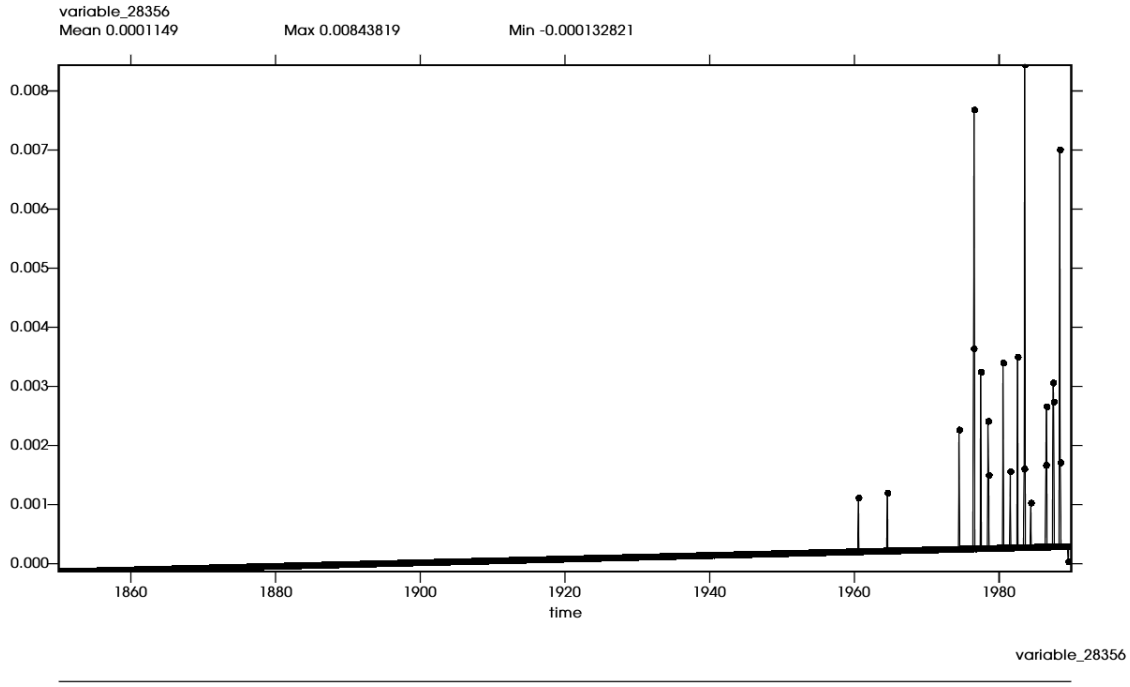
[37]:



Now we can show that the order of operations again matters by plotting the difference between the time series produced by processing options B and A:

[38]:
```
x.clear()
x.plot(datamskd_departures_detrend_ts - datamskd_departures_ts_detrend)
```

[38]:



(To recap: `datamskd_departures_detrend_ts` (Option B) was created by removing the annual cycle from the masked data, then removing the trend (at each grid cell), then creating a single time series by averaging over the whole spatial area of the dataset, and `datamskd_departures_ts_detrend` (Option A) was created by removing the annual cycle from the masked data, then creating a spatially averaged time series, before removing the warming trend.)

It is evident from the above plot that the difference between the two time series includes a residual linear trend and a non-zero mean. This again is caused by missing data. Under Option B, the time series at individual cells should not be simply area-weighted to produce an area-mean because they have differing amounts of missing data. When combining the grid cell values to produce an area mean, the missing values in the individual time series lead to a time-series with a slight trend and non-zero mean. As in the removal of the climatological mean annual cycle, the best way to remedy this would be to appropriately weight values contributing to the area mean, but a simple alternative is to modify the final time series by removing the residual trend and mean. We'll accomplish this by executing the following steps: 1. Compute regression coefficients (slope and intercept) of `datamskd_departures_detrend_ts` 2. Subtract regression line from `datamskd_departures_detrend_ts`. 3. Subtract `datamskd_departures_ts_detrend` from resulting time series to show difference between Options B & A, after this correction.

Step 1. Computing the regression coefficients (slope and intercept) of `datamskd_departures_detrend_ts`:

[39]:
```
slope_detrend_ts, intercept_detrend_ts = genutil.statistics.
 ↪linearregression(datamskd_departures_detrend_ts, axis="t")
print("Shapes: slope_detrend_ts {}, intercept_detrend_ts {}".
 ↪format(slope_detrend_ts.shape, intercept_detrend_ts.shape))
```

```
Shapes: slope_detrend_ts (), intercept_detrend_ts ()
```

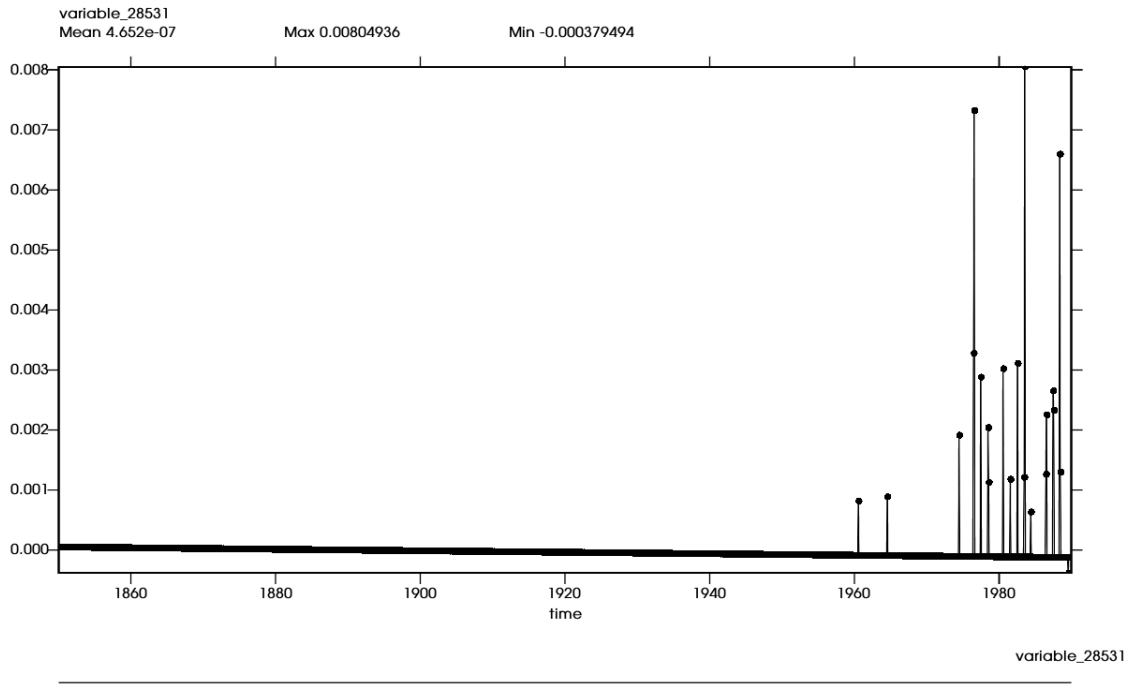
Step 2. Subtracting the regression line from `datamskd_departures_detrend_ts` and storing the result in `datamskd_departures_detrend_ts_corrected`:

[40]:
```
datamskd_departures_detrend_ts_corrected = datamskd_departures_detrend_ts -␣
 ↪times * slope_detrend_ts - intercept_detrend_ts
```

Step 3. Subtracting `datamskd_departures_ts_detrend` from the resulting time series, `datamskd_departures_detrend_ts_corrected`, to show difference between Options B & A, after this correction:

[41]:
```
x.clear()
x.plot(datamskd_departures_detrend_ts_corrected -␣
 ↪datamskd_departures_ts_detrend)
```

[41]:



Question for Karl: Which option is the best, Option A or Option B (or Option A Corrected - see below), or does it depend on what question you want to answer?

5.1.3 Option A Corrected

Section **??**

[42]:
```
slope_ts_corrected, intercept_ts_corrected = genutil.statistics.
↪linearregression(datamskd_departures_ts_corrected, axis="t")
```

Now we can remove the trend from the `datamskd_departures_ts_corrected` by subtracting the trend line. Since we've already extracted the vector of times constituting the time axis for the data and stored it in `times`, we can just reuse `times` here.

Now remove the trend:

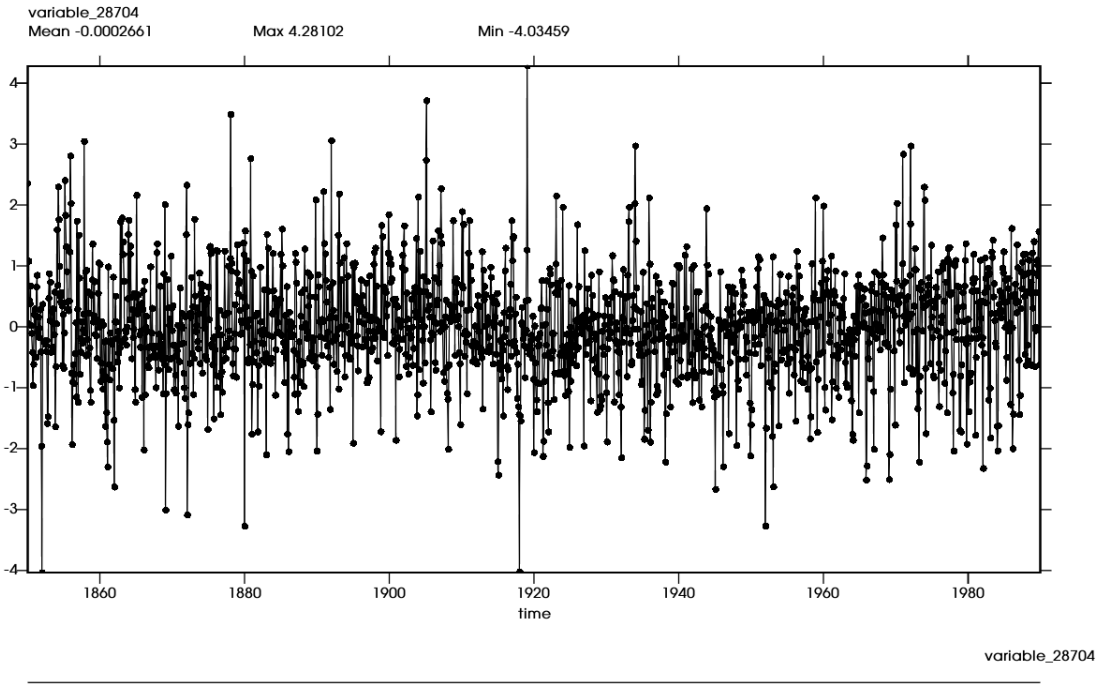
[43]:
```
datamskd_departures_ts_corrected_detrend = datamskd_departures_ts_corrected -␣
↪times * slope_ts_corrected - intercept_ts_corrected
datamskd_departures_ts_corrected_detrend.shape
```

[43]:
```
(1680,)
```

Plotting the resulting time series yields:

[44]:
```
x.clear()
x.plot(datamskd_departures_ts_corrected_detrend)
```

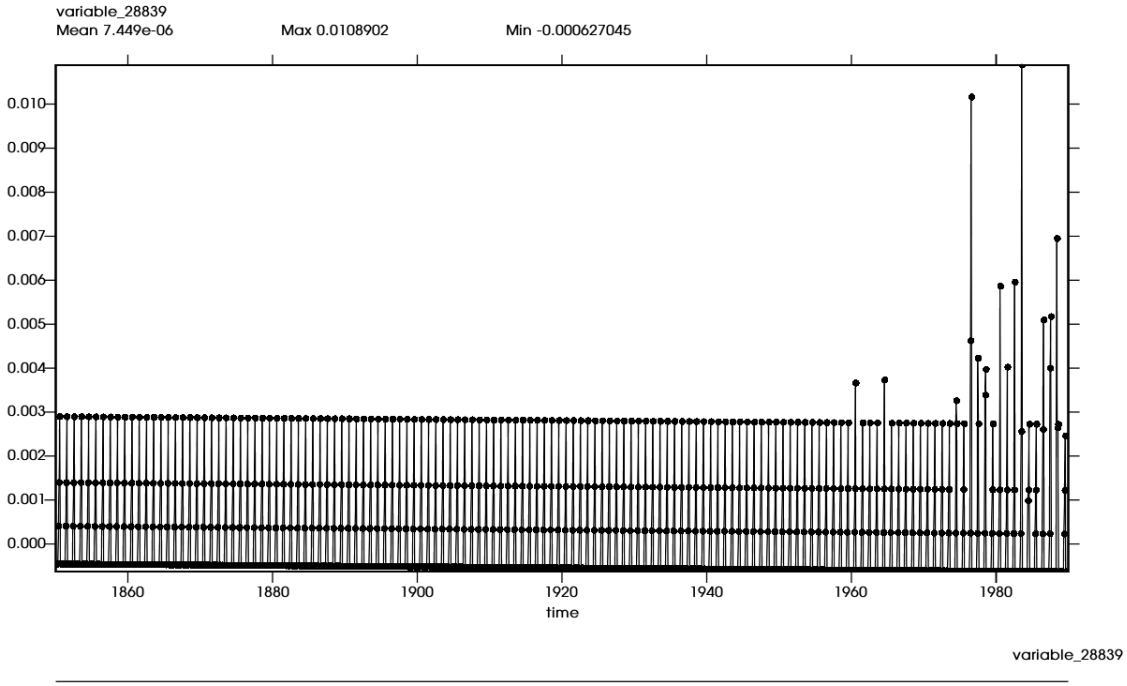
[44]:



Finally, subtracting Option A Corrected from Option B Corrected yields:

[45]:
```
x.clear()
x.plot(datamskd_departures_detrend_ts_corrected -␣
↪datamskd_departures_ts_corrected_detrend)
```

[45]:



Karl and Charles: The following questions were lost in the edits. Perhaps are just a distraction and they don't need to be included, but I thought I'd put them here in case we should add some content back in.

The index of time `t` can be expressed in any units (days, months, years, etc.) and `genutil` will determine its index. **QUESTION FOR CHARLES: WHAT IS MEANT BY INDEX? IS IT THE LOCATION OF THE DIMENSION AS IN "TIME IS DIMENSION 0"?**

After computation we lose the time axis.

The units for the coefficent are `K/day` since the time axis units are in `days since XXX`. **ASK KARL - DATA IS MONTHLY DATA, NOT DAILY SO SHOULDN'T THE UNITS BE K/MONTH?**

The units for the intercept are `K` since y is temperature.

Section **??**

The CDAT software was developed by LLNL. This tutorial was written by Charles Doutriaux, Holly Davis, and Karl Taylor. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

If you have questions about this notebook, please email our CDAT Support address, cdat-support@llnl.gov.