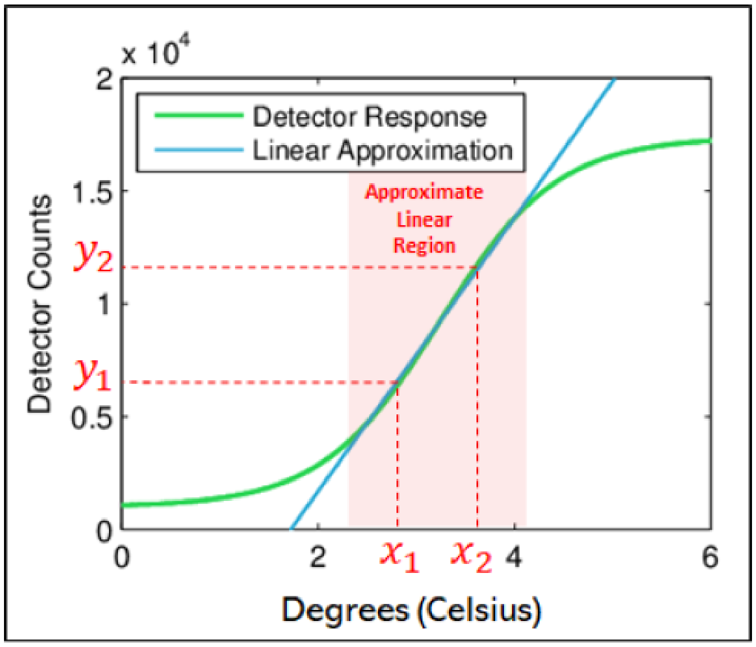
Image Restoration

Complete each of the following tasks.

Task 1: Infrared Focal Plane Array Nonuniformity Correction (NUC). Infrared focal plane array sensors are known to suffer from pixel-to-pixel response nonuniformity and manifests in the imagery as fixed pattern spatial noise. While the response of individual pixels' drifts with time, they can be assumed to be constant within a reasonably short period of time (minutes to hours depending on the sensor). A common approach to removing fixed pattern noise is to perform a calibration procedure that involves placing a uniform blackbody radiation source in front of the camera system at a known radiance level. The idea is that we expose each pixel to the same level of input radiation. If each pixel has a uniform response, we would expect each pixel to measure the same value. This is of course not the case, but if we measure several different flat-field images at different controlled radiance levels we can fit a curve to the measured pixel response data and effectively invert out each individual pixel response to ensure that they all indeed respond the same.

Although pixel response curves are often nonlinear, it is common in practice to assume that the response curves are linear. This assumption is reasonable because most of the time the pixels are fabricated to have an approximately linear response over the average expected operational range of the camera. For such a linear fit, we simply need two at-field image measurements (often called a 2-point correction) taken within the linear region of pixel responses. The figure below illustrates this concept for a single pixel response. Thus, x1 and x2 correspond to the input radiation level (in this case in degrees Celcius for a thermal infrared camera). The corresponding pixel measurements are y1 and y2, respectively, and are in units of detector counts. Detector counts are simply the raw fixed point integer values that are output by the analog-to-digital read-out circuitry of the camera system.



Under this linear model, notice that the two flat field measurements are of the form

$$y_1(m,n) = a(m,n)x_1 + b(m,n) \qquad y_2(m,n) = a(m,n)x_2 + b(m,n).$$

Thus, our goal in performing a two-point calibration is to determine estimates for the gain, $a(m,n)$, and bias, $b(m,n)$ terms. Since we know the input temperatures, x_1 and x_2, and the corresponding measurements at each pixel $y_1(m,n)$ and $y_2(m,n)$, we can solve for each gain and bias term that characterize our linear fit to a given pixel response according to:

$$a(m,n) = \frac{y_2(m,n) - y_1(m,n)}{x_2 - x_1} \qquad b(m,n) = y_1(m,n) - a(m,n)x_1.$$

Once we have estimated the gain and bias for each pixel we can apply them to correct for nonuniformity in each pixel in an arbitrary image, i.e.,

$$\hat{x}(m,n) = \frac{y(m,n) - b(m,n)}{a(m,n)}.$$

After nonuniformity correction, assuming our linear model is a reasonable assumption, the image should be free of fixed pattern noise. Moreover, the corrected image $\hat{x}$ will also be in units of Celcius, and hence we have calibrated our image into physically meaningful radiometric units.

For this task, I have provided 7 flat-field image sequence (as .mat files). Each flat-field data set is of size 128×128 and consists of 192 frames. The are images of a flat-field blackbody source at temperatures ranging from $[18,\ 30]^\circ$ Celsius in 2-degree increments (as indicated in the filenames). 192 frames were obtained so that they can be averaged in time to reduce the effects of temporal noise. I have also provided a sequence of images entited "irscene.mat" that contains 20 image frames of the camera panning across an outdoor scene. Complete the following tasks:

1. Load the 18° and 30° flat field data files and average each of them in time to obtain two corresponding 128×128 images, $\hat{x}_{18}$ and $\hat{x}_{30}$.
2. Use $\hat{x}_{18}$ and $\hat{x}_{30}$ to perform a two-point calibration and obtain estimates of the gain and bias values at each pixel.
3. Load the IR scene data and apply the gain and bias values to perform a nonuniformity correction on the first frame of the data.
4. Display the averaged flat-field images $\hat{x}_{18}$ and $\hat{x}_{30}$.
5. Display the estimated gain and bias images.
6. Display the corrected IR scene image.
7. Submit the displayed images.

Be sure that you properly scale each image prior to displaying them (this is where the `imgscale`() function that you wrote earlier in the semester comes in handy). Some images may look fine with max-min scaling, others may look better with with statistical or absolute scaling. Absolute scaling can be particularly useful when displaying data that is in calibrated radiometric units since you can now ascribe meaning to the scaling range (units of Celsius in this case).

It is also worth noting that if you would like to calibrate your images into a different physical unit (such as Kelvin, Fahrenheit, or radiance) you can easily do so by simply converting x_1 and x_2 from Celcius into the corresponding values of the desired units prior to computing the two-point calibration.

Task 2: Sensor Noise Characterization. Flat-field imagery is useful for characterizing noise performance of an imaging sensor. In many cases, it is often desirable to characterize spatial noise and temporal noise independently. To perform some basic characterization, load each of the flat-field image sequences and perform the following tasks:

1. **Average Detector Response:** let's examine what the average detector response looks like. To do so, simply compute the average of each flat-field image across all dimensions. An easy was to do this is `mux` = `mean(ffx(:))`. This will result in a single value for each flat-field data set. Organize these average values as a vector in order of increasing flat-field temperature, i.e., `mu` = `[mu18 mu20 ... mu30]`. Then, create a corresponding temperature vector `temp` = `[18 : 2 : 30]`. We will plot these results in a later step.
2. **Spatial Noise Characterization:** It is often useful to quantify the amount of spatial noise in our sensor. One simple way to do this is to first average each flat-field image in time (to remove the temporal component). Then, simply compute the standard deviation of each resulting average flat-field image. This will give us a measurement of how much variability there is across the image. Since we know the scene is flat, we can attribute this variability directly to spatial noise. This can be easily done using the `std2`() function, i.e., `sdsx` = `std2(mean(ffx, 3))`. Organize these standard deviation values as a vector `sds` = `[sds18 sds20 ... sds30]`.
3. **Temporal Noise Characterization:** We can do a similar exercise to isolate the temporal component of the noise for characterization. In this case, we want to average out the spatial components of the noise while preserving the temporal dimension. To do this, for each flat field sequence, we average across each individual image in the flat-field sequence. This will result in a 1D vector along the time dimension. Then, we simply compute the standard deviation of this vector to get an estimate of the average temporal noise. Thus, `sdtx` = `std(mean(mean(ffx, 1), 2))`. Organize these standard deviation values in as a vector `sdt` = `[sdt18 sdt20 ... sdt30]`.

4. Now, as a simple visualization of these results, plot the average detector response, `mu` as a function of the temperature vector `temp` in black. Next, on the same plot, plot `mu + sds` and `mu − sds` as a function of `temp` in green. This will provide "error bars" about the average detector response that indicate how much variability is present in our detector response curves due to spatial noise. Finally, plot `mu + sdt` and `mu − sdt` as a function of `temp` in red. This will similarly provide error bars about the average detector response that indicate how much variability is present in our detector response curves due to temporal noise. Submit this single plot in your assignment.

There are many other ways to perform more detailed characterization of focal plane array noise performance, however this approach provides a quick look that gives good insight to the spatial and temporal noise characteristics as a function of input radiance. We often repeat this exercise after applying a two-point calibration to the flat-field data. Then, we can assess how much spatial noise remains after calibration and also perform other radiometric characterizations. As a couple examples, we examine calibration error by performing a statistical assessment to determine how accurately each flat-field image has been calibrated to the actual flat-field temperature (nonlinearity and noise will cause deviations). As another example, for infrared systems, we may want to determine the minimum change in temperature that we can reliably detect (called net-equivalent temperature difference, or NETD and noise plays an important role in that calculation as well).

Task 3: Resolution Enhancement. As we discussed in lecture, resolution enhancement algorithms (also known as super-resolution or micro-scanning algorithms) work to estimate a high resolution image from a sequence of lower resolution images. It is important to distinguish these types of algorithms from interpolation. Interpolation increases the effective sampling rate of an image, but it does not change the frequency content, i.e., no new information is added. Resolution enhancement techniques do attempt to estimate high frequency information by attempting to "unravel" aliased information within the data, i.e., aliased information is higher frequency information that has rolled over into lower resolution frequencies due to an insufficient sampling rate. Thus, if done properly, resolution enhancement can indeed recover true lost high frequency information.

The goal of this task is to simply experiment with one such super-resolution algorithm. This algorithm is a maximum *a posteriori* (MAP) algorithm based upon a least square (LS) framework. It works by taking a sequence of low resolution image frames (assumed to have camera motion) and estimates the frame-to-frame translations and rotations. It then registers the frames and performs a gradient descent optimization to estimate the underlying high resolution image that minimizes the error for a given cost function. This particular algorithm uses regularization terms based upon image smoothness similar to the discussion of the least squares algorithm in lecture.

For this task perform the following:

1. Load the IR scene sequence. Calibrate all 20 frames using the gain and bias estimates obtained from the two-point calibration in Task 1.

2. Run the $[\mathtt{z}, \mathtt{zi}, \mathtt{R}] = \mathtt{resEnahnce}(\mathtt{data}, \mathtt{L}, \mathtt{lambda}, \mathtt{Niter})$ function provided with this assignment. Experiment with the parameters (described below) until you find a result that you think is good.

    - `data: This parameter is simply the sequence of input low resolution image frames. Here you should experiment with the number of frames that you pass the algorithm. I would suggest you begin with all 20 frames and reduce the number, i.e., data(:,:,1:N) where N is the number of frames to use.`
    - `L: This parameter is the upsample factor for the data. Thus, for our data, the output image will be of size` $128L \times 128L$. `I would suggest experimenting with` L `in the range of [2,5]. Keep in mind this must be a positive integer.`
    - `lambda: This parameter specifies the weight of the regularization term in the cost function and should be specified in the range of [0,1]. Note that 0 will result in no regularization whereas a value of 1 will result in significant contribution of the regularization term.`
    - `Niter: This algorithm is iterative and uses a gradient descent algorithm to optimally find the minimum of the resulting complex error surface. This can take some time (or it can also get stuck in a localized minimum). This parameter simply tells the algorithm to quit after a fixed number of iterations. I would suggest you start by setting this parameter to 5 and increase from there. The more iterations you select the longer it can take to run (but with the data set being small you can get away with a fairly high number of iterations).`

3. The function will return the high resolution images `z` and `zi`. `z` is the final high resolution estimate after `Niter` iterations. The image returned in `zi` is the initial estimate of the high resolution scene that is obtained by simply interpolating the first low resolution image by a factor `L` using bilinear interpolation. It is useful to compare the two images so that resolution gains can be observed. Once you find a set of parameters that you like, display the first low resolution image `data(:,:,1)`, the initial high resolution image estimate `zi`, and the final high resolution image estimate `z`. Also indicate the parameters you use used (including the range of frames you used from the low resolution image sequence). You should be able to obtain a fairly impressive result!

NOTE: Due to parts of this code being proprietary I had to hide it inside of a `.p` file. It runs the same as a normal MATLAB function; you simply cannot edit it. My hope was to rewrite parts of this code to get around this but it is going to take much longer than anticipated. So, my apologies as I generally like to provide code to students. However, in this case I still wanted you to be able to experiment with these types of algorithms as I think they are pretty impressive and hope you find that true as well!