In [2]:
```
def create_colormap(colors, position=None, bit=False, reverse=False, nam
e='custom_colormap'):
    """
    returns a linear custom colormap
    Parameters
    ----------
    colors : array-like
        contain RGB values. The RGB values may either be in 8-bit [0 to
 255]
        or arithmetic [0 to 1] (default).
        Arrange your tuples so that the first color is the lowest value
 for the
        colorbar and the last is the highest.
    position : array like
        contains values from 0 to 1 to dictate the location of each colo
r.
    bit : Boolean
        8-bit [0 to 255] (in which bit must be set to
        True when called) or arithmetic [0 to 1] (default)
    reverse : Boolean
        If you want to flip the scheme
    name : string
        name of the scheme if you plan to save it
    Returns
    -------
    cmap : matplotlib.colors.LinearSegmentedColormap
        cmap with equally spaced colors
    """
    from matplotlib.colors import LinearSegmentedColormap
    if not isinstance(colors, np.ndarray):
        colors = np.array(colors, dtype='f')
    if reverse:
        colors = colors[::-1]
    if position is not None and not isinstance(position, np.ndarray):
        position = np.array(position)
    elif position is None:
        position = np.linspace(0, 1, colors.shape[0])
    else:
        if position.size != colors.shape[0]:
            raise ValueError("position length must be the same as color
s")
        elif not np.isclose(position[0], 0) and not np.isclose(position[
-1], 1):
            raise ValueError("position must start with 0 and end with 1"
)
    if bit:
        colors[:] = [tuple(map(lambda x: x / 255., color)) for color in
colors]
    cdict = {'red':[], 'green':[], 'blue':[]}
    for pos, color in zip(position, colors):
        cdict['red'].append((pos, color[0], color[0]))
        cdict['green'].append((pos, color[1], color[1]))
        cdict['blue'].append((pos, color[2], color[2]))
    return LinearSegmentedColormap(name, cdict, 256)
```