

```
In [1]: 1 # This is always required for inline plot rendering in IPython Notebooks; might
2 # as well do it first, even before the markdown sections, just to be safe
3 #%matplotlib inline
4 %matplotlib notebook
```

## Adjusted Data Algorithm - Contents:

- [Theoretical Basis](#)
  - [Motivation](#)
  - [Traditional Baseline Adjustments](#)
  - [Affine Transformations](#)
  - [Estimating Affine Transform Matrix](#)
    - [Ordinary Least Squares](#)
    - [Singular Value Decomposition](#)
  - [Adaptive Coefficients](#)
    - [Weighted Least Squares](#)
    - [Linear Interpolation Between Affine Transforms](#)
  - [Anomaly Detection](#)
- [Notebook Functions](#)
  - [Imports](#)
  - [Baseline and Absolute Data Retrieval](#)
  - [Observation Time Weighting Functions](#)
  - [Vector Distance Calculator](#)
  - [Statistical Time Series Filters](#)
  - [Affine Transform Matrix Generators](#)
  - [Affine Transform Matrix Interpolators](#)
  - [Do-It-All Demonstration and Validation Wrapper](#)
- [Demonstration and Validation](#)
  - [Synthetic Data Demonstration](#)
    - [Construct synthetic time series](#)
    - [Estimate Affine Transformation Matrix](#)
  - [Comparison with \(Quasi-\)Definitive Data](#)
    - [Boulder \(BOU\) Observatory](#)
    - [Barrow \(BRW\) Observatory](#)
    - [Stennis \(BSL\) Observatory](#)
    - [College \(CMO\) Observatory](#)
    - [Deadhorse \(DED\) Observatory](#)
    - [Fredericksburgh \(FRD\) Observatory](#)
    - [Fresno \(FRN\) Observatory](#)
    - [Guam \(GUA\) Observatory](#)
    - [Honolulu \(HON\) Observatory](#)
    - [Newport \(NEW\) Observatory](#)
    - [San Juan \(SJG\) Observatory](#)
    - [Shumagin \(SHU\) Observatory](#)
    - [Sitka \(SIT\) Observatory](#)
    - [Tucson \(TUC\) Observatory](#)

## Theoretical Basis

### Motivation

The USGS employs 3-axis vector magnetometers, typically using fluxgate technology in the modern era. Alone, these magnetometers are high-quality variometers. That is, they record the relative variation of Earth's magnetic field over time very accurately, but are notoriously difficult to calibrate in an absolute sense. It is only when variometers are combined with frequent absolute calibration measurements that we get what is generally described as a **geomagnetic observatory**.

Historically, the merging of these absolute calibrations and magnetic variation measurements has been a laborious process, ultimately resulting in the magnetic observatory community's standard "definitive" data product for each observatory. Definitive data is processed in one year blocks, starting no less than one year after the first observation made that year, and so not nearly a real time product. More recently, some of the most stringent requirements for definitive data were relaxed so that a new community standard, "quasi-definitive" data, could be produced in a more timely (~1 month delay), but this was still not real time enough for many modern technological applications.

There is a growing and still largely unmet demand for calibrated near real time data. Following INTERMAGNET terminology, we will refer to this as Adjusted Data, although it is also often referred to as "provisional" in the magnetic observatory community. The Adjusted Data standard is not strictly defined, but the goal stated in the newest version of INTERMAGNET's Technical Reference Manual is to match statistical specifications of quasi-definitive data, while acknowledging and accepting the fact that real time data is likely to be more noisy, and/or have more gaps.

There was a previous attempt to produce Adjusted Data at the USGS that went largely undocumented. Given subsequent staff turnover and decommissioning of legacy computer systems, it is now impossible to assess the quality of this prototype data product, or fully understand why it was never deployed operationally. We speculate, with some anecdotal evidence, that the algorithm for generating these data was only a minor adaptation of the definitive processing software, which we already know is not particularly suited to real time processing.

This report is an attempt to: 1) distill and document institutional knowledge associated with legacy procedures and software for generating (quasi-)definitive data, partly to 2) assess how or why these legacy procedures and software may have been inadequate for near real time adjusted data generation, and 3) describe, demonstrate, and validate a new methodology that is specifically tailored for near real time processing.

## Traditional Baseline Adjustments

Fluxgate sensors have a limited range over which their response functions can be considered linear. In order to maximize their sensitivity while still measuring the extremely wide range of magnetic environments encountered across Earth's surface, it is common practice to generate a bias magnetic field that opposes and (mostly) nullifies the Earth's main magnetic field along each of the instrument's axes. In doing so, the fluxgate can precisely capture minute relative variations in the magnetic field relative to the main field. One need only add the static oppositional field back to these variational measurements for an accurate and precise measure of the time-varying geomagnetic field.

While this oppositional magnetic field strength might be estimated from first principals, significant static and time-dependent uncertainties in the fluxgate's mechanical geometries, plus electronic/electrical inefficiencies, conspire to require frequent and regular calibration to meet tolerances expected of magnetic observatories. So frequent, in fact, that removing the sensors to a lab to do so is not generally advisable, so the sensors are effectively calibrated in-place using absolute measurements, along with various physical and geometric assumptions, to calculate so-called baseline adjustments.

Absolute measurements should not be made too closely to the fluxgate sensor, or there is risk of contaminating the data. Therefore, part of the baseline adjustment implicitly includes the quasi time-stationary vector difference between the fluxgate and absolute pier different locations. While this value could, in principal, be obtained through careful simultaneous absolute measurements at both locations, it is not really necessary unless one is genuinely interested in isolating the fluxgate's true response functions.

Also, alignment of the fluxgate and absolute measurement coordinate frames is never perfect. But, if they are close, misalignment can be corrected using simple baseline adjustments and implicit small angle assumptions. When coordinate frames are not well aligned an initial rotation may be applied to obtain nominal alignment, and allow simple baseline corrections after that.

The USGS, as well as many international magnetic observatories, install their fluxgate sensors so that the primary horizontal axis,  $H$ , aligns with the local magnetic meridian. This means that the secondary horizontal axis,  $E$  (for eastward), need not nullify the Earth's main field in this direction since, on average, it will be zero. Also, it is assumed that the absolute-fluxgate pier difference is zero in the  $E$  direction. There is undoubtedly additional uncertainty in the  $E$  direction, but in practice, it is ignored in favor of rotating the  $H$  axis by a declination baseline angle:

$$D_{base} = D_{abs} - \arcsin(E/H_{abs})$$

With the absolute and fluxgate coordinate axes now roughly aligned, the  $H$  baseline correction is estimated by projecting the absolute horizontal intensity onto the fluxgate's  $H$  axis, and subtracting the fluxgate-measured magnetic field:

$$H_{base} = \sqrt{H_{abs}^2 - E^2} - H$$

This is illustrated in the following figure, where the blue vector represents the absolute total horizontal magnetic vector in geographic coordinates, the red vector represents the horizontal vector measured by the fluxgate in its own Cartesian coordinates, the green items comprise baseline corrections, and the brown angle  $D$  is the declination relative to the fluxgate's  $H$  axis:



$$\begin{aligned}
 [\mathbf{M}] &= \begin{bmatrix} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot (\text{i.e., } \mathbf{T}) \\
 &\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta_x & -\sin \theta_x & 0 \\ 0 & \sin \theta_x & \cos \theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \theta_y & 0 & \sin \theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta_y & 0 & \cos \theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \theta_z & -\sin \theta_z & 0 & 0 \\ \sin \theta_z & \cos \theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot (\text{i.e., } \mathbf{R}) \\
 &\begin{bmatrix} s_1 & 0 & 0 & 0 \\ 0 & s_2 & 0 & 0 \\ 0 & 0 & s_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot (\text{i.e., } \mathbf{S}) \\
 &\begin{bmatrix} 1 & h_{xy} & h_{xz} & 0 \\ 0 & 1 & h_{yz} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot (\text{i.e., } \mathbf{H})
 \end{aligned}$$

Note that order of operation is critical, but there is no single standard for composing a full affine transformation from its constituent matrices. The preceding example follows a norm used in the [Python Transforms3D](https://pypi.org/project/transforms3d/) (<https://pypi.org/project/transforms3d/>) package, that might better be represented algorithmically as  $\text{dot}(\mathbf{T}, \text{dot}(\mathbf{R}, \text{dot}(\mathbf{S}, \mathbf{H})))$ , where "dot" is a matrix dot product function. In words, a shear correction ( $\mathbf{H}$ ) is first applied to the original  $HEZ$  vector to orthogonalize it. This is followed by rescaling the orthogonalized  $HEZ$  vectors with  $\mathbf{S}$ , which is followed by a rigid rotation  $\mathbf{R}$  to align all axes with the desired  $XYZ$  coordinate frame. Finally, the origin of the now rotated, scaled, and orthogonal vector is translated to make the coordinate frame origins coincide using  $\mathbf{T}$ .

Looking more closely at  $\mathbf{R} = \text{dot}(\mathbf{R}_x, \text{dot}(\mathbf{R}_y, \mathbf{R}_z))$ , it should be even more evident that order of operation is important. The rotation angles ( $\theta_x$ ,  $\theta_y$ , and  $\theta_z$ ) that combine to form  $\mathbf{R}$  are presented here in a manner that resembles typical yaw-pitch-roll rotations. This is 1 of 6 possible so-called Tait-Bryan rotation sequences (re-orientation by rotating around each unique axis once). Proper Euler angles have 6 additional possible sequences that involve a rotation about one axis, then a second, then again about the first. If that were not enough, both the Tait-Bryan and proper Euler angles can be with respect to fixed axes (extrinsic; as above), or with respect to the new axes after each rotation (intrinsic). This leads to 24 possible valid rotations! It is highly recommended that one convention is chosen and used consistently.

The astute reader may wonder at the particular form of  $\mathbf{H}$ . It is certainly possible to allow shear coefficients in all the off-diagonal elements of the upper-left 3x3 matrix, but we choose not to here for at least two reasons:

1. shear, by definition, must not alter the volume of the original data points, or in mathematical terms, its determinant is always 1, which is guaranteed by this triangular form; and
2. this triangular form permits an efficient mechanism for separating rigid rotation from pure shear (see "Decomposing a matrix into simple transformations" by Spencer W. Thomas, pp 320-323 in *Graphics Gems II*, James Arvo (editor), Academic Press, 1991, ISBN: 0120644819).

A nuance related to item 2 is that the solution is only unique in so far as this structure is chosen for the shear. If a different structure is chosen, the rotation matrix will be different, even though the composition of shear and rotation will always be the same. In more practical terms, this means that it is not correct to speak of "rotation" and "shear" separately, and certainly not appropriate to drop one without careful consideration. In particular, the structure chosen here (i.e., upper-triangular) means that the vector is rotated so that the  $Z$  axis of both coordinate frames are aligned perfectly. Then the  $E$  axis is sheared in the  $Z$  direction, and the  $H$  axis is sheared in both  $Z$  and  $E$  directions.

## Estimating Affine Transform Matrix

While the constituent affine matrices can, in theory, be created individually, through careful sensor design and construction, laboratory calibration, and/or surveying of the observatory site, in practice these are not always possible or adequate. Ultimately, if absolute  $XYZ$  measurements are considered the "truth" to which variational measurements are to be corrected, the linear structure of the affine transformation allows us to invoke linear estimation theory to determine an optimal transform matrix  $\mathbf{M}$  that encapsulates all the necessary corrections to the original  $HEZ$  vector data.

$$\begin{bmatrix} X_1 & X_2 & \dots & X_n \\ Y_1 & Y_2 & \dots & Y_n \\ Z_1 & Z_2 & \dots & Z_n \\ 1 & 1 & \dots & 1 \end{bmatrix} = \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{bmatrix} \cdot \begin{bmatrix} H_1 & H_2 & \dots & H_n \\ E_1 & E_2 & \dots & E_n \\ Z_1 & Z_2 & \dots & Z_n \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

## Ordinary Least Squares

The most straight-forward estimation technique is least-squares. Many mathematical software libraries have efficient routines that, given measurement arrays constructed similarly to above, will use linear least squares to determine an optimal 4x4  $\mathbf{M}$  matrix to map  $HEZ1$  into  $XYZ1$  vectors. However, what these all inevitably do, at least under the hood, is rearrange the previous matrix equation algebraically such that:

$$\begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \\ X_2 \\ Y_2 \\ Z_2 \\ 1 \\ \vdots \\ X_n \\ Y_n \\ Z_n \\ 1 \end{bmatrix} = \begin{bmatrix} H_1 & E_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & H_1 & E_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & H_1 & E_1 & Z_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & H_1 & E_1 & Z_1 \\ H_2 & E_2 & Z_2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & H_2 & E_2 & Z_2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & H_2 & E_2 & Z_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & H_2 & E_2 & Z_2 \\ \vdots & \vdots \\ H_n & E_n & Z_n & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & H_n & E_n & Z_n & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & H_n & E_n & Z_n & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & H_n & E_n & Z_n \end{bmatrix} \cdot \begin{bmatrix} M_{11} \\ M_{12} \\ M_{13} \\ M_{14} \\ M_{21} \\ M_{22} \\ M_{23} \\ M_{24} \\ M_{31} \\ M_{32} \\ M_{33} \\ M_{34} \\ M_{41} \\ M_{42} \\ M_{43} \\ M_{44} \end{bmatrix}$$

This system of equations is equivalent to the previous matrix equation with a 4x4 **M**, but perhaps makes it more clear that we are solving a set of linear equations for 16 unknowns. Given that a single vector measurement comprises only three actual data points, plus the augmented "1", it should also be clear that no fewer than 4 sets of absolute and fluxgate measurements are required to obtain a solution. Preferably, there would be many more than 4 sets, thus reducing uncertainty associated with the solution.

It is often desirable to constrain the system of equations, usually to reduce the degrees of freedom and reduce the uncertainty associated with the solution to a given system of equations. In fact, while one of the biggest advantages to using affine transformations might be that they include scale factors and shear corrections, in addition to rotation and translation, it has been our experience that allowing so many free parameters leads to over-fitting of the training data. When this happens, the solution, **M**, does not generalize well when used to adjust raw data between absolute measurements. If the frequency of absolute measurements could be increased substantially, it might be sufficient to reduce uncertainty enough that realistic scale and shear calibrations could be obtained from absolute measurements. Until such time, limiting the number of degrees of freedom helps avoid over-fitting the limited absolute training data.

We recommend doing so in a manner somewhat consistent with traditional (quasi)definitive processing. To start, if we look closely at all the transform matrices in the previous subsection, it is evident that the final row of **M** should always be  $[0 \ 0 \ 0 \ 1]$ . We can exploit this knowledge to remove the  $M_{4*}$  unknowns from our system of equations, and their corresponding columns in the measurement matrices, thus reducing our number of unknowns from 16 to 12, and slightly reducing uncertainty in the remaining estimated coefficients.

$$\begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ X_2 \\ Y_2 \\ Z_2 \\ \vdots \\ X_n \\ Y_n \\ Z_n \end{bmatrix} = \begin{bmatrix} H_1 & E_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & H_1 & E_1 & Z_1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & H_1 & E_1 & Z_1 & 1 \\ H_2 & E_2 & Z_2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & H_2 & E_2 & Z_2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & H_2 & E_2 & Z_2 & 1 \\ \vdots & \vdots \\ H_n & E_n & Z_n & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & H_n & E_n & Z_n & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & H_n & E_n & Z_n & 1 \end{bmatrix} \cdot \begin{bmatrix} M_{11} \\ M_{12} \\ M_{13} \\ M_{14} \\ M_{21} \\ M_{22} \\ M_{23} \\ M_{24} \\ M_{31} \\ M_{32} \\ M_{33} \\ M_{34} \end{bmatrix}$$

This simple constraint does not buy us much. But now suppose we wish to reduce the the number of free parameters even further by imposing additional prior knowledge/assumptions on the structure of our matrix that are (mostly) consistent with traditional definitive processing techniques. For example, assume that 1) the vertical vector component Z is perfectly aligned between the variometer and the absolute instrument, and 2) it is perfectly orthogonal to the horizontal components; also, 3) the transform of the horizontal vector component is a scaled rotation about the Z axis (that is, no skew). Finally, 4) don't allow any translation of the horizontal components, and 5) only allow translation of the vertical component. This is accomplished by removing and/or moving elements of the previous matrix equation to give the following:



$$\mathbf{C} = \mathbf{USV}^*$$

...where singular values  $s_i$  comprise the diagonal elements of  $\mathbf{S}$ , and the orthonormal matrices  $\mathbf{U}$  and  $\mathbf{V}^*$  can be combined to give the unscaled rotation that aligns  $HE$  with  $XY$ :

$$\mathbf{R} = \mathbf{V}^{*T} \cdot \mathbf{U}^T$$

Finally, with  $\mathbf{R}$ , simply rotate  $\overline{H}$  and  $\overline{E}$  into absolute coordinates to obtain  $T_X$  and  $T_Y$ :

$$\begin{bmatrix} T_X \\ T_Y \end{bmatrix} = \mathbf{R} \cdot \begin{bmatrix} \overline{H} \\ \overline{E} \end{bmatrix}$$

( $T_Z$  is simply the difference in the averages of the vertical components, just as with traditional processing)

Notice that there are now four degrees of freedom ( $\theta_z$ ,  $T_X$ ,  $T_Y$ , and  $T_Z$ ) instead of the three traditional (quasi)definitive baselines. This is because  $T_X$  and  $T_Y$  are allowed to float, rather than be trigonometrically locked to one another via  $\theta_z$ . We might effectively do this with an affine transformation, and obtain a near perfect analog to (quasi)definitive processing, but one of the main problems with the traditional approach presented above is that it requires that the fluxgate sensor be aligned with the local magnetic meridian for the math/geometry to work out. Using affine transformations, this is no longer necessary. Of course by increasing the degrees of freedom from 3 to 4, the uncertainty of our final solution is slightly higher; but then the uncertainty *should be* higher, since a potential offset in the variometer's  $E$  axis is now included in the calculations.

## Adaptive Coefficients

Even after switching to affine matrices to transform  $HEZ$  variometer vector observations into  $XYZ$  adjusted vectors, there remain long period biases in the residuals that often exceed (quasi)definitive tolerances. In order to remove these biases, and therefore track the non time-stationary relationship between the variometer and absolute instrumentation, some kind of time-adaptive algorithm is required. What's more, to support arbitrary epoch times, it is often more computationally efficient to interpolate between less frequent, but optimal transform matrices that we will refer to as "keyframes" (a concept borrowed from video compression).

## Weighted Least Squares

Absolute observations tend not to be regular, otherwise a recursive least-squares algorithm with a finite "memory" might be the best way to estimate non time-stationary affine matrices. However, since Absolute observations are also relatively infrequent (sampling intervals on the order of days to weeks), it is not unreasonable to build up data matrices from all, or a large subset of all, available observations, then estimate the optimal transformation for a given interval using ordinary least squares. However, to reduce noise and obtain a more reliable solution, typically many more observations than a bare minimum equal to the number of free parameters are required. This number may span an interval of several months or more, and since ordinary least squares gives the optimal solution for the entire interval, the keyframe affine matrix estimated for any given epoch may not reflect the true (noise-free) transform for that epoch.

If a method existed to emphasize more recent measurements over older measurements, the solution might be made more representative of the desired epoch, while still using many observations to reduce noise and obtain a reliable solution. Weighted least squares, with weights that are a function of the age of the observation relative to the epoch, is just such a method. There are many techniques and software libraries available to estimate a weighted least squares solution, so we just give a general matrix-oriented description here before discussing how to choose weights:

$$\hat{\mathbf{M}} = (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} \mathbf{y}$$

...where  $\mathbf{X}$  is the independent variable matrix containing the  $HEZ$  variometer observations described earlier,  $\mathbf{W}$  is a diagonal matrix whose elements are the weights to be assigned to each observation, and  $\mathbf{y}$  is the dependent variable array containing the  $XYZ$  absolute observations described earlier. If  $\mathbf{W}$  were the identity matrix,  $\mathbf{I}$ , weighted least squares reduces to ordinary least squares. The solution,  $\hat{\mathbf{M}}$ , is actually the vector of matrix coefficients described previously, so each element must then be placed into its corresponding location in the 2D matrix used to actually adjust the  $HEZ$  variometer vector measurements.

## Linear Interpolation Between Affine Transforms

It is straight-forward to linearly interpolate between scalar values that fall on a common axis. Almost all mathematical software libraries implement such a function, so we will not describe the algorithm here.

However, problems arise with the rotational components of an affine transform. Indeed, if linear interpolation were performed for each of the matrix elements used to define a rotation, none of the interpolated matrices would actually be a rotation matrix, and strange, often non-physical transformations would result. Fortunately, this problem has a long-established solution known as spherical linear interpolation (Slerp; Shoemake, 1985. "Animating Rotation with Quaternion Curves" (PDF). SIGGRAPH 1985.).

Briefly, the Slerp technique finds the vector axis of rotation that characterizes each given rotation matrix, then rotates from the first to the last at a constant rate as if the axis of rotation was tracing out a unit sphere. Slerp is not nearly as common as standard linear interpolation, but Scientific Python's spatial transforms sub-package.

Assuming one is starting with an arbitrary affine transformation matrix, it is necessary to first decompose the matrix into scale, rotation, and translational components. There are numerous ways to decompose matrices this way, including Singular Value Decomposition (SVD) as described above. A simpler, and arguably more robust method involves so-called polar decomposition, and is described in detail by Shoemake and Duff (1992, "Matrix animation and polar decomposition" in *Proceedings of the Conference on Graphics Interface '92*, Morgan Kauffman Publishers, Inc., San Francisco, pp 258-264).

Once decomposed, standard linear interpolation is performed on the scale and translation matrices, while Slerp is applied to the rotation matrix. The constituent transforms are recombined as described previously to give a valid, more physically consistent, affine transform at each intermediate time step.

## Anomaly Detection

TBD

## Demonstration and Validation

The purpose of this section is to demonstrate that the algorithm works as expected, and more critically, validate the algorithm using real data, including a comparison with reviewed quasi-definitive data. While some material here might be extracted to generate unit tests for an official "Geomag-Algorithm", these are primarily *functional* tests, and may be more complex than one might wish to incorporate into an automated testing framework. Explanatory markdown, inline comments, or both, will describe the purpose of each subsection below:

## Notebook Functions

Before proceeding, it is necessary to run the cells in this subsection to define and initialize functions used in the actual demonstrations and validations. Some functions have a general functional interface, with multiple actual functions to reflect alternative algorithmic choices.

### Imports

```
In [2]: 1 ▶ # import Python libraries↔
```

### Baseline and Absolute Data Retrieval

Functions here should retrieve baseline and absolutes measurements:

#### Inputs

```
obs_code    - 3-character IAGA code for observatory
start_date  - UTCDateTime for start of interval
end_date    - UTCDateTime for end of interval
```

#### Options

```
path_or_url - string that holds a base path or url at which to
              find baseline and absolute observations
              (default = max(times))
```

#### Output

```
h_abs_bas_utc - array holding vectors of h_abs, h_bas, and h_utc
d_abs_bas_utc - array holding vectors of d_abs, d_bas, and d_utc
z_abs_bas_utc - array holding vectors of z_abs, z_bas, and z_utc
pc            - array holding pier corrections
```

**FIXME:** Presently, functions here rely on an *ad hoc* "flag" that is defined by baseline observations with absolute H values that are all zero to identify a known change in the observatory configuration. All observations prior to that time will be discarded. This needs to be modified to incorporate a desired "epoch", and to discard observations on either side of that epoch (past or future) that occur before/after observatory reconfigurations.

```
In [3]: 1 ▶ def retrieve_baselines_resid_summary_xlsm(obs_code, start_date, end_date,↔
```

```
In [4]: 1 ▶ def retrieve_baselines_webabsolutes(obs_code, start_date=None, end_date=None,↔
```

### Observation Time Weighting Functions

Functions here should calculate time-dependent weights given:

#### Inputs

times - 1D array of times, or any time-like index whose relative values represent spacing between events

memory - time scale over which weights decrease by a prescribed amount relative to the maximum weight

**Options**

epoch - time at which weights maximize  
(default = max(times))

**Output**

weights - 1D array of weights

```
In [5]: 1 ▶ def time_weights_exponential(times, memory, epoch=None):↔
```

```
In [6]: 1 ▶ def time_weights_linear(times, memory, epoch=None):↔
```

## Vector Distance Calculator

Function here should calculate vector distances given:

**Inputs**

vectors\_A - NxM array where N is number of vector axes and M is number of observations

vectors\_B - NxM array where N is number of vector axes and M is number of observations

**Options**

metric - string specifying a supported metric

VI - inverse (co)variance by which to scale distances  
(if None, calculate scaling (co)variance appropriate for metric from vectors\_A and vectors\_B, then apply)

**Output**

dist - an M element array of vector distances/metrics

```
In [7]: 1 ▶ def vector_dist(vectors_A, vectors_B, metric=None, VI=None):↔
```

## Statistical Time Series Filters

Functions here should identify "good" elements in a univariate series:

**Inputs**

series - univariate data to filter

**Options**

threshold - threshold value to be used for filter

weights - weights that can be applied to series  
(defaults to uniform if None)

**Output**

good - a boolean array where True corresponds to "good" observations

```
In [8]: 1 ▶ def filter_zscore(series, threshold=None, weights=None):↔
```

```
In [9]: 1 ▶ def filter_iqr(series, threshold=None, weights=None):↔
```

## Affine Transform Matrix Generators

Functions here should generate a 4x4 affine transformation matrix given:

**Inputs**

- ord - 3xN array of training data where rows correspond to 3D Cartesian vectors, and columns are observations; these are the "raw" vector input to be transformed
- abs - 3xN array of training data where rows correspond to 3D Cartesian vectors, and columns are observations; these are the desired "absolute" vector output

**Options**

- weights - array of N weights that can be applied to observations (defaults to uniform if None)

**Output**

- M - a 4x4 affine transformation matrix that maps ord to abs  
NOTE: functions should include some sort of condition check (e.g., minimum rank of system), and return a 4x4 matrix of NaNs if this check fails

```
In [10]: 1 ▶ def generate_affine_0(ord_hez, abs_xyz, weights=None):↔
```

```
In [11]: 1 ▶ def generate_affine_1(ord_hez, abs_xyz, weights=None):↔
```

```
In [12]: 1 ▶ def generate_affine_2(ord_hez, abs_xyz, weights=None):↔
```

```
In [13]: 1 ▶ def generate_affine_3(ord_hez, abs_xyz, weights=None):↔
```

```
In [14]: 1 ▶ def generate_affine_4(ord_hez, abs_xyz, weights=None):↔
```

```
In [15]: 1 ▶ def generate_affine_5(ord_hez, abs_xyz, weights=None):↔
```

```
In [16]: 1 ▶ def generate_affine_6(ord_hez, abs_xyz, weights=None):↔
```

```
In [17]: 1 ▶ def generate_affine_7(ord_hez, abs_xyz, weights=None):↔
```

```
In [18]: 1 ▶ def generate_affine_8(ord_hez, abs_xyz, weights=None):↔
```

```
In [19]: 1 ▶ def generate_affine_9(ord_hez, abs_xyz, weights=None):↔
```

**Affine Transform Matrix Interpolators**

Functions here should interpolate between 4x4 affine transformation matrices:

**Inputs**

- utc\_target - list of UTCs at which to interpolate affine matrices
- utc\_list - list of UTCs that correspond to a list of known affine matrices
- affine\_list - list of known affine matrices

**Options**

- fill\_value - if None, disallow extrapolation; if not None, use this value when utc\_target falls outside utc\_list range; if "extrapolate", extrapolate based on first/last two in affine\_list.

**Output**

- affine\_target - list of interpolated affine matrices

```
In [20]: 1 ▶ def interpolate_affine_polar(utc_target, utc_list, affine_list, fill_value=None):↔
```

**Do-It-All Demonstration and Validation Wrapper**

Function that retrieves baseline data, filters baseline data, calculates affine transforms, and optionally, applies transforms to raw data and pulls down (quasi-)definitive data for validation:

**Inputs**

```

obs_code      - 3-character IAGA code for observatory
start.UTC     - beginning date to do stuff (UTCdatetime)
end.UTC       - final date to do stuff (UTCdatetime)

```

### Options

```

update_interval - how often (in seconds) to update the Adjusted matrices
                  (default = end.UTC - start.UTC)
acausal         - use absolute/ordinate pairs from the future if True
                  (default = False)
interpolate     - interpolate between key transforms
                  (default = False)
first.UTC       - earliest observation date to retrieve
                  (default = start.UTC)
last.UTC        - latest observation date to retrieve
                  (default = end.UTC)
M_funcs         - list of function objects used to generate affine matrices
                  given 3D Cartesian vector inputs; compose final Adjusted
                  affine matrix by:
                  1) calculate 1st matrix from inputs->outputs;
                  2) transform initial inputs to intermediate inputs;
                  3) calculate 2nd matrix from intermediate inputs to outputs;
                  4) repeat until all M_funcs used;
                  5) final Adjusted matrix is composition of all in reverse
                  (default = [generate_affine_0])
memories        - time constant(s) used to calculate weights; memories may be
                  a scalar, or a list with same length as M_funcs
                  (default = np.inf)
path_or_url     - url for absolutes web service, or path to summary xlsx files
                  (default = 'https://geomag.usgs.gov/')
validate        - if True, pull and process raw data, then compare with QD
                  (default = False)
edge_host       - edge host for raw and QD magnetometer time series
                  (default = 'cwbpub.cr.usgs.gov')

```

### Output

```

utc_list        - list of first UTCDateTimes for each update_interval
M_composed_list - list of composed Adjusted Data matrices for each update_interval
pc_list         - list of pier corrections for each update_interval

(if validate is True)
utc_xyzf_list   - list of UTCDateTime arrays for each observation
xyzf_trad_list  - list of static baseline adjusted data arrays for each update_interval
xyzf_adj_list   - list of Adjusted Data arrays for each update_interval
xyzf_def_list   - list of Definitive Data arrays for each update_interval
utc_bas         - UTCDateTimes for absolute measurements
abs_xyz         - absolute XYZ values used to train affine transforms
ord_hez         - ordinate HEZ values used to train affine transforms
Ms_list         - list of lists of Adjusted Data matrices for each M_func,
                  for each update_interval
weights_list    - list of lists of weights used to estimate Adjusted Data
                  matrices for each M_func, for each update_interval

```

```
In [21]: 1 ▶ def do_it_all(obs_code, start.UTC, end.UTC,↔
```

## Synthetic Data Demonstration

This subsection demonstrates the general capabilities of affine transforms with carefully constructed, non-physical synthetic data. It may contain snippets/gists that can be adapted into unit tests.

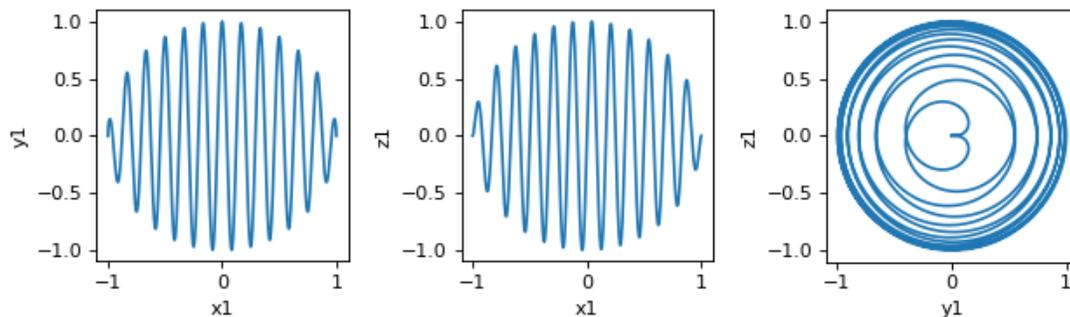
### Construct synthetic time series

First, define a regular series of times  $t$ , then construct synthetic "Truth" vectors  $XYZ$  that represent the desired results as functions of  $t$ :

- regular sample times  $t$  fall between 0 and  $\pi$
- $X_{\text{truth}} = (2t - \pi)/\pi$
- $Y_{\text{truth}} = \sqrt{1 - X_{\text{truth}}^2} \cos(24t)$

$$\bullet Z_{\text{truth}} = \sqrt{1 - X_{\text{truth}}^2} \sin(24t)$$

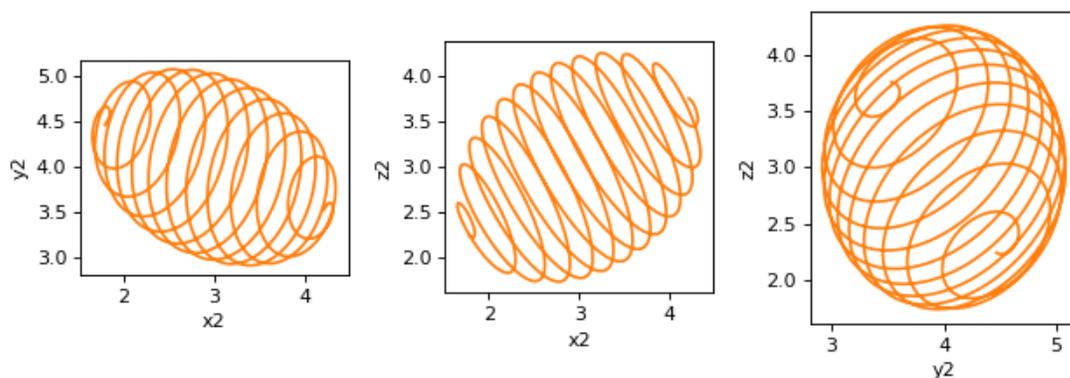
```
In [22]: 1 ▶ # define and plot truth vectors↔
```



Next, define an affine transformation matrix that converts the truth vectors into a different "Observed" basis:

- scale XYZ axes by [1.5, 0.95, 1.2], respectively
- rotate basis by 45 degrees about the [1, 1, 1] vector
- translate origin by [3, 4, 3], respectively

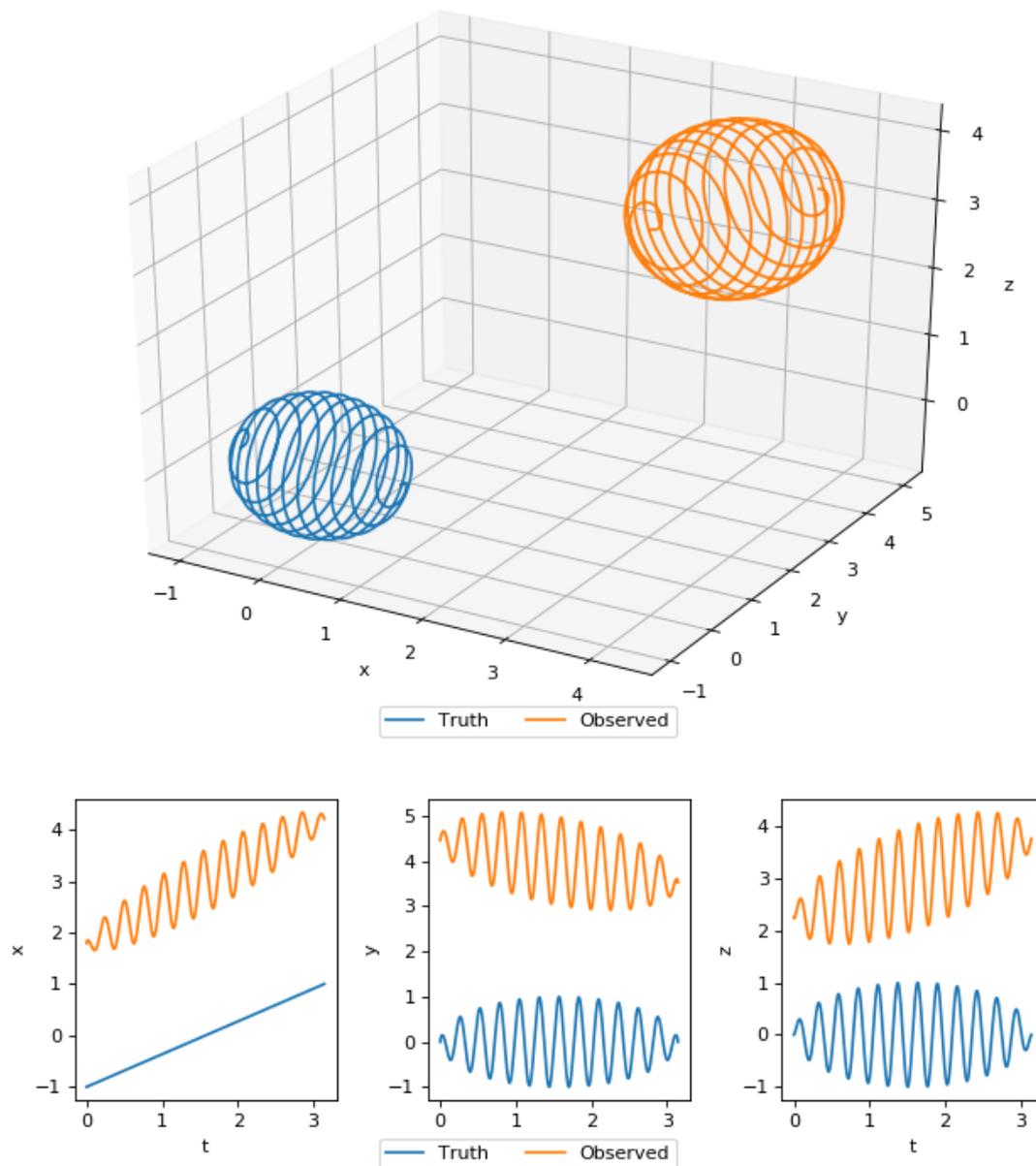
```
In [23]: 1 ▶ # define affine matrix, transform truth vectors, and plot results in new basis↔
```



Compare 'Truth' and 'Observed' vectors directly:

- 3D view
- time series view

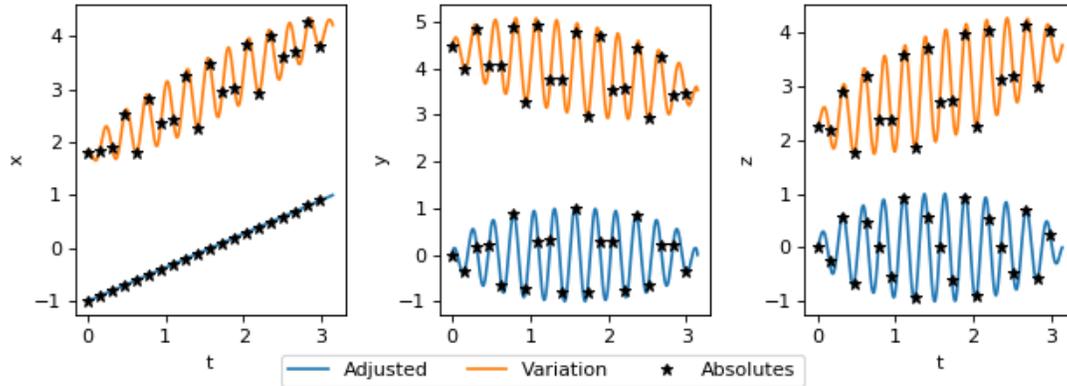
```
In [24]: 1 ▶ # plot a 3D view of 'truth' and transformed vectors↔
```



### Estimate Affine Transformation Matrix

- subsample "Truth" (similar to absolute measurements);
- subsample "Observed" (similar to variation measurements);
- solve for optimal affine transformation (Observed -> Truth);
- plot "adjusted", "variation", and "absolute" training data;
- compare original affine and inverted solution matrices.

```
In [25]: 1 ▶ # sub-sample observations, solve for M, apply transform, and plot results↔
```



```
[ [ 1.207  0.481 -0.373  3. ]
  [-0.466  0.765  0.607  4. ]
  [ 0.759 -0.295  0.966  3. ]
  [ 0.      0.      0.      1. ] ]
```

```
[ [ 1.207  0.481 -0.373  3. ]
  [-0.466  0.765  0.607  4. ]
  [ 0.759 -0.295  0.966  3. ]
  [ 0.      0.      0.      1. ] ]
```

## Comparison with (Quasi-)Definitive Data

This subsection cycles through each of the USGS observatories, retrieves ~6 months worth of baseline data, and generates three different affine transforms:

- a "short" memory, causal transform that is similar to what would be used in real time operations;
- a "short" memory, acausal transform to demonstrate the improvement possible if future observations are used, and is similar to an automated quasi-definitive;
- an infinite memory, acausal transform to demonstrate the improvement possible with any kind of adaptive affine transform estimation.

Furthermore, these transforms are applied to raw data to validate the technique over that ~6 month interval, and compared to quasi-definitive data when it is available (not all USGS observatories have QD data available at all times). The comparisons are vector-component by vector-component differences, a Euclidean "distance" between vectors, and the traditional " $\Delta F$ " metric, which is the only one that does NOT require QD data, and is available for each observatory.

In general, QD data is considered the "gold standard" to which we strive. The short memory, acausal transform is generally comparable to QD. The infinite memory, acausal transform highlights the non-stationarity observatory baselines. The short memory, causal transform is an improvement over the infinite memory acausal, but tends to lag with a time constant that is defined by the memory. Also, to make this comparison as similar to actual operations as possible, the causal transforms were NOT interpolated like the others, resulting in step changes. Interpolation does significantly improve results, but this is not surprising, nor especially meaningful in the context of this validation study.

Finally, while most of the results speak for themselves, it is important to call attention to the CMO results. The reader will quickly notice that the QD data from the CMO observatory tend to exhibit bias, even/especially in the  $\Delta F$  metric. This is because CMO (and DED) baselines are generated using different software and geometric assumptions than the rest of the observatories. All processing performed in this notebook makes appropriate adjustments, but QD data, which is done using the USGS' MagProc software, does not.

## Boulder (BOU) Observatory

```
In [26]: 1 # configuration parameters for BOU
2
3 # INPUTS
4 obs_code = 'BOU'
5 start.UTC = UTCDateTime('2019-01-01T00:00:00Z')
6 end.UTC = UTCDateTime('2019-06-30T23:59:00Z')
7
8 # OPTIONS
9 update_interval = 86400 * 7
10 acausal = False
11 first.UTC = UTCDateTime('2018-10-01T00:00:00Z')
12 last.UTC = UTCDateTime('2019-07-31T23:59:00Z')
13
14 # This is slowly evolving horizontal rotation, and
15 # quickly evolving baseline offsets (including Ebase)
16 M_funcs = [generate_affine_8, generate_affine_6]
17 memories = [86400 * 100, 86400 * 10]
18
19 #path_or_url = '/Volumes/geomag/pub/Caldata/Checked Baseline Data/'
20 path_or_url = 'https://geomag.usgs.gov'
21
22 validate = True
23 edge_host = 'cwbpublish.cr.usgs.gov'
```

Run do\_it\_all() with a "short" memory in causal mode

```
In [27]: 1 %%capture↔
```

Run do\_it\_all() with a "short" memory in acausal mode

```
In [28]: 1 %%capture↔
```

Run do\_it\_all() with infinite memory, but update every update\_interval

```
In [29]: 1 %%capture↔
```

Run do\_it\_all() with infinite memory, for entire interval

```
In [30]: 1 %%capture↔
```

Print the last causal M matrix, then save it to a JSON file for production runs.

```
In [31]: 1 # print and save last M matrix↔

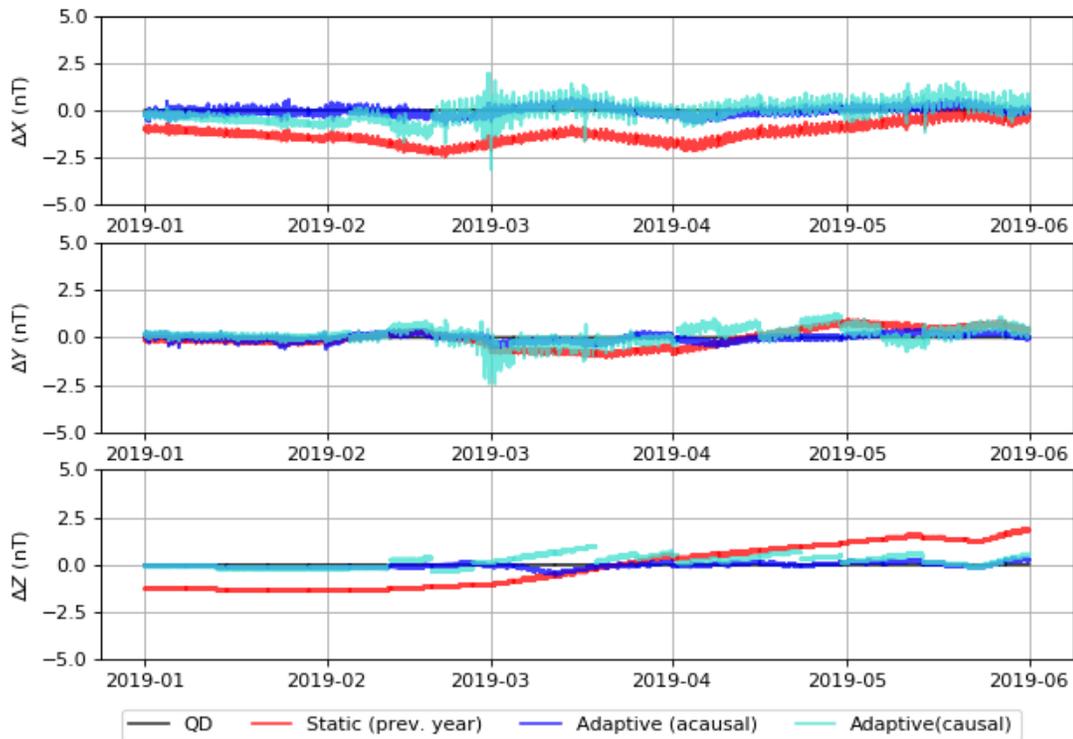
[[ 9.888e-01 -1.491e-01 0.000e+00 -7.364e+01]
 [ 1.491e-01 9.888e-01 0.000e+00 -8.578e+00]
 [ 0.000e+00 0.000e+00 1.000e+00 5.744e+02]
 [ 0.000e+00 0.000e+00 0.000e+00 1.000e+00]]
```

Plot differences between different "adjusted" data (including quasi-definitive) and a static version of traditional HDZ baseline-adjusted data

```
In [ ]: 1 # plot differences with traditional HDZ baseline-adjusted data↔
```

Plot differences between different "adjusted" and quasi-definitive data

In [32]: 1 ▶ # plot differences↔

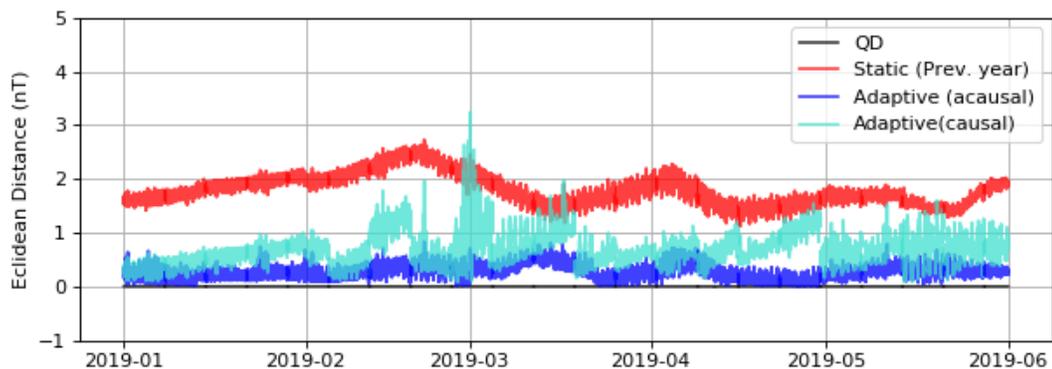


```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
warnings.warn(message, mplDeprecation, stacklevel=1)
```

Out[32]: <matplotlib.legend.Legend at 0x1c383b2470>

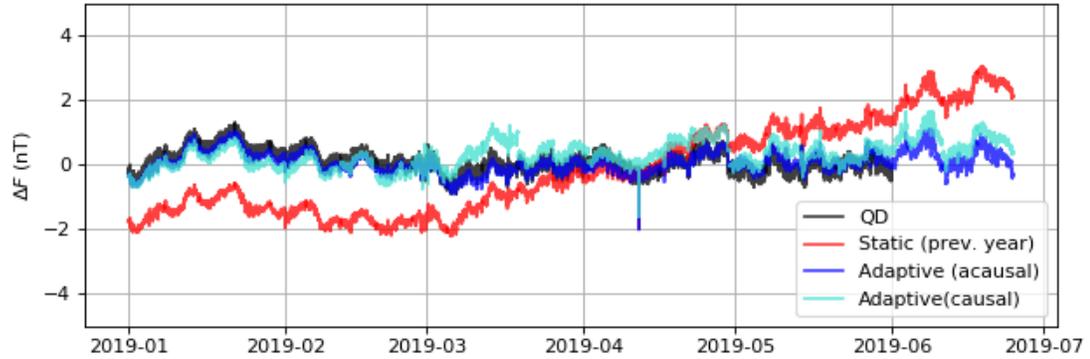
Plot Euclidean (vector) distance between different "Adjusted" and quasi-definitive data.

In [33]: 1 ▶ # Plot Euclidean distances↔



Plot  $\Delta F$  for different "adjusted" and quasi-definitive data

```
In [34]: 1 ▶ # plot delta-Fs ↔
```



## Barrow (BRW) Observatory

```
In [35]: 1 ▾ # configuration parameters for BRW
2
3 # INPUTS
4 obs_code = 'BRW'
5 start.UTC = UTCDateTime('2019-01-01T00:00:00Z')
6 end.UTC = UTCDateTime('2019-06-30T23:59:00Z')
7
8 # OPTIONS
9 update_interval = 86400 * 7
10 acausal = False
11 first.UTC = UTCDateTime('2018-10-01T00:00:00Z')
12 last.UTC = UTCDateTime('2019-07-31T23:59:00Z')
13
14 # This is slowly evolving horizontal rotation, and
15 # quickly evolving baseline offsets (including Ebase)
16 M_funcs = [generate_affine_8, generate_affine_6]
17 memories = [86400 * 100, 86400 * 10]
18
19 #path_or_url = '/Volumes/geomag/pub/Caldata/Checked Baseline Data/'
20 path_or_url = 'https://geomag.usgs.gov'
21
22 validate = True
23 edge_host = 'cwbpub.cr.usgs.gov'
```

Run do\_it\_all() with a "short" memory in causal mode

```
In [36]: 1 ▶ %%capture↔
```

Run do\_it\_all() with a "short" memory in acausal mode

```
In [37]: 1 ▶ %%capture↔
```

Run do\_it\_all() with infinite memory, but update every update\_interval

```
In [38]: 1 ▶ %%capture↔
```

Run do\_it\_all() with infinite memory, for entire interval

```
In [39]: 1 ▶ %%capture↔
```

Print the last causal M matrix, then save it to a JSON file for production runs.

```
In [40]: 1 ▶ # print and save last M matrix↔

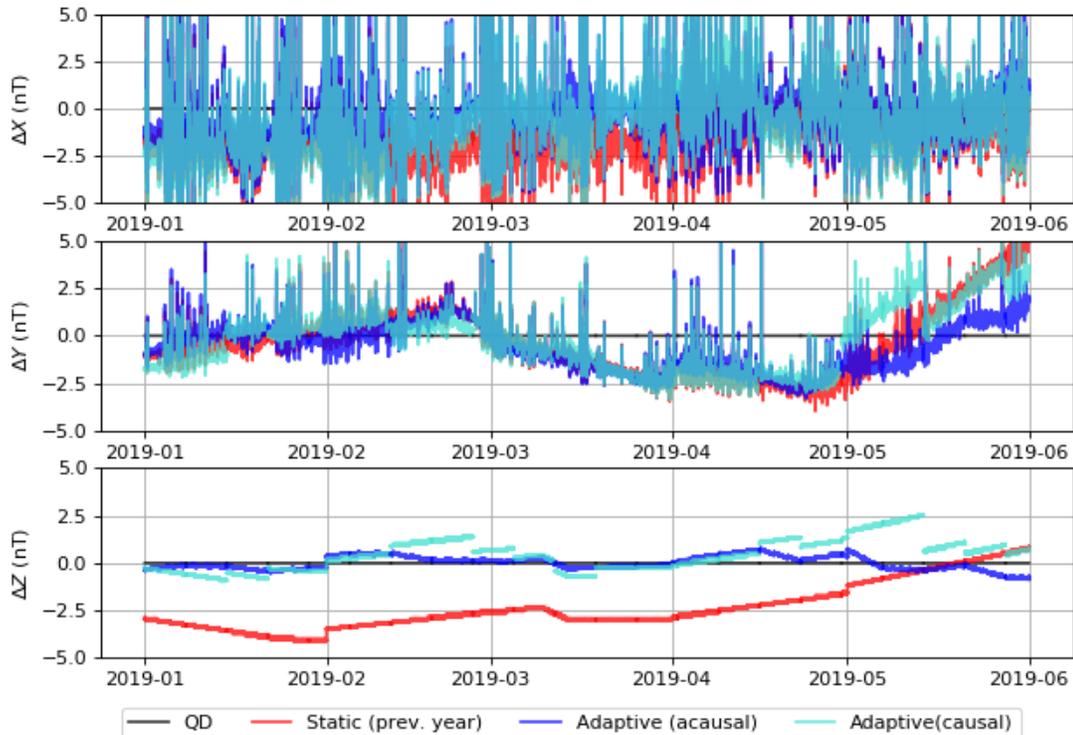
[[ 0.966 -0.258  0. -122.238]
 [ 0.258  0.966  0.  -9.155]
 [ 0.      0.      1. -141.274]
 [ 0.      0.      0.   1.   ]]
```

Plot differences between different "adjusted" data (including quasi-definitive) and a static version of traditional HDZ baseline-adjusted data

```
In [ ]: 1 ▶ # plot differences with traditional HDZ baseline-adjusted data↔
```

Plot differences between different "adjusted" and quasi-definitive data

```
In [41]: 1 ▶ # plot differences↔
```

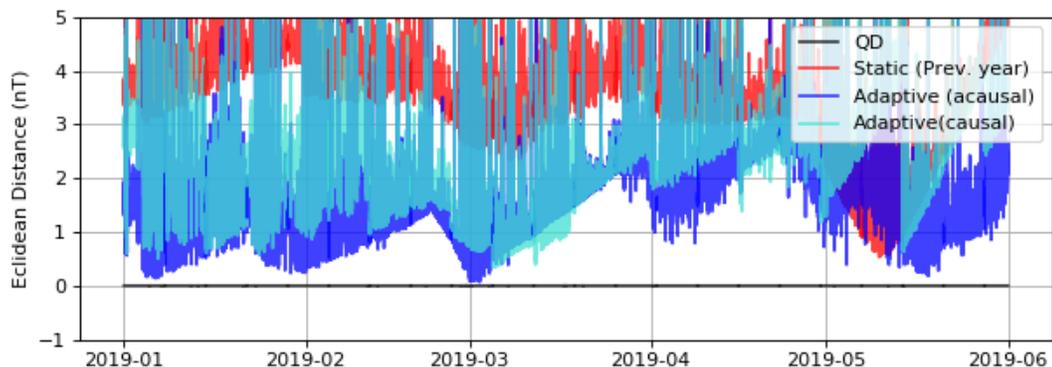


```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
warnings.warn(message, mplDeprecation, stacklevel=1)
```

```
Out[41]: <matplotlib.legend.Legend at 0x1c2fb04518>
```

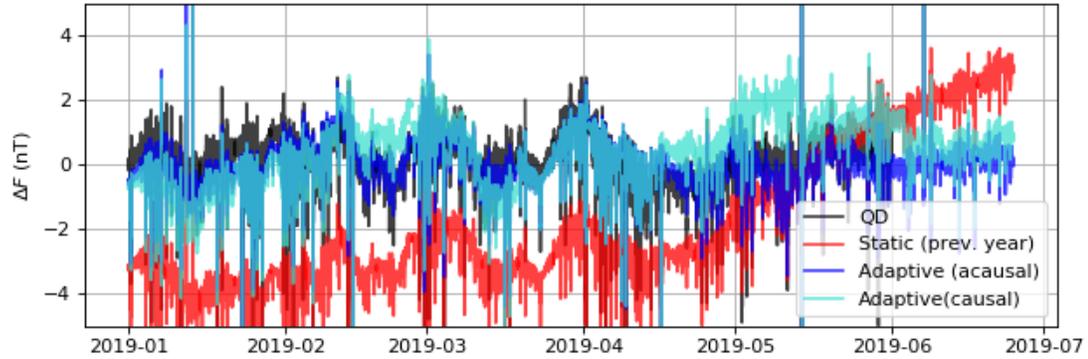
Plot Euclidean (vector) distance between different "Adjusted" and quasi-definitive data.

```
In [42]: 1 ▶ # Plot Euclidean distances↔
```



Plot  $\Delta F$  for different "adjusted" and quasi-definitive data

In [43]: 1 ▶ # plot delta-Fs ↔



## Stennis (BSL) Observatory

```
In [44]: 1 ▾ # configuration parameters for BSL
2
3 # INPUTS
4 obs_code = 'BSL'
5 start.UTC = UTCDateTime('2019-01-01T00:00:00Z')
6 end.UTC = UTCDateTime('2019-06-30T23:59:00Z')
7
8 # OPTIONS
9 update_interval = 86400 * 7
10 acausal = False
11 first.UTC = UTCDateTime('2018-10-01T00:00:00Z')
12 last.UTC = UTCDateTime('2019-07-31T23:59:00Z')
13
14 # This is slowly evolving horizontal rotation, and
15 # quickly evolving baseline offsets (including Ebase)
16 M_funcs = [generate_affine_8, generate_affine_6]
17 memories = [86400 * 100, 86400 * 10]
18
19 #path_or_url = '/Volumes/geomag/pub/Caldata/Checked Baseline Data/'
20 path_or_url = 'https://geomag.usgs.gov'
21
22 validate = True
23 edge_host = 'cwbpub.cr.usgs.gov'
```

Run do\_it\_all() with a "short" memory in causal mode

In [45]: 1 ▶ %%capture↔

Run do\_it\_all() with a "short" memory in acausal mode

In [46]: 1 ▶ %%capture↔

Run do\_it\_all() with infinite memory, but update every update\_interval

In [47]: 1 ▶ %%capture↔

Run do\_it\_all() with infinite memory, for entire interval

In [48]: 1 ▶ %%capture↔

Print the last causal M matrix, then save it to a JSON file for production runs.

In [49]: 1 ▶ # print and save last M matrix↔

```
[[ 9.966e-01  8.293e-02  0.000e+00  7.297e+02]
 [-8.293e-02  9.966e-01  0.000e+00  1.635e+03]
 [ 0.000e+00  0.000e+00  1.000e+00  9.679e+01]
 [ 0.000e+00  0.000e+00  0.000e+00  1.000e+00]]
```

Plot differences between different "adjusted" data (including quasi-definitive) and a static version of traditional HDZ baseline-adjusted data

```
In [ ]: 1 ▶ # plot differences with traditional HDZ baseline-adjusted data↔
```

Plot differences between different "adjusted" and quasi-definitive data

**Note:** Quasi-definitive data not available for BSL

```
In [ ]: 1 ▶ # plot differences↔
```

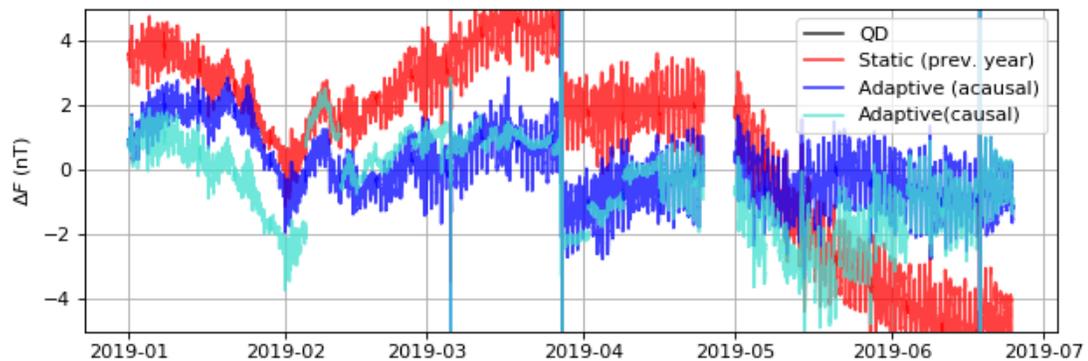
Plot Euclidean (vector) distance between different "Adjusted" and quasi-definitive data.

**Note:** Quasi-definitive data not available for BSL

```
In [ ]: 1 ▶ # Plot Euclidean distances↔
```

Plot  $\Delta F$  for different "adjusted" and quasi-definitive data

```
In [51]: 1 ▶ # plot delta-Fs ↔
```



## College (CMO) Observatory

```
In [52]: 1 ▾ # configuration parameters for CMO
2
3 # INPUTS
4 obs_code = 'CMO'
5 start_UTC = UTCDateTime('2019-01-01T00:00:00Z')
6 end_UTC = UTCDateTime('2019-06-30T23:59:00Z')
7
8 # OPTIONS
9 update_interval = 86400 * 7
10 acausal = False
11 first_UTC = UTCDateTime('2018-12-01T00:00:00Z')
12 last_UTC = UTCDateTime('2019-07-31T23:59:00Z')
13
14 # This is slowly evolving horizontal rotation, and
15 # quickly evolving baseline offsets (including Ebase)
16 M_funcs = [generate_affine_8, generate_affine_6]
17 memories = [86400 * 100, 86400 * 10]
18
19 path_or_url = '/Volumes/geomag/pub/Caldata/Checked Baseline Data/'
20 #path_or_url = 'https://geomag.usgs.gov'
21
22 validate = True
23 edge_host = 'cwbpub.cr.usgs.gov'
```

Run do\_it\_all() with a "short" memory in causal mode

```
In [53]: 1 ▶ %%capture↔
```

Run do\_it\_all() with a "short" memory in acausal mode

```
In [54]: 1 ▶ %%capture↔
```

Run do\_it\_all() with infinite memory, but update every update\_interval

```
In [55]: 1 ▶ %%capture↔
```

Run do\_it\_all() with infinite memory, for entire interval

```
In [56]: 1 ▶ %%capture↔
```

Print the last causal M matrix, then save it to a JSON file for production runs.

```
In [57]: 1 ▶ # print and save last M matrix↔
```

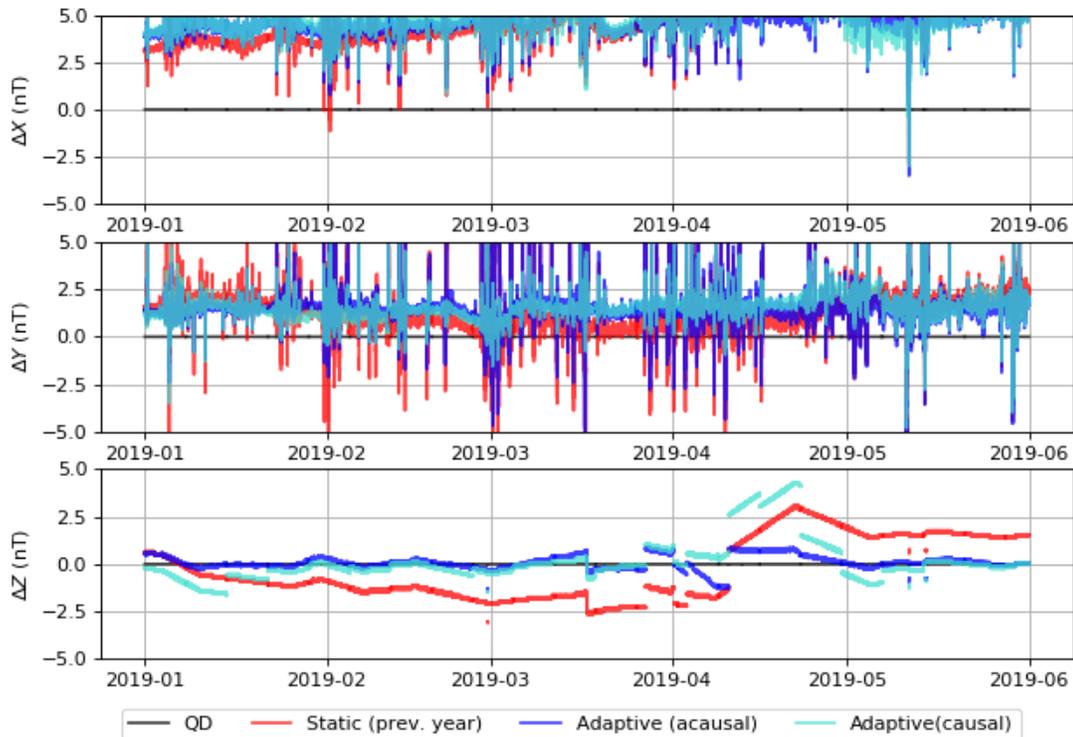
```
[[ 0.953 -0.304 0.    97.479]
 [ 0.304 0.953 0.   269.167]
 [ 0.    0.    1.  -55.313]
 [ 0.    0.    0.    1.   ]]
```

Plot differences between different "adjusted" data (including quasi-definitive) and a static version of traditional HDZ baseline-adjusted data.

**Note:** H baselines are defined differently for WebAbsolutes and the USGS residual method spreadsheets (which is where CMO baselines are obtained from here). The do\_it\_all() function in this notebook properly accounts for this, but MagProc does not seem to, thus leading to bias in quasi-definitive's X and Y calculations, and ultimately  $\Delta F$ .

```
In [ ]: 1 ▶ # plot differences with traditional HDZ baseline-adjusted data↔
```

```
In [58]: 1 ▶ # plot differences↔
```



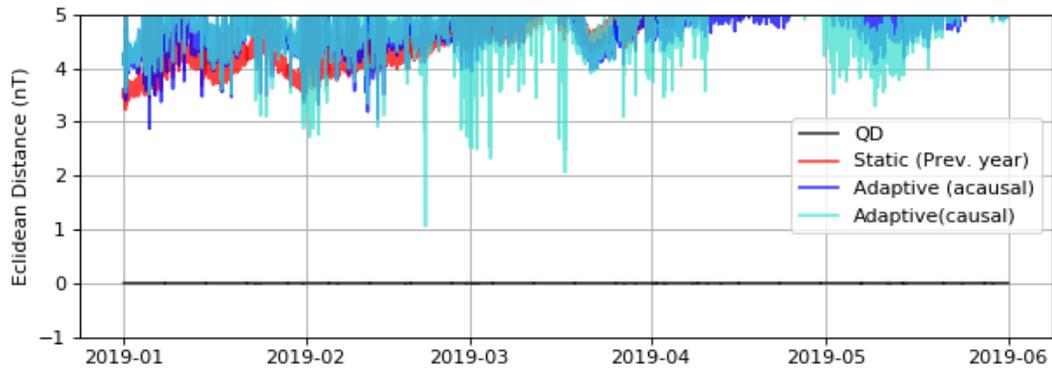
```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/cbook/deprecation.py:107: M
atplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the ea
rlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warn
ing can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
warnings.warn(message, mplDeprecation, stacklevel=1)
```

```
Out[58]: <matplotlib.legend.Legend at 0x1c28a2bac8>
```

Plot Euclidean (vector) distance between different "Adjusted" and quasi-definitive data.

**Note:** H baselines are defined differently for WebAbsolutes and the USGS residual method spreadsheets (which is where CMO baselines are obtained from here). The do\_it\_all() function in this notebook properly accounts for this, but MagProc does not seem to, thus leading to bias in quasi-definitive's X and Y calculations, and ultimately  $\Delta F$ .

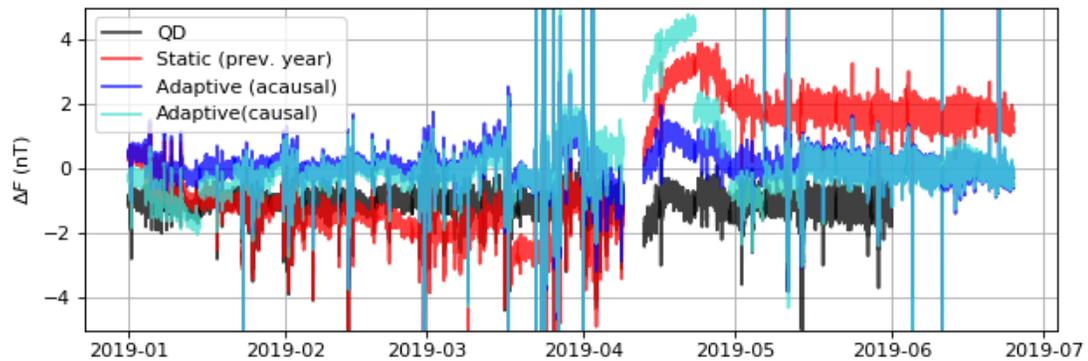
In [59]: 1 ▶ # Plot Euclidean distances↔



Plot  $\Delta F$  for different "adjusted" and quasi-definitive data.

**Note:** H baselines are defined differently for WebAbsolutes and the USGS residual method spreadsheets (which is where CMO baselines are obtained from here). The `do_it_all()` function in this notebook properly accounts for this, but MagProc does not seem to, thus leading to bias in quasi-definitive's X and Y calculations, and ultimately  $\Delta F$ .

In [60]: 1 ▶ # plot delta-Fs ↔



## Deadhorse (DED) Observatory

```
In [61]: 1 # configuration parameters for DED
2
3 # INPUTS
4 obs_code = 'DED'
5 start_utc = UTCDateTime('2019-01-01T00:00:00Z')
6 end_utc = UTCDateTime('2019-06-30T23:59:00Z')
7
8 # OPTIONS
9 update_interval = 86400 * 7
10 acausal = False
11 first_utc = UTCDateTime('2018-10-01T00:00:00Z')
12 last_utc = UTCDateTime('2019-07-31T23:59:00Z')
13
14 # This is slowly evolving horizontal rotation, and
15 # quickly evolving baseline offsets (including Ebase)
16 M_funcs = [generate_affine_8, generate_affine_6]
17 memories = [86400 * 100, 86400 * 10]
18
19 path_or_url = '/Volumes/geomag/pub/Caldata/Checked Baseline Data/'
20 #path_or_url = 'https://geomag.usgs.gov'
21
22 validate = True
23 edge_host = 'cwbpub.cr.usgs.gov'
```

Run `do_it_all()` with a "short" memory in causal mode

In [62]: 1 ▶ %%capture↔

Run do\_it\_all() with a "short" memory in acausal mode

In [63]: 1 ▶ %%capture↔

Run do\_it\_all() with infinite memory, but update every update\_interval

In [64]: 1 ▶ %%capture↔

Run do\_it\_all() with infinite memory, for entire interval

In [65]: 1 ▶ %%capture↔

Print the last causal M matrix, then save it to a JSON file for production runs.

In [66]: 1 ▶ # print and save last M matrix↔

```
[ [ 0.951  -0.311  0.   -101.46 ]
 [ 0.311  0.951  0.    38.33 ]
 [ 0.     0.     1.    12.488]
 [ 0.     0.     0.     1.   ]]
```

Plot differences between different "adjusted" data (including quasi-definitive) and a static version of traditional HDZ baseline-adjusted data

In [ ]: 1 ▶ # plot differences with traditional HDZ baseline-adjusted data↔

Plot differences between different "adjusted" and quasi-definitive data

**Note:** Quasi-definitive data not available for DED

In [ ]: 1 ▶ # plot differences↔

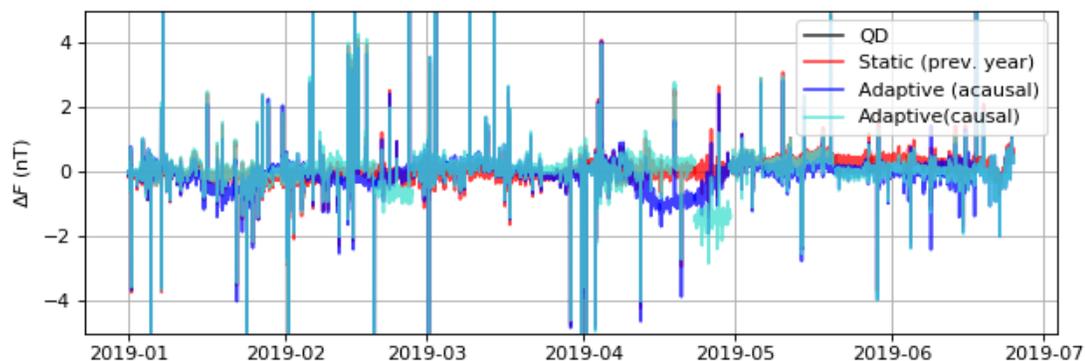
Plot Euclidean (vector) distance between different "Adjusted" and quasi-definitive data.

**Note:** Quasi-definitive data not available for DED

In [ ]: 1 ▶ # Plot Euclidean distances↔

Plot  $\Delta F$  for different "adjusted" and quasi-definitive data

In [67]: 1 ▶ # plot delta-Fs ↔



## [Fredericksburgh \(FRD\) Observatory](#)

```
In [68]: 1 # configuration parameters for FRD
2
3 # INPUTS
4 obs_code = 'FRD'
5 start.UTC = UTCDateTime('2019-01-01T00:00:00Z')
6 end.UTC = UTCDateTime('2019-06-30T23:59:00Z')
7
8 # OPTIONS
9 update_interval = 86400 * 7
10 acausal = False
11 first.UTC = UTCDateTime('2018-10-01T00:00:00Z')
12 last.UTC = UTCDateTime('2019-07-31T23:59:00Z')
13
14 # This is slowly evolving horizontal rotation, and
15 # quickly evolving baseline offsets (including Ebase)
16 M_funcs = [generate_affine_8, generate_affine_6]
17 memories = [86400 * 100, 86400 * 10]
18
19 #path_or_url = '/Volumes/geomag/pub/Caldata/Checked Baseline Data/'
20 path_or_url = 'https://geomag.usgs.gov'
21
22 validate = True
23 edge_host = 'cwbpublish.cr.usgs.gov'
```

Run do\_it\_all() with a "short" memory in causal mode

```
In [69]: 1 %%capture↔
```

Run do\_it\_all() with a "short" memory in acausal mode

```
In [70]: 1 %%capture↔
```

Run do\_it\_all() with infinite memory, but update every update\_interval

```
In [71]: 1 %%capture↔
```

Run do\_it\_all() with infinite memory, for entire interval

```
In [72]: 1 %%capture↔
```

Print the last causal M matrix, then save it to a JSON file for production runs.

```
In [73]: 1 # print and save last M matrix↔

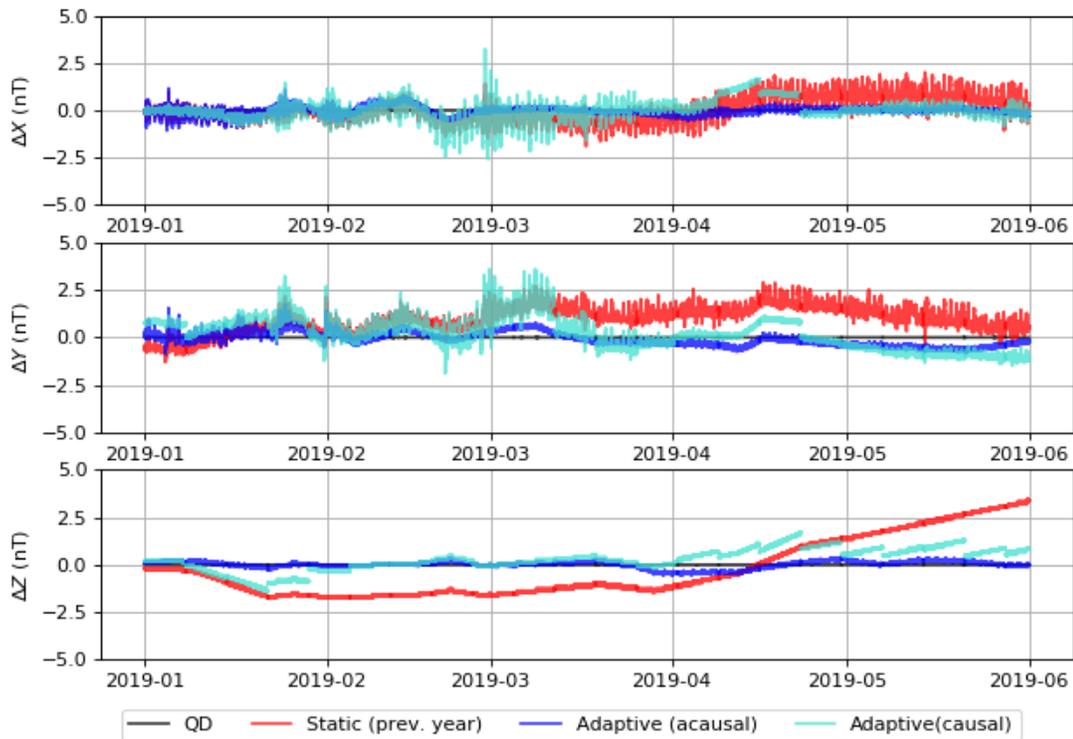
[[ 9.832e-01  1.827e-01  0.000e+00 -1.003e+02]
 [-1.827e-01  9.832e-01  0.000e+00  1.187e+01]
 [ 0.000e+00  0.000e+00  1.000e+00  7.033e+02]
 [ 0.000e+00  0.000e+00  0.000e+00  1.000e+00]]
```

Plot differences between different "adjusted" data (including quasi-definitive) and a static version of traditional HDZ baseline-adjusted data

```
In [ ]: 1 # plot differences with traditional HDZ baseline-adjusted data↔
```

Plot differences between different "adjusted" and quasi-definitive data

In [74]: 1 ▶ # plot differences↔

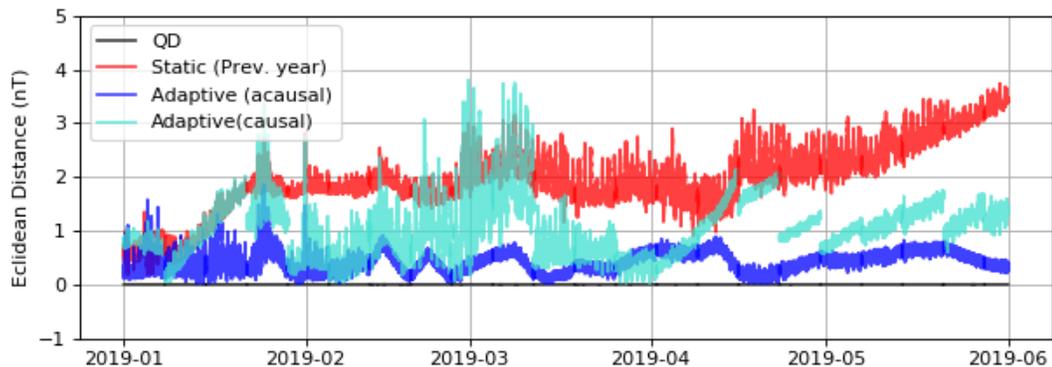


```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
warnings.warn(message, mplDeprecation, stacklevel=1)
```

Out[74]: <matplotlib.legend.Legend at 0x1c6e4ebc88>

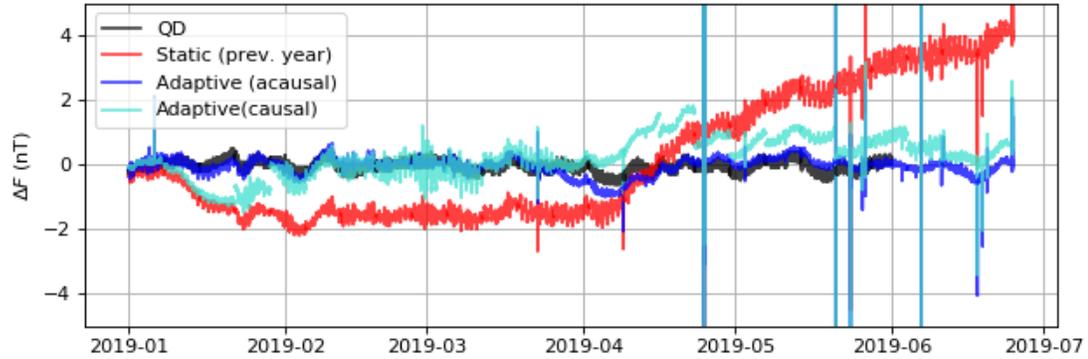
Plot Euclidean (vector) distance between different "Adjusted" and quasi-definitive data.

In [75]: 1 ▶ # Plot Euclidean distances↔



Plot  $\Delta F$  for different "adjusted" and quasi-definitive data

```
In [76]: 1 ▶ # plot delta-Fs ↔
```



## Fresno (FRN) Observatory

```
In [77]: 1 ▾ # configuration parameters for FRN
2
3 # INPUTS
4 obs_code = 'FRN'
5 start.UTC = UTCDateTime('2019-01-01T00:00:00Z')
6 end.UTC = UTCDateTime('2019-06-30T23:59:00Z')
7
8 # OPTIONS
9 update_interval = 86400 * 7
10 acausal = False
11 first.UTC = UTCDateTime('2018-10-01T00:00:00Z')
12 last.UTC = UTCDateTime('2019-07-31T23:59:00Z')
13
14 # This is slowly evolving horizontal rotation, and
15 # quickly evolving baseline offsets (including Ebase)
16 M_funcs = [generate_affine_8, generate_affine_6]
17 memories = [86400 * 100, 86400 * 10]
18
19 #path_or_url = '/Volumes/geomag/pub/Caldata/Checked Baseline Data/'
20 path_or_url = 'https://geomag.usgs.gov'
21
22 validate = True
23 edge_host = 'cwbpub.cr.usgs.gov'
```

Run do\_it\_all() with a "short" memory in causal mode

```
In [78]: 1 ▶ %%capture↔
```

Run do\_it\_all() with a "short" memory in acausal mode

```
In [79]: 1 ▶ %%capture↔
```

Run do\_it\_all() with infinite memory, but update every update\_interval

```
In [80]: 1 ▶ %%capture↔
```

Run do\_it\_all() with infinite memory, for entire interval

```
In [81]: 1 ▶ %%capture↔
```

Print the last causal M matrix, then save it to a JSON file for production runs.

```
In [82]: 1 ▶ # print and save last M matrix↔
```

```
[[ 9.556e-01 -2.945e-01 0.000e+00 2.169e+02]
 [ 2.945e-01 9.556e-01 0.000e+00 -1.738e+03]
 [ 0.000e+00 0.000e+00 1.000e+00 6.510e+01]
 [ 0.000e+00 0.000e+00 0.000e+00 1.000e+00]]
```

Plot differences between different "adjusted" data (including quasi-definitive) and a static version of traditional HDZ baseline-adjusted data

```
In [ ]: 1 ▶ # plot differences with traditional HDZ baseline-adjusted data↔
```

Plot differences between different "adjusted" and quasi-definitive data

**Note:** Quasi-definitive data not available for FRN

```
In [ ]: 1 ▶ # plot differences↔
```

Plot Euclidean (vector) distance between different "Adjusted" and quasi-definitive data.

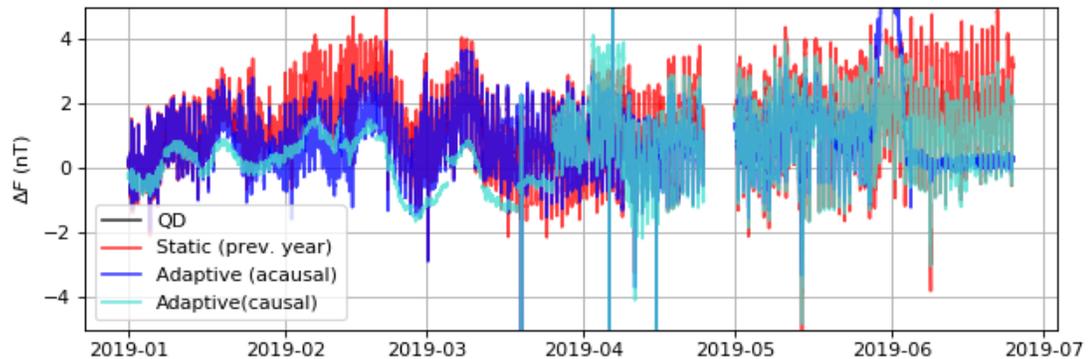
**Note:** Quasi-definitive data not available for FRN

```
In [ ]: 1 ▶ # Plot Euclidean distances↔
```

Plot  $\Delta F$  for different "adjusted" and quasi-definitive data

```
In [84]: 1 ▶ # plot delta-Fs ↔
```

```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/pyplot.py:522: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
max_open_warning, RuntimeWarning)
```



## Guam (GUA) Observatory

```
In [85]: 1 ▾ # configuration parameters for GUA
2
3 # INPUTS
4 obs_code = 'GUA'
5 start.UTC = UTCDateTime('2019-01-01T00:00:00Z')
6 end.UTC = UTCDateTime('2019-06-30T23:59:00Z')
7
8 # OPTIONS
9 update_interval = 86400 * 7
10 acausal = False
11 first.UTC = UTCDateTime('2018-10-01T00:00:00Z')
12 last.UTC = UTCDateTime('2019-07-31T23:59:00Z')
13
14 # This is slowly evolving horizontal rotation, and
15 # quickly evolving baseline offsets (including Ebase)
16 M_funcs = [generate_affine_8, generate_affine_6]
17 memories = [86400 * 100, 86400 * 10]
18
19 #path_or_url = '/Volumes/geomag/pub/Caldata/Checked Baseline Data/'
20 path_or_url = 'https://geomag.usgs.gov'
21
22 validate = True
23 edge_host = 'cwbpub.cr.usgs.gov'
```

Run do\_it\_all() with a "short" memory in causal mode

```
In [86]: 1 ▶ %%capture↔
```

Run do\_it\_all() with a "short" memory in acausal mode

In [87]: 1 ▶ `%%capture↔`

Run `do_it_all()` with infinite memory, but update every `update_interval`

In [88]: 1 ▶ `%%capture↔`

Run `do_it_all()` with infinite memory, for entire interval

In [89]: 1 ▶ `%%capture↔`

Print the last causal M matrix, then save it to a JSON file for production runs.

In [90]: 1 ▶ `# print and save last M matrix↔`

```
[[ 9.998e-01 -2.221e-02  0.000e+00  5.761e+02]
 [ 2.221e-02  9.998e-01  0.000e+00 -8.021e+01]
 [ 0.000e+00  0.000e+00  1.000e+00  2.522e+02]
 [ 0.000e+00  0.000e+00  0.000e+00  1.000e+00]]
```

Plot differences between different "adjusted" data (including quasi-definitive) and a static version of traditional HDZ baseline-adjusted data

In [ ]: 1 ▶ `# plot differences with traditional HDZ baseline-adjusted data↔`

Plot differences between different "adjusted" and quasi-definitive data

**Note:** Quasi-definitive data not available for GUA

In [ ]: 1 ▶ `# plot differences↔`

Plot Euclidean (vector) distance between different "Adjusted" and quasi-definitive data.

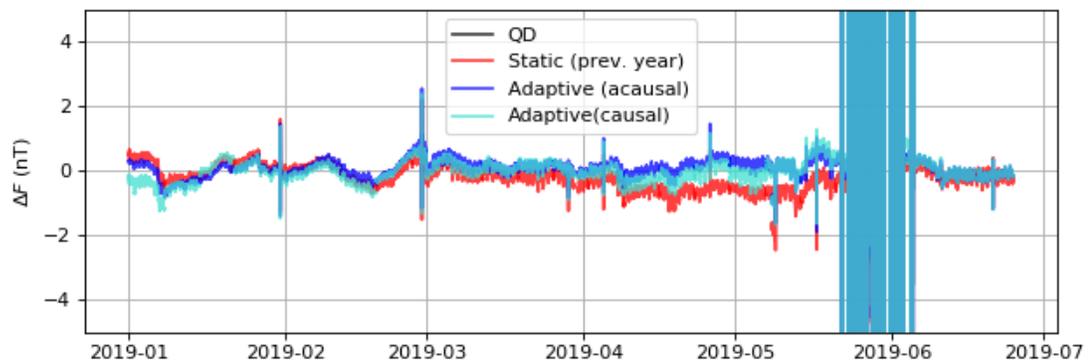
**Note:** Quasi-definitive data not available for GUA

In [ ]: 1 ▶ `# Plot Euclidean distances↔`

Plot  $\Delta F$  for different "adjusted" and quasi-definitive data

In [92]: 1 ▶ `# plot delta-Fs ↔`

```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/pyplot.py:522: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```



## [Honolulu \(HON\) Observatory](#)

```
In [93]: 1 # configuration parameters for HON
2
3 # INPUTS
4 obs_code = 'HON'
5 start.UTC = UTCDateTime('2019-01-01T00:00:00Z')
6 end.UTC = UTCDateTime('2019-06-30T23:59:00Z')
7
8 # OPTIONS
9 update_interval = 86400 * 7
10 acausal = False
11 first.UTC = UTCDateTime('2018-10-01T00:00:00Z')
12 last.UTC = UTCDateTime('2019-07-31T23:59:00Z')
13
14 # This is slowly evolving horizontal rotation, and
15 # quickly evolving baseline offsets (including Ebase)
16 M_funcs = [generate_affine_8, generate_affine_6]
17 memories = [86400 * 100, 86400 * 10]
18
19 #path_or_url = '/Volumes/geomag/pub/Caldata/Checked Baseline Data/'
20 path_or_url = 'https://geomag.usgs.gov'
21
22 validate = True
23 edge_host = 'cwbpublish.cr.usgs.gov'
```

Run do\_it\_all() with a "short" memory in causal mode

```
In [94]: 1 %%capture↔
```

Run do\_it\_all() with a "short" memory in acausal mode

```
In [95]: 1 %%capture↔
```

Run do\_it\_all() with infinite memory, but update every update\_interval

```
In [96]: 1 %%capture↔
```

Run do\_it\_all() with infinite memory, for entire interval

```
In [97]: 1 %%capture↔
```

Print the last causal M matrix, then save it to a JSON file for production runs.

```
In [98]: 1 # print and save last M matrix↔

[[ 0.986 -0.168  0.   -45.084]
 [ 0.168  0.986  0.   100.831]
 [ 0.     0.     1.   127.731]
 [ 0.     0.     0.     1.   ]]
```

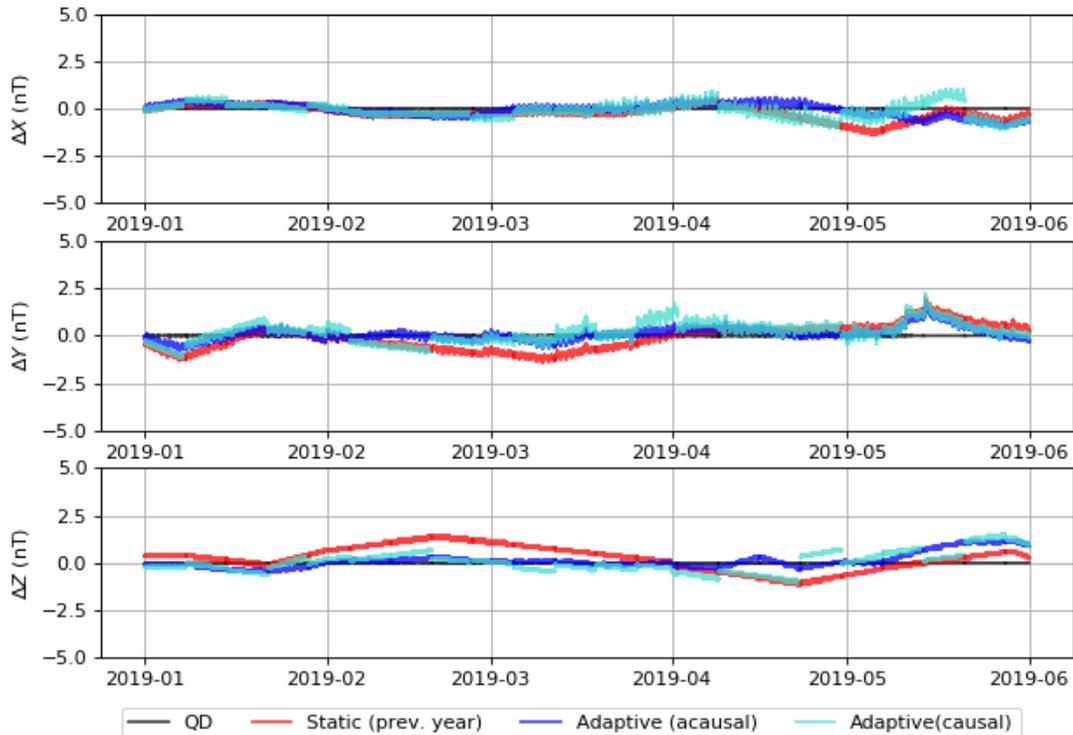
Plot differences between different "adjusted" data (including quasi-definitive) and a static version of traditional HDZ baseline-adjusted data

```
In [ ]: 1 # plot differences with traditional HDZ baseline-adjusted data↔
```

Plot differences between different "adjusted" and quasi-definitive data

In [99]: 1 ▶ # plot differences↔

```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/pyplot.py:522: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
max_open_warning, RuntimeWarning)
```



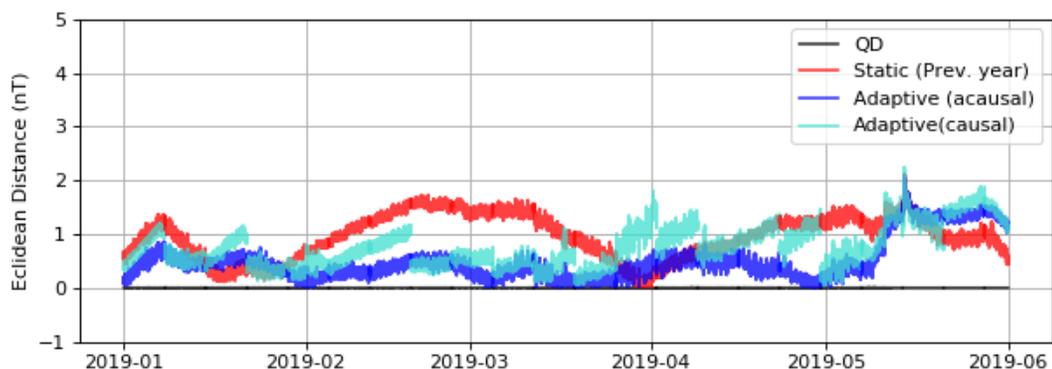
```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
warnings.warn(message, mplDeprecation, stacklevel=1)
```

Out[99]: <matplotlib.legend.Legend at 0x1cb8f7f9b0>

Plot Euclidean (vector) distance between different "Adjusted" and quasi-definitive data.

In [100]: 1 ▶ # Plot Euclidean distances↔

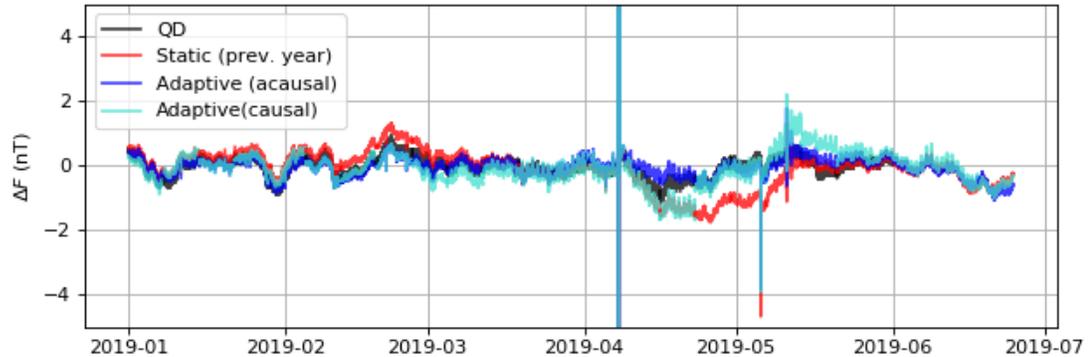
```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/pyplot.py:522: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
max_open_warning, RuntimeWarning)
```



Plot  $\Delta F$  for different "adjusted" and quasi-definitive data

```
In [101]: 1 ▶ # plot delta-Fs ↔
```

```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/pyplot.py:522: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
max_open_warning, RuntimeWarning)
```



## Newport (NEW) Observatory

```
In [102]: 1 ▾ # configuration parameters for NEW
2
3 # INPUTS
4 obs_code = 'NEW'
5 start.UTC = UTCDateTime('2019-01-01T00:00:00Z')
6 end.UTC = UTCDateTime('2019-06-30T23:59:00Z')
7
8 # OPTIONS
9 update_interval = 86400 * 7
10 acausal = False
11 first.UTC = UTCDateTime('2018-10-01T00:00:00Z')
12 last.UTC = UTCDateTime('2019-07-31T23:59:00Z')
13
14 # This is slowly evolving horizontal rotation, and
15 # quickly evolving baseline offsets (including Ebase)
16 M_funcs = [generate_affine_8, generate_affine_6]
17 memories = [86400 * 100, 86400 * 10]
18
19 #path_or_url = '/Volumes/geomag/pub/Caldata/Checked Baseline Data/'
20 path_or_url = 'https://geomag.usgs.gov'
21
22 validate = True
23 edge_host = 'cwbpub.cr.usgs.gov'
```

Run `do_it_all()` with a "short" memory in causal mode

```
In [103]: 1 ▶ %%capture↔
```

Run `do_it_all()` with a "short" memory in acausal mode

```
In [104]: 1 ▶ %%capture↔
```

Run `do_it_all()` with infinite memory, but update every `update_interval`

```
In [105]: 1 ▶ %%capture↔
```

Run `do_it_all()` with infinite memory, for entire interval

```
In [106]: 1 ▶ %%capture↔
```

Print the last causal M matrix, then save it to a JSON file for production runs.

```
In [107]: 1 ▶ # print and save last M matrix↔
```

```
[[ 9.644e-01 -2.646e-01 0.000e+00 7.591e+01]
 [ 2.646e-01 9.644e-01 0.000e+00 1.436e+01]
 [ 0.000e+00 0.000e+00 1.000e+00 7.429e+02]
 [ 0.000e+00 0.000e+00 0.000e+00 1.000e+00]]
```

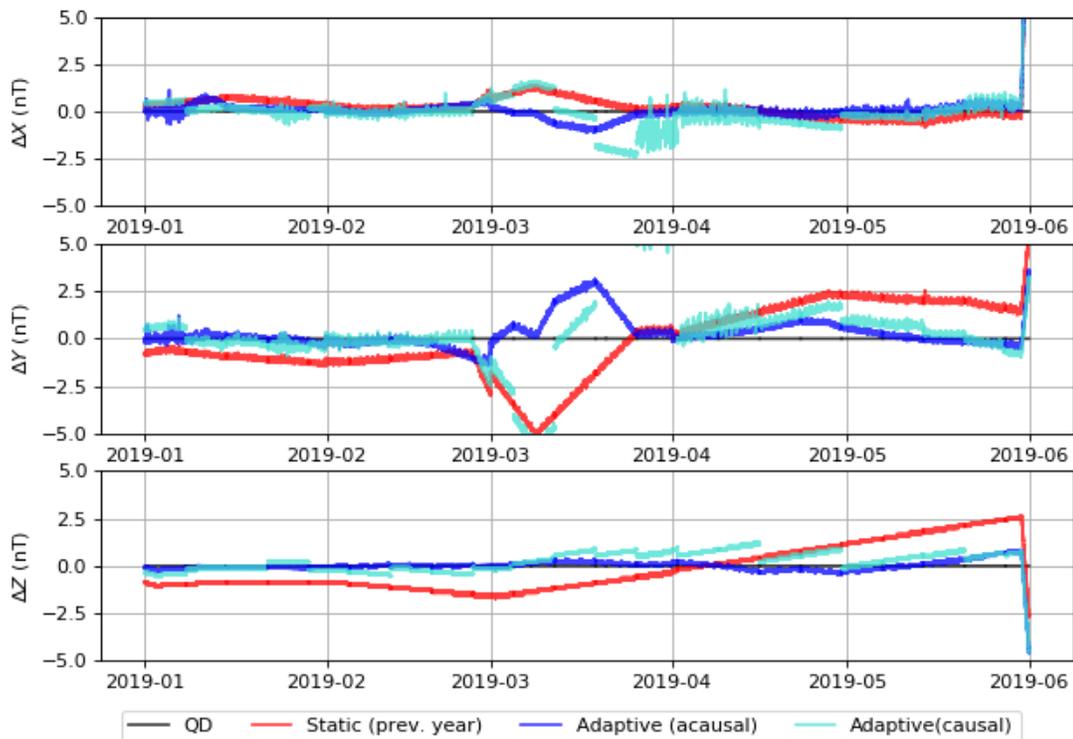
Plot differences between different "adjusted" data (including quasi-definitive) and a static version of traditional HDZ baseline-adjusted data

```
In [ ]: 1 ▶ # plot differences with traditional HDZ baseline-adjusted data↔
```

Plot differences between different "adjusted" and quasi-definitive data

```
In [109]: 1 ▶ # plot differences↔
```

```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/pyplot.py:522: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```



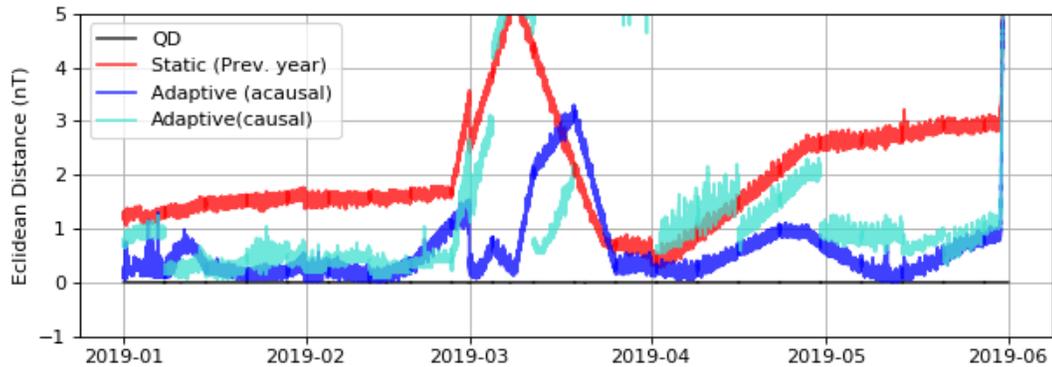
```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
  warnings.warn(message, mplDeprecation, stacklevel=1)
```

```
Out[109]: <matplotlib.legend.Legend at 0x1cccc2dd8>
```

Plot Euclidean (vector) distance between different "Adjusted" and quasi-definitive data.

In [110]: 1 ▶ # Plot Euclidean distances↔

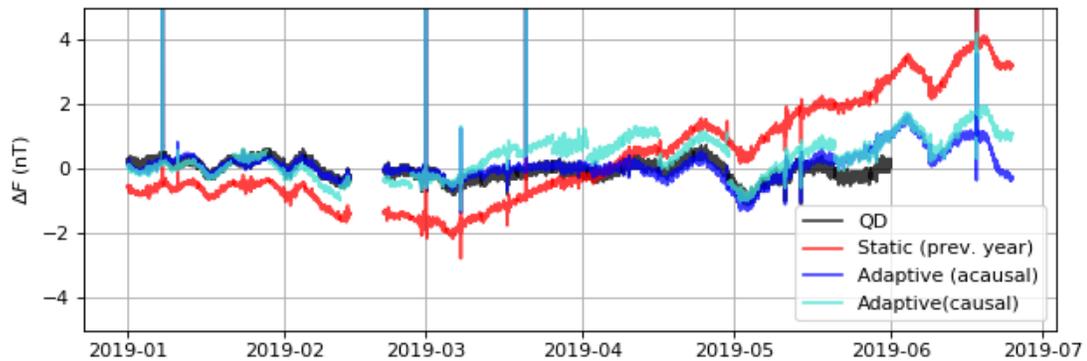
```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/pyplot.py:522: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
max_open_warning, RuntimeWarning)
```



Plot  $\Delta F$  for different "adjusted" and quasi-definitive data

In [111]: 1 ▶ # plot delta-Fs ↔

```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/pyplot.py:522: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
max_open_warning, RuntimeWarning)
```



## [San Juan \(SJG\) Observatory](#)

```
In [112]: 1 ▾ # configuration parameters for SJG
2
3 # INPUTS
4 obs_code = 'SJG'
5 start.UTC = UTCDateTime('2019-01-01T00:00:00Z')
6 end.UTC = UTCDateTime('2019-06-30T23:59:00Z')
7
8 # OPTIONS
9 update_interval = 86400 * 7
10 acausal = False
11 first.UTC = UTCDateTime('2018-10-01T00:00:00Z')
12 last.UTC = UTCDateTime('2019-07-31T23:59:00Z')
13
14 # This is slowly evolving horizontal rotation, and
15 # quickly evolving baseline offsets (including Ebase)
16 M_funcs = [generate_affine_8, generate_affine_6]
17 memories = [86400 * 100, 86400 * 10]
18
19 #path_or_url = '/Volumes/geomag/pub/Caldata/Checked Baseline Data/'
20 path_or_url = 'https://geomag.usgs.gov'
21
22 validate = True
23 edge_host = 'cwbpub.cr.usgs.gov'
```

Run do\_it\_all() with a "short" memory in causal mode

```
In [113]: 1 ▸ %%capture↔
```

Run do\_it\_all() with a "short" memory in acausal mode

```
In [114]: 1 ▸ %%capture↔
```

Run do\_it\_all() with infinite memory, but update every update\_interval

```
In [115]: 1 ▸ %%capture↔
```

Run do\_it\_all() with infinite memory, for entire interval

```
In [116]: 1 ▸ %%capture↔
```

Print the last causal M matrix, then save it to a JSON file for production runs.

```
In [117]: 1 ▸ # print and save last M matrix↔

[[ 9.787e-01  2.053e-01  0.000e+00  1.847e+01]
 [-2.053e-01  9.787e-01  0.000e+00 -4.315e+02]
 [ 0.000e+00  0.000e+00  1.000e+00  2.691e+02]
 [ 0.000e+00  0.000e+00  0.000e+00  1.000e+00]]
```

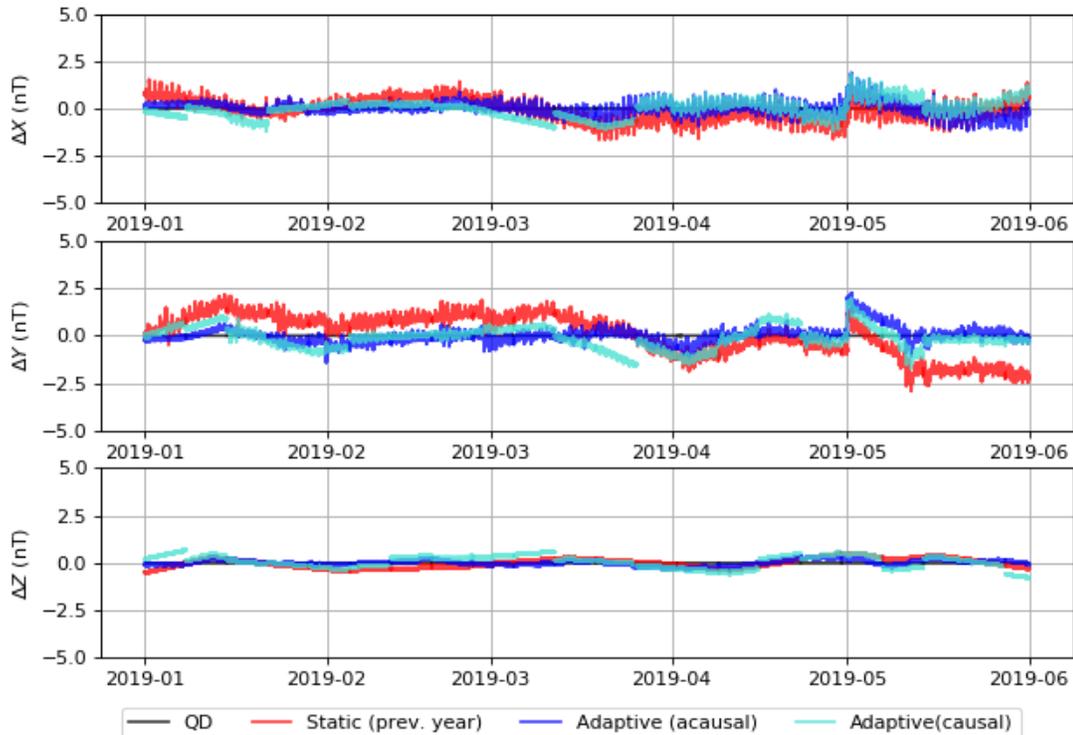
Plot differences between different "adjusted" data (including quasi-definitive) and a static version of traditional HDZ baseline-adjusted data

```
In [ ]: 1 ▸ # plot differences with traditional HDZ baseline-adjusted data↔
```

Plot differences between different "adjusted" and quasi-definitive data

In [119]: 1 ▶ # plot differences↔

```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/pyplot.py:522: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
max_open_warning, RuntimeWarning)
```



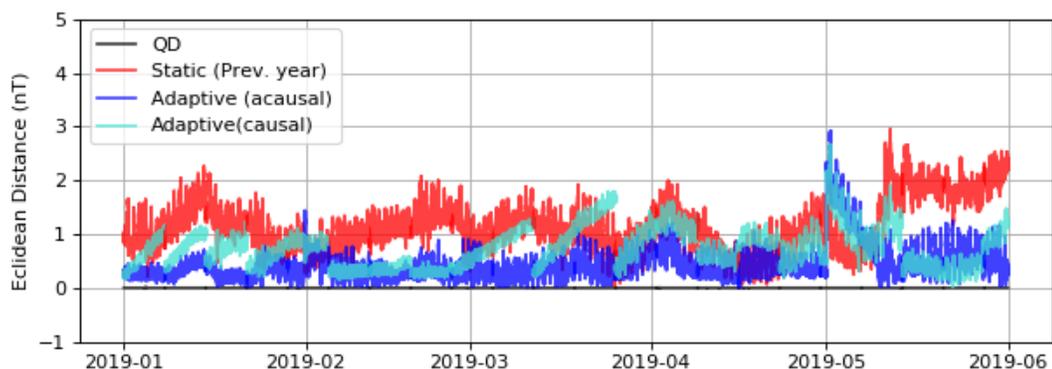
```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
warnings.warn(message, mplDeprecation, stacklevel=1)
```

Out[119]: <matplotlib.legend.Legend at 0x1cbaef7710>

Plot Euclidean (vector) distance between different "Adjusted" and quasi-definitive data.

In [120]: 1 ▶ # Plot Euclidean distances↔

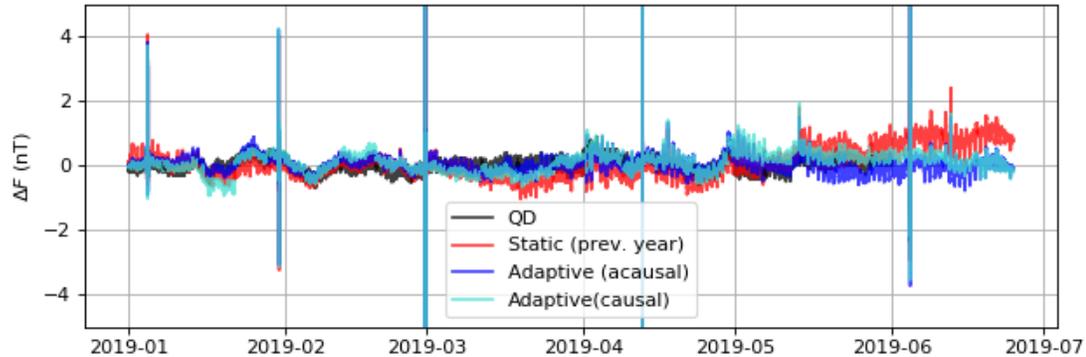
```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/pyplot.py:522: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
max_open_warning, RuntimeWarning)
```



Plot  $\Delta F$  for different "adjusted" and quasi-definitive data

```
In [121]: 1 | # plot delta-Fs ↔
```

```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/pyplot.py:522: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
max_open_warning, RuntimeWarning)
```



## Shumagin (SHU) Observatory

**Note:** SHU is kind of a mess. There were known instrumentation issues in the summer of 2018, but it was thought these were resolved. Regardless, no data from this period was used here. Things seem to settle down by the end of the validation interval (~June 2019), so the final affine transform matrix is probably good.

```
In [122]: 1 | # configuration parameters for SHU
2
3 | # INPUTS
4 | obs_code = 'SHU'
5 | start_utc = UTCDateTime('2019-01-01T00:00:00Z')
6 | end_utc = UTCDateTime('2019-06-30T23:59:00Z')
7
8 | # OPTIONS
9 | update_interval = 86400 * 7
10 | acausal = False
11 | first_utc = UTCDateTime('2018-10-01T00:00:00Z')
12 | last_utc = UTCDateTime('2019-07-31T23:59:00Z')
13
14 | # This is slowly evolving horizontal rotation, and
15 | # quickly evolving baseline offsets (including Ebase)
16 | M_funcs = [generate_affine_8, generate_affine_6]
17 | memories = [86400 * 100, 86400 * 10]
18
19 | #path_or_url = '/Volumes/geomag/pub/Caldata/Checked Baseline Data/'
20 | path_or_url = 'https://geomag.usgs.gov'
21
22 | validate = True
23 | edge_host = 'cwbpub.cr.usgs.gov'
```

Run `do_it_all()` with a "short" memory in causal mode

```
In [123]: 1 | %%capture↔
```

Run `do_it_all()` with a "short" memory in acausal mode

```
In [124]: 1 | %%capture↔
```

Run `do_it_all()` with infinite memory, but update every `update_interval`

```
In [125]: 1 | %%capture↔
```

Run `do_it_all()` with infinite memory, for entire interval

```
In [126]: 1 | %%capture↔
```

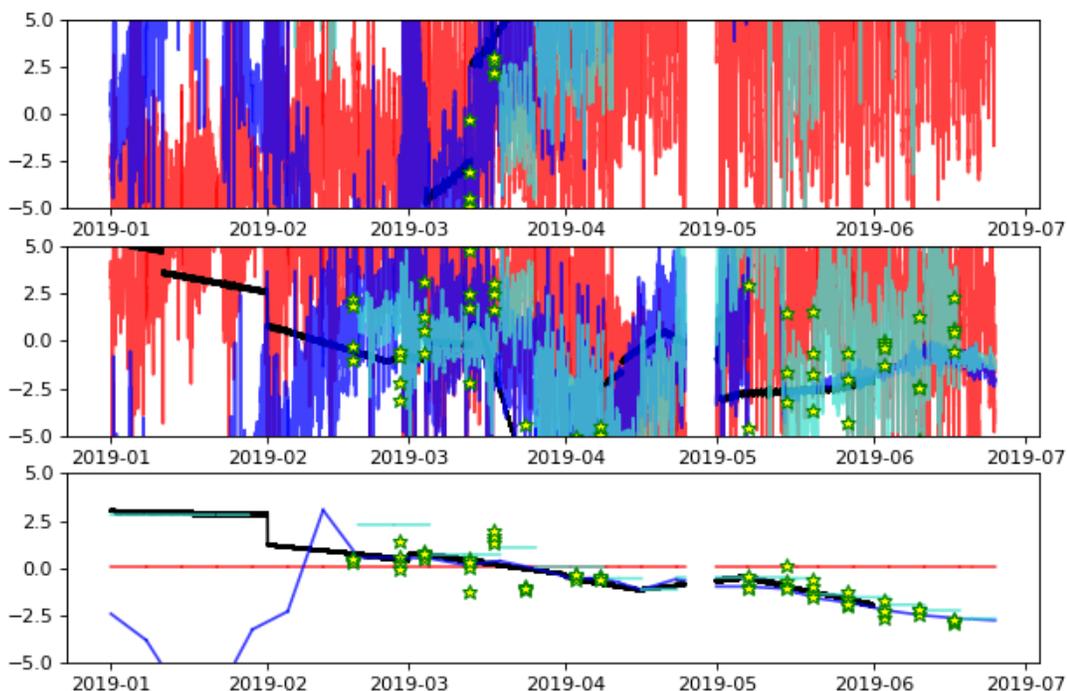
Print the last causal M matrix, then save it to a JSON file for production runs.

```
In [127]: 1 ▶ # print and save last M matrix↔
[[ 9.759e-01 -2.184e-01  0.000e+00 -4.665e+01]
 [ 2.184e-01  9.759e-01  0.000e+00 -5.371e+02]
 [ 0.000e+00  0.000e+00  1.000e+00 -3.283e+02]
 [ 0.000e+00  0.000e+00  0.000e+00  1.000e+00]]
```

Plot differences between different "adjusted" data (including quasi-definitive) and a static version of traditional HDZ baseline-adjusted data

```
In [128]: 1 ▶ # plot differences with traditional HDZ baseline-adjusted data↔

/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/pyplot.py:522: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```



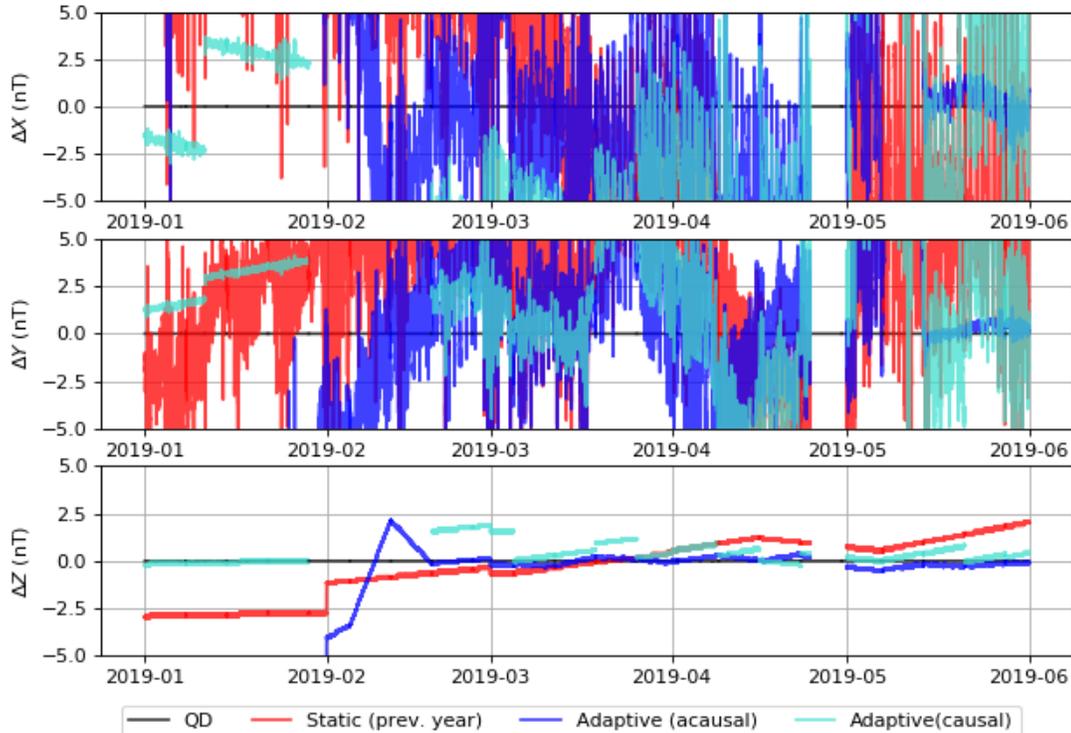
```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
  warnings.warn(message, mplDeprecation, stacklevel=1)
```

```
Out[128]: [<matplotlib.lines.Line2D at 0x1c6dd4e048>]
```

Plot differences between different "adjusted" and quasi-definitive data

In [129]: 1 ▶ # plot differences↔

```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/pyplot.py:522: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
max_open_warning, RuntimeWarning)
```



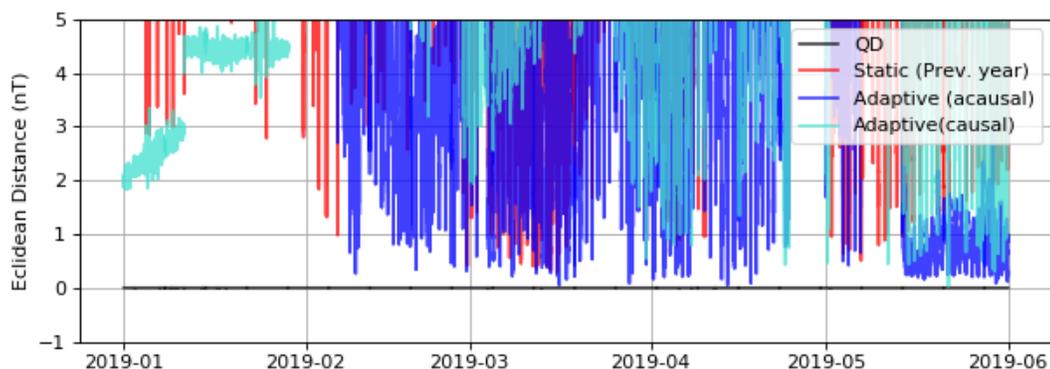
```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
warnings.warn(message, mplDeprecation, stacklevel=1)
```

Out[129]: <matplotlib.legend.Legend at 0x1cd3094f60>

Plot Euclidean (vector) distance between different "Adjusted" and quasi-definitive data.

In [130]: 1 ▶ # Plot Euclidean distances↔

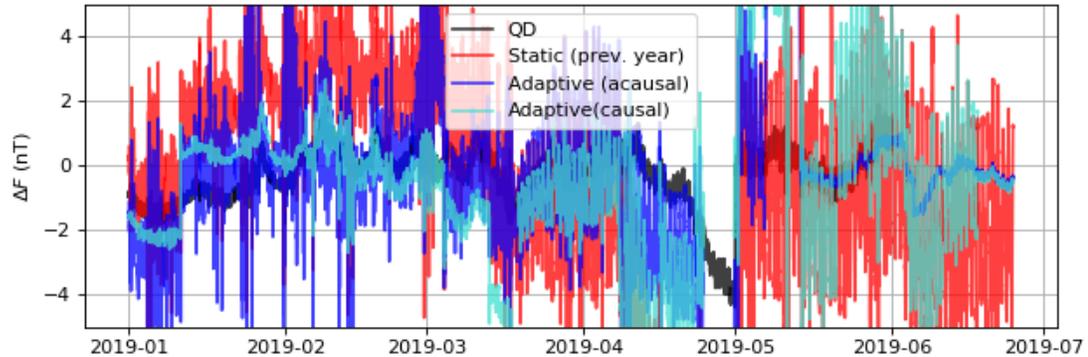
```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/pyplot.py:522: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
max_open_warning, RuntimeWarning)
```



Plot  $\Delta F$  for different "adjusted" and quasi-definitive data

```
In [131]: 1 ▶ # plot delta-Fs ↔
```

```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/pyplot.py:522: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```



## Sitka (SIT) Observatory

```
In [132]: 1 ▼ # configuration parameters for SIT
2
3 # INPUTS
4 obs_code = 'SIT'
5 start.UTC = UTCDateTime('2019-01-01T00:00:00Z')
6 end.UTC = UTCDateTime('2019-06-30T23:59:00Z')
7
8 # OPTIONS
9 update_interval = 86400 * 7
10 acausal = False
11 first.UTC = UTCDateTime('2018-10-01T00:00:00Z')
12 last.UTC = UTCDateTime('2019-07-31T23:59:00Z')
13
14 # This is slowly evolving horizontal rotation, and
15 # quickly evolving baseline offsets (including Ebase)
16 M_funcs = [generate_affine_8, generate_affine_6]
17 memories = [86400 * 100, 86400 * 10]
18
19 #path_or_url = '/Volumes/geomag/pub/Caldata/Checked Baseline Data/'
20 path_or_url = 'https://geomag.usgs.gov'
21
22 validate = True
23 edge_host = 'cwbpub.cr.usgs.gov'
```

Run do\_it\_all() with a "short" memory in causal mode

```
In [133]: 1 ▶ %%capture↔
```

Run do\_it\_all() with a "short" memory in acausal mode

```
In [134]: 1 ▶ %%capture↔
```

Run do\_it\_all() with infinite memory, but update every update\_interval

```
In [135]: 1 ▶ %%capture↔
```

Run do\_it\_all() with infinite memory, for entire interval

```
In [136]: 1 ▶ %%capture↔
```

Print the last causal M matrix, then save it to a JSON file for production runs.

```
In [137]: 1 ▶ # print and save last M matrix↔
```

```
[[ 0.941  -0.339   0.    147.305]
 [ 0.339   0.941   0.   -226.881]
 [ 0.      0.      1.    74.17 ]
 [ 0.      0.      0.     1.   ]]
```

Plot differences between different "adjusted" data (including quasi-definitive) and a static version of traditional HDZ baseline-adjusted data

```
In [ ]: 1 ▶ # plot differences with traditional HDZ baseline-adjusted data↔
```

Plot differences between different "adjusted" and quasi-definitive data

**Note:** Quasi-definitive data not available for SIT

```
In [ ]: 1 ▶ # plot differences↔
```

Plot Euclidean (vector) distance between different "Adjusted" and quasi-definitive data.

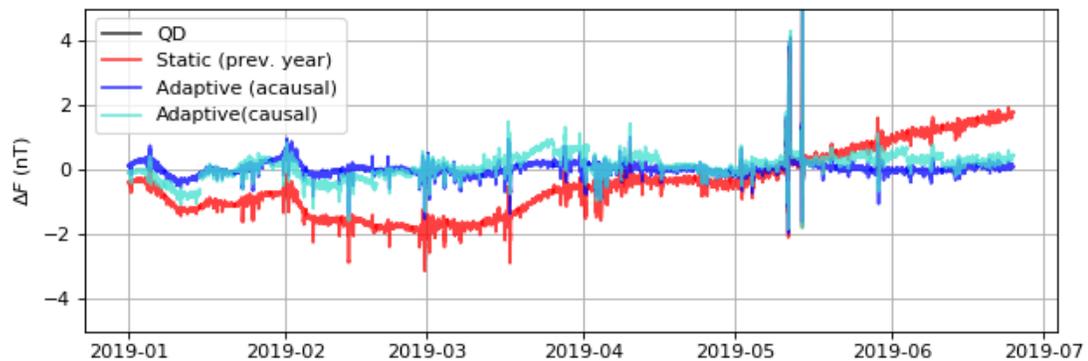
**Note:** Quasi-definitive data not available for SIT

```
In [ ]: 1 ▶ # Plot Euclidean distances↔
```

Plot  $\Delta F$  for different "adjusted" and quasi-definitive data

```
In [139]: 1 ▶ # plot delta-Fs ↔
```

```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/pyplot.py:522: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```



## [Tucson \(TUC\) Observatory](#)

```
In [140]: 1 # configuration parameters for TUC
2
3 # INPUTS
4 obs_code = 'TUC'
5 start.UTC = UTCDateTime('2019-01-01T00:00:00Z')
6 end.UTC = UTCDateTime('2019-06-30T23:59:00Z')
7
8 # OPTIONS
9 update_interval = 86400 * 7
10 acausal = False
11 first.UTC = UTCDateTime('2018-10-01T00:00:00Z')
12 last.UTC = UTCDateTime('2019-07-31T23:59:00Z')
13
14 # This is slowly evolving horizontal rotation, and
15 # quickly evolving baseline offsets (including Ebase)
16 M_funcs = [generate_affine_8, generate_affine_6]
17 memories = [86400 * 100, 86400 * 10]
18
19 #path_or_url = '/Volumes/geomag/pub/Caldata/Checked Baseline Data/'
20 path_or_url = 'https://geomag.usgs.gov'
21
22 validate = True
23 edge_host = 'cwbpub.cr.usgs.gov'
```

Run do\_it\_all() with a "short" memory in causal mode

```
In [141]: 1 %%capture↔
```

Run do\_it\_all() with a "short" memory in acausal mode

```
In [142]: 1 %%capture↔
```

Run do\_it\_all() with infinite memory, but update every update\_interval

```
In [143]: 1 %%capture↔
```

Run do\_it\_all() with infinite memory, for entire interval

```
In [144]: 1 %%capture↔
```

Print the last causal M matrix, then save it to a JSON file for production runs.

```
In [145]: 1 # print and save last M matrix↔

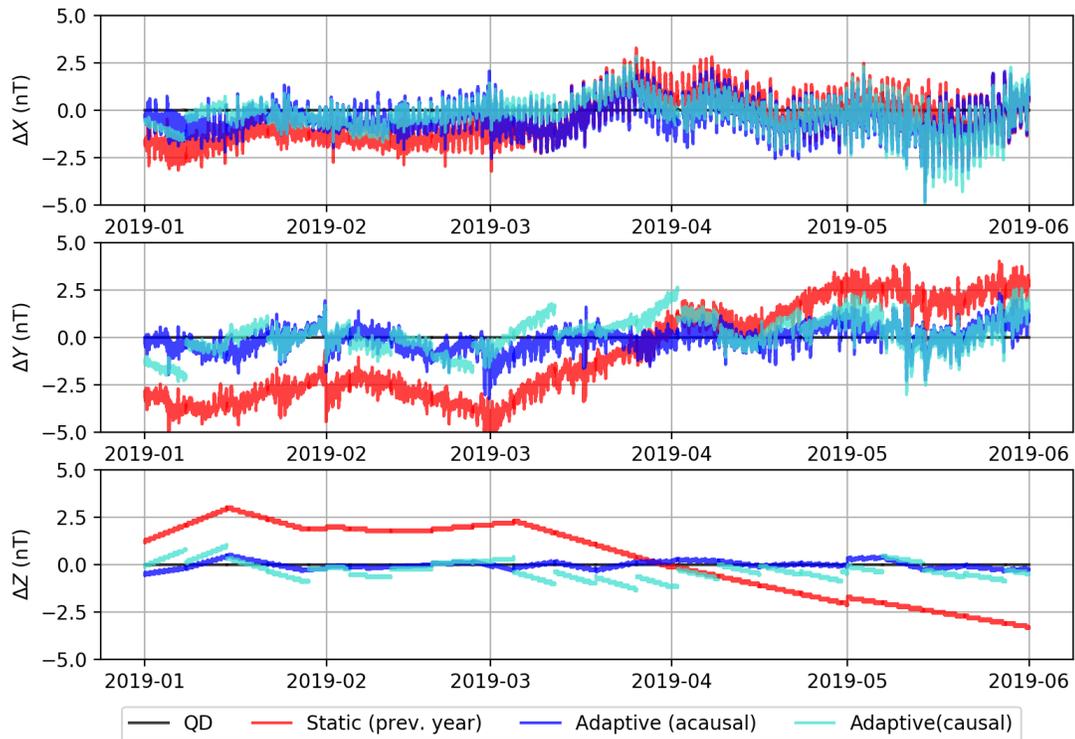
[[ 9.795e-01 -2.014e-01 0.000e+00 -6.626e+01]
 [ 2.014e-01 9.795e-01 0.000e+00 -9.680e+02]
 [ 0.000e+00 0.000e+00 1.000e+00 3.506e+02]
 [ 0.000e+00 0.000e+00 0.000e+00 1.000e+00]]
```

Plot differences between different "adjusted" data (including quasi-definitive) and a static version of traditional HDZ baseline-adjusted data

```
In [ ]: 1 # plot differences with traditional HDZ baseline-adjusted data↔
```

Plot differences between different "adjusted" and quasi-definitive data

In [82]: 1 ▶ # plot differences↔

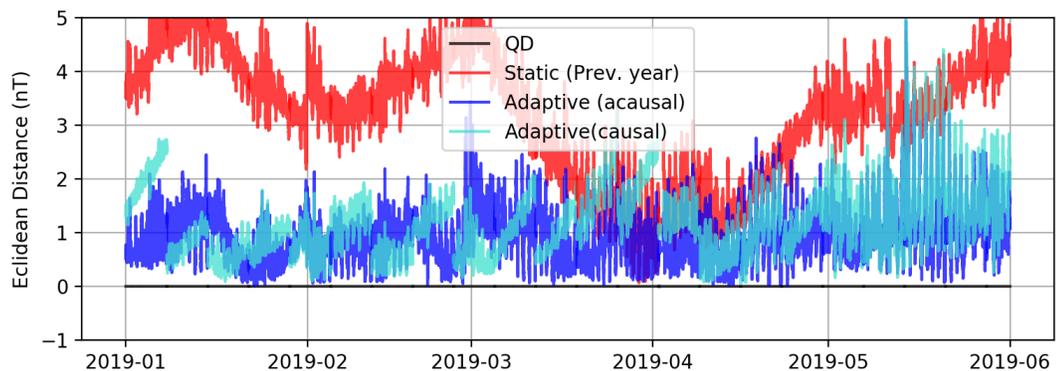


```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
warnings.warn(message, mplDeprecation, stacklevel=1)
```

Out[82]: <matplotlib.legend.Legend at 0x1cb2fe9828>

Plot Euclidean (vector) distance between different "Adjusted" and quasi-definitive data.

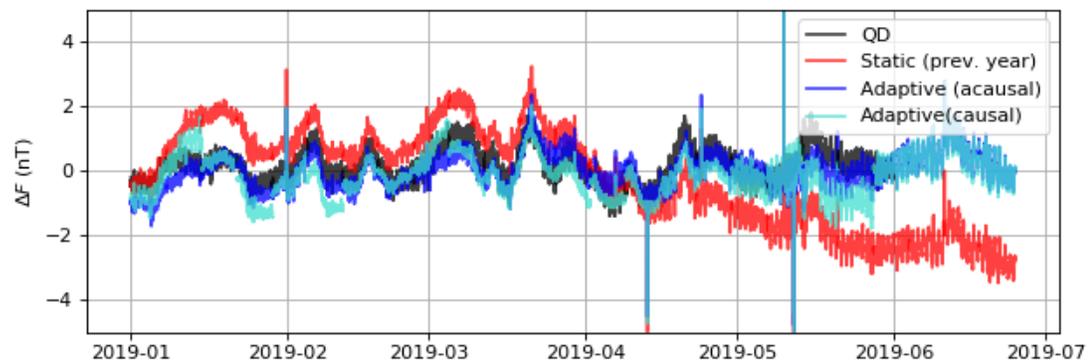
In [83]: 1 ▶ # Plot Euclidean distances↔



Plot  $\Delta F$  for different "adjusted" and quasi-definitive data

```
In [147]: 1 ▶ # plot delta-Fs ↔
```

```
/Users/erigler/anaconda3/envs/test_GIMP_py36/lib/python3.6/site-packages/matplotlib/pyplot.py:522: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).  
max_open_warning, RuntimeWarning)
```



```
In [ ]: 1
```

```
In [ ]: 1
```

```
In [ ]: 1
```