To replicate experimental procedure, we want to approximate a vector field using sampled velocities at a subset of points. To do so, we first take a smaller set of velocity points that will represent the **experimental velocity data**. This is representative of velocity data obtained using sensors in the experiment. We will then use bilinear interpolation to find the intermediary points between the known experimental data. We can check the correctness of these calculated intermediary points by comparing them with actual velocity points.

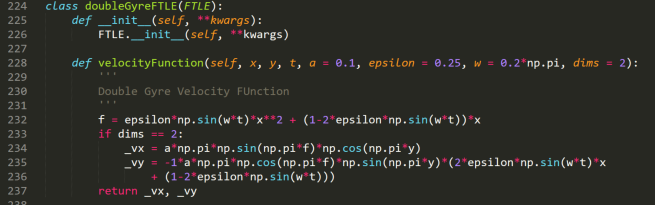
In our implementation, we have two methods which return velocity fields: one for generating an experimental, or estimated, velocity field, and another for generating a completely accurate velocity field. These are *getVelocity()* and *getEstimatedVelocity()*. The arguments for these methods are *x, y,* and *t*, where *x* and *y* are 1-dim arrays of the same length and *t* is a float representing the point in time of which the velocity field is calculated. These methods will both return 2 1-dim arrays of the same length as the arguments *x* and *y* corresponding to the *i* and *j* scalar components of the velocity field.

In the current experimental setup, we use the double gyre velocity field, which is:

where:

over the domain [ 0, 2 ] x [ 0, 1 ]. The velocity field is given by:

The implementation of this velocity field in the code is under the extended class *DoubleGyreFTLE* in the method *velocityFunction()*:



*getEstimatedVelocity()* is broken down in the following steps:

1. Create sampling coordinate set (subset of points where we will place “sensors” record the true velocity vectors. \_xarr and \_yarr are 1-dim arrays of length *self.length\*self.percentLength* and *self.height\*self.percentHeight* with equally spaced values between specified minimum and maximum x and y values.

\_xarr = np.linspace(self.minx, self.maxx, self.length \* self.percentLength)

\_yarr = np.linspace(self.miny, self.maxy, self.height \* self.percentHeight)

1. Convert \_xarr and \_yarr to a meshgrid (numPy object) which essentially create a coordinate matrix, or a 2-dim array of coordinates (x-y pairs). The x-y pairs can be used as inputs to the velocity function in order to get velocity data for x and y components.

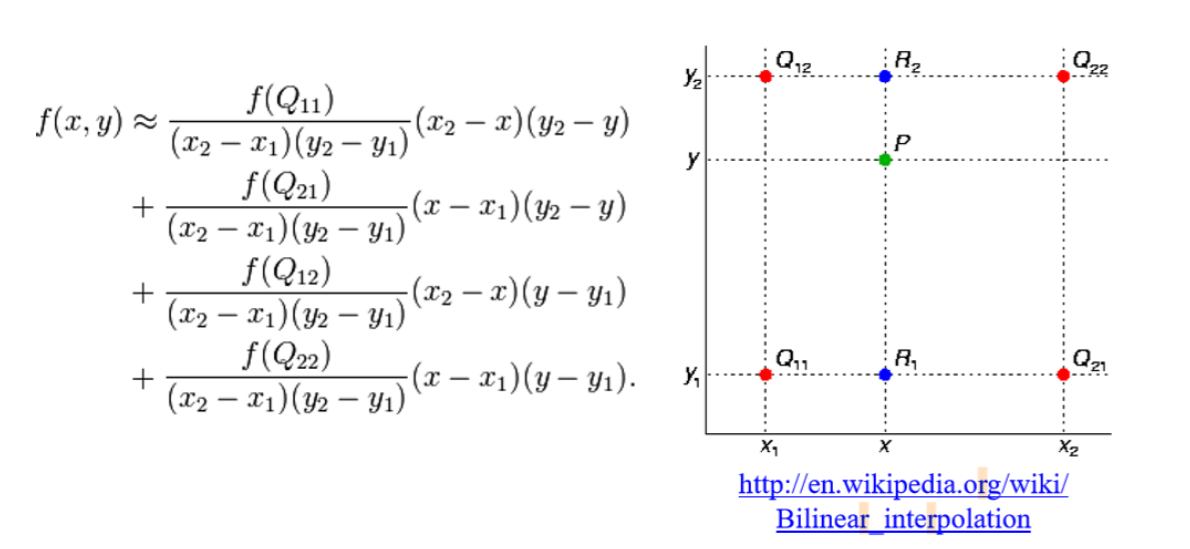
xdata, ydata = \*self.velocityFunction( \* np.meshgrid(\_xarr, \_yarr, indexing = 'ij', sparse = True), t)

1. Using the *RegularGridInterpolator* method of the Scipy library, we can generate an analytical function for the originally specified domain (*self.minx, self.maxx)* using linear interpolation between the points where velocity data is known.

vxfunc = RegularGridInterpolator((\_xarr, \_yarr), xdata, method = 'linear')

vyfunc = RegularGridInterpolator((\_xarr, \_yarr), ydata, method = 'linear')

Here is a simple illustration of how intermediary points are calculated :



1. The functions vxfunc and vyfunc are then used to calculated the points from the input 1-dim arrays x and y, and the function outputs 2 1-dim arrays outxv and outyv, which are of the same length as x and y.