

The `field.py` script is meant to be as close to vector calculus as possible. It uses 2D lists to describe a 2D scalar field, so that index i corresponds to coordinate x_i , i.e. $f(x_1, x_2) \sim f[x_1, x_2]$.

For 2D vector fields, however, we need another index to describe which component of the vector, at a point in \mathbb{R}^2 , we want. The first index is used for this, so that $f_i(x_1, x_2) = f[i, x_1, x_2]$. NumPy's `einsum` allows us to preform regular vector calculus operations on these fields. dot product is given by

$$h(x_1, x_2) = \vec{f} \cdot \vec{g} = \hat{e}_i \cdot \hat{e}_j f_i(x_1, x_2) g_j(x_1, x_2) = f_i(x_1, x_2) g_i(x_1, x_2),$$

and becomes

$$h = \text{einsum}["ixyz, ixyz \rightarrow xyz", f, g]$$

This is straight forward to generalize into 3D.

Defining `eijk[i, j, k] = ϵ_{ijk}` , the Levi-Cevita symbol, allows us to implement the cross product of two fields, as

$$\vec{h} = \vec{f} \times \vec{g} = \hat{e}_i h_i(x, y, z) = \epsilon_{ijk} f_j(x, y, z) g_k(x, y, z)$$

becomes

$$h = \text{einsum}["ijk, jxyz, kxyz \rightarrow ixyz", eijk, f, g]$$

NumPys build in function `gradient()` makes it possible to implement differential operators like the curl. As `gradien(f, axis = i + 1)` takes the difference along axis coordinate x_i (the +1 comes from skipping the component index), we can make a 5D (!) matrix `Df` with indexes such that $\partial_i f_j(x, y, z) \sim \text{Df}[i, j, x, y, z]$, and thus implement curl,

$$\vec{h} = \nabla \times \vec{f} = \hat{e}_i h_i(x, y, z) = \hat{e}_i \epsilon_{ijk} \partial_j f_k(x, y, z)$$

as

$$h = \text{einsum}["ijk, jkxyz \rightarrow ixyz", eijk, Df]$$