```
SafeguardsCodec(
    codec=codec,  # any numcodecs codec
    safeguards=[
        # we allow for arbitrary grid point spacing,
        # and switch from 2nd order central finite differences to 2nd
        #  order forward/backwards finite differences at the data
        #  boundaries, where data points do not have a left, right, upper,
        #  and lower neighbour
        # this enables us to compute the Laplacian at all data points,
        #  including at the boundaries
        # we pad the data with NaNs to denote the missing values
        dict(
            kind="qoi_eb_stencil",
            qoi="""
                # declare a new variable d2Udy2
                V["d2Udy2"] = where(
                    c["Y"] > -1,
                    where(
                        c["Y"] < 1,
                        # use 2nd order central finite difference,
                        # where possible
                        finite_difference(
                            x, order=2, accuracy=2, type=0, axis=0,
                            grid_centre=c["Y"]
                        ),
                        # use 2nd order backwards finite difference
                        # at the upper boundary
                        finite_difference(
                            x, order=2, accuracy=1, type=-1, axis=0,
                            grid_centre=c["Y"]
                        ),
                    ),
                    # use 2nd order forward finite difference at the
                    # lower boundary
                    finite_difference(
                        x, order=2, accuracy=1, type=1, axis=0,
                        grid_centre=c["Y"]
                    ),
                );

                V["d2Udx2"] = where(
                    c["X"] > -1,
                    where(
                        c["X"] < 1,
                        # use 2nd order central finite difference
                        # where possible
                        finite_difference(
                            x, order=2, accuracy=2, type=0, axis=1,
                            grid_centre=c["X"]
                        ),
                        # use 2nd order backwards finite difference
                        # at the right boundary
                        finite_difference(
                            x, order=2, accuracy=1, type=-1, axis=1,
                            grid_centre=c["X"]
                        ),
                    ),
                    # use 2nd order forward finite difference at the
                    # left boundary
                    finite_difference(
                        x, order=2, accuracy=1, type=1, axis=1,
                        grid_centre=c["X"]
                    ),
                );

                # compute the natural logarithm of the Laplacian
                return ln(V["d2Udy2"] + V["d2Udx2"]);
            """,
            neighbourhood=[
                dict(
                    axis=0, before=1, after=1, boundary="constant",
                    constant_boundary=numpy.nan
                ),
                dict(
                    axis=1, before=1, after=1, boundary="constant",
                    constant_boundary=numpy.nan
                ),
            ],
            type="abs",
            eb=0.1,
        ),
    ],
    fixed_constants=dict(X=X, Y=Y),
)
```