The `fields.py` script is meant to be as close to vector calculus as possible. It uses 3D lists to describe a 3D scalar field, so that index i corresponds to coordinate x_i, i.e. $f(x, y, z) \sim f[x, y, z]$.

For 3D vector fields, however, we need another index to describe which component of the vector, at a point in R^3, we want. The first index is used for this, so that $f_i(x, y, z) = f[i, x, y, z]$. NumPy's `einsum` allows us to preforme regular vector calculus operations on these fields. dot product is given by

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

and becomes

```
h = einsum["ixyz, ixyz -> xyz", f, g]
```

. Defining `eijk[i, j, k]` $= \epsilon_{ijk}$, 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 = einsum["ijk, jxyz, kxyz -> 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 $\partial_i f_j(x, y, z) \sim$ `Df[i, j, x, y, z]`, and thus impmlement 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 = einsum["ijk, jkxyz -> ixyz", eijk, Df]
```