Dislocation

Zelong Guo[1]
Section 1.4 Remote Sensing and Geoinformatics
GFZ German Research Center of Geosciences, Potadam, Germany
zelong.guo@gfz.de[2]

28.2.2025

[1]Personal Website: https://zelongguo.github.io/
[2]E-mail also: zelong.guo@outlook.com

Contents

Chapter 1

Okada Dislocation

1.1 The definition of the Okada Coordinates System

Understanding the parameters in **Okada DC3D**:

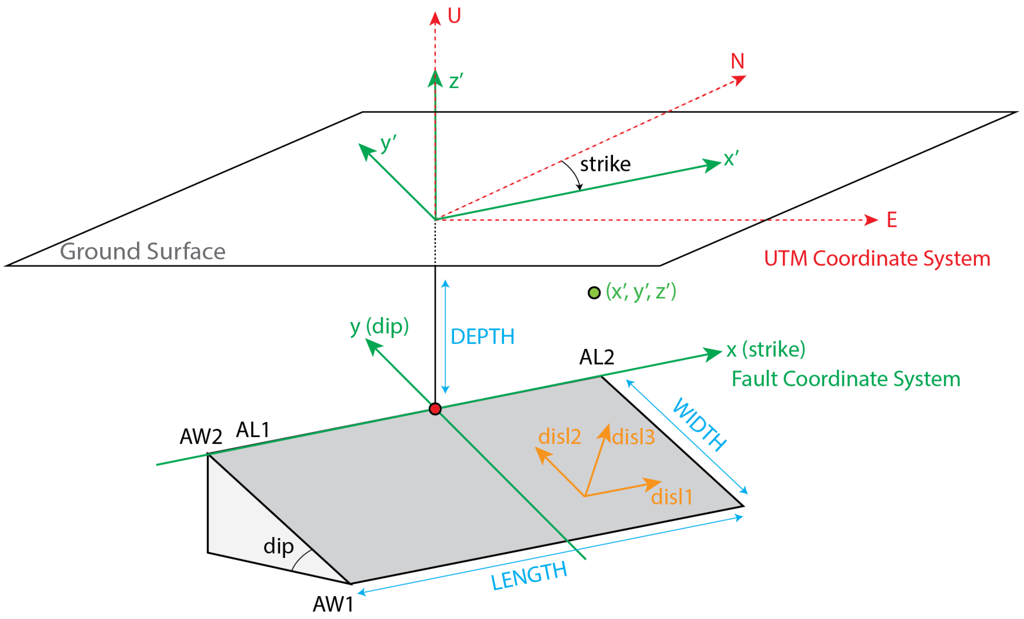

Figure 1.1: Okada fault geometry.

```
// This is a function codes in dislocation
dc3d(alpha, x, y, z, depth, dip, al1, al2, aw1, aw2, disl1, disl2, disl3,
      ux, uy, uz, uxx, uyx, uzx, uxy, uyy, uzy, uxz, uyz, uzz, iret);
```

Note in `dislocation`, `dc3d` is a very important function. There are several steps to convert the observation station UTM coordinates to local fault coordinates system. In above figure, we need changing the observations coordinates from UTM to `x'y'z'` (Note the fault coordinate system `xyz` is only for representing `(al1, al2, aw1, aw2)` in the `dc3d` function). This involves the knowledge of **coordinates transformation** (see our other notes for this part).

The part of the input parameters in `dc3d` are as follows (More details need to refer to Okada's materials):

- The input parameters (x, y, z) of `dc3d` are the **station coordinates** in `x'y'z'` coordinates system (not the local fault coordinate system `xyz`), as is shown in above Figure 1.1.
- `depth` is the depth of **reference point** of the fault (i.e. the red dot in Figure 1.1).

- Because we set the fault reference point with the **fault upper center edge point**, thus, $(al1, al2)$ should be $(-0.5 * length, 0.5 * length)$, while the $(aw1, aw2)$ should be $(-width, 0)$. The values of $(al1, al2)$ and $(aw1, aw2)$ are determined within **fault coordinates system** `xyz` (see Figure 1.1). That is, the fault coordinate system is only used for representing the `aw` and `al` parameters in `dc3d`.

Thus, the output parameters by `dc3d` are all based on intermediate fault system `x'y'z`, so we need to convert the output `ux, uy, uz` and the `uxx-uzz` paramters from `x'y'z'` to `UTM` system.

In the following sections, we make formula derivation to the coordinate transformation, espatially for `uxx-uzz` from `x'y'z'` to `UTM`.

1.1.1 Basic Transformation Matrix

Firstly, we need to review the transformation matrix. Here, we give the transformation matrix directly, and you could refer to the old notes for more details:

We define the `counterclockwise rotation` is positive direction, that is also the **right-hand rule**, we have:

- Rotating α based on `x` axis (counterclockwise rotation within `yoz` plane):

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \tag{1.1}$$

- Rotating α based on `y` axis (counterclockwise rotation within `xoz` plane):

$$R_y(\alpha) = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix} \tag{1.2}$$

- Rotating α based on `z` axis (counterclockwise rotation within `xoy` plane):

$$R_z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.3}$$

According to whether the ratation is based on **fixed axis** or **dynamic axis**, you should consider whether it is right or left multiplication of R carefully.

1.1.2 Displacements Transformation

From `UTM` to `x'y'z'`, calculating coordinates in `x'y'z'`

In Fig 1.1, we note if we want converting `UTM` to `x'y'z'`, then `UTM` needs rotating `(90-strike)` **counterclockwise** based on `z` axis. Because we want to calculate the **a new `x'y'z'` coordinates of a same point which has known coordinates in the old `UTM` system**. According to the "Robot Theory" we can make the transformation with this: ${}^{A}P = {}^{A}_{B}R \cdot {}^{B}P$, thus we can get:

$${}^{x'y'z'}P = {}^{x'y'z'}_{UTM}R \cdot {}^{UTM}P \tag{1.4}$$

where ${}^{x'y'z'}_{UTM}R$ represents the process from `x'y'z'` to `UTM` with `x'y'z'` system rotating $360 - (90 - \text{strike})$ counterclockwise to `UTM`. We take this as rotating with the `fixed z axis`, thus it would be left multiplication:

$$\begin{aligned} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\mathrm{x'y'z'}} &= \begin{bmatrix} \cos(360-(90-\text{strike})) & -\sin(360-(90-\text{strike})) & 0 \\ \sin(360-(90-\text{strike})) & \cos(360-(90-\text{strike})) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\mathrm{UTM}} \\ &= \begin{bmatrix} \cos(\text{strike}-90) & -\sin(\text{strike}-90) & 0 \\ \sin(\text{strike})-90 & \cos(\text{strike}-90) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\mathrm{UTM}} \end{aligned} \tag{1.5}$$

In our codes `okada_disloc3d.c`, we have:

```
// okada_disloc3d.c
strike = model[5] - 90.0;  // model[5] is the original strike angle
// Apply some translations to transfer Observation Cartesian to Fault Coordinate
x = cs * (obs[0] - model[0]) - ss * (obs[1] - model[1]);
y = ss * (obs[0] - model[0]) + cs * (obs[1] - model[1]);
z = obs[2];
```

From `x'y'z'` to `UTM`, calculating coordinates in `UTM`

Similarly, `UTM` should counterclockwise rotate $(90 - \text{strike})$ to `x'y'z'` based on `z` axis, then we have:

$$ {}^{UTM}P = {}^{UTM}_{x'y'z'}R \cdot {}^{x'y'z'}P \tag{1.6} $$

i.e.,

$$ \begin{aligned} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{UTM} &= \begin{bmatrix} \cos(90-\text{strike}) & -\sin(90-\text{strike}) & 0 \\ \sin(90-\text{strike}) & \cos(90-\text{strike}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{x'y'z'} \\ &= \begin{bmatrix} \cos(\text{strike}-90) & \sin(\text{strike}-90) & 0 \\ -\sin(\text{strike}-90) & \cos(\text{strike}-90) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{x'y'z'} \end{aligned} \tag{1.7} $$

Thus, in the `okada_disloc3d`, we have:

```
// rotate then add, from x'y'z' system back to UTM system
uxt += cs * ux + ss * uy;
uyt += -ss * ux + cs * uy;
uzt += uz;
```

1.1.3 Stress and Strain Transformation

From `x'y'z'` to `UTM`, calculating coordinates in `UTM`

Because stress and strain involves with the first-order derivatives of the displacement, so it is kind of complex. Firstly, we need get to know the output of `dc3d` are the **first-order displacement derivatives (`uxx-uzz`)**, then we can based on the relationship of **displacements, stress and strian** of **Elasticity Theory** to make the transformation.

We need to know the `dc3d` output displacements `(ux, uy, uz)` and the 1st derivatives `(uxx-uzz)` are all based on the `x'y'z'` coordinate system. We need transform these parameters from `x'y'z'` to `UTM`. For displacements, we have already talked how we can do such transformation in above section. In this section, we focus on how to do the transformation for **stress** and **strain**.

Displacement components can be wrote with **tensor**:

$$ \mathbf{u} = \mathbf{u}(x_1, x_2, x_3) \tag{1.8} $$

which could be further expanded as:

$$ \begin{aligned} u_1 &= u_1(x_1, x_2, x_3) \\ u_2 &= u_2(x_1, x_2, x_3) \\ u_3 &= u_3(x_1, x_2, x_3) \end{aligned} \tag{1.9} $$

or

$$ \begin{aligned} u &= u(x, y, z) \\ v &= v(x, y, z) \\ w &= w(x, y, z) \end{aligned} \tag{1.10} $$

That is, the output displacements of the `dc3d` `(ux, uy, uz)` are the above (u_1, u_2, u_3) or the (u, v, w) components. I.e., $\mathtt{ux} = u, \mathtt{uy} = v, \mathtt{uz} = w$. In the following part, we use (u, v, w) to represent the displacements and the corresponding derivatives.

Important Note

Displacements are the **continuous function** of the coordinate values! （位移是坐标值的连续函数！）Thus, let's say, the displacement components between 2 points A and B along `x, y` and `z` axis are (u, v, w), which could be represented as:

$$\begin{aligned} &u(x+dx, y, z); v(x+dx, y, z); w(x+dx, y, z) \\ &u(x, y+dy, z); v(x, y+dy, z); w(x, y+dy, z) \\ &u(x, y, z+dz); v(x, y, z+dz); w(x, y, z+dz) \end{aligned} \tag{1.11}$$

See Elasticity Theory for more details.

Thus, the 9 1st spatial derivatives of the displacement components can be wrote with:

$$\begin{aligned} \mathtt{uxx} &= \frac{\partial u}{\partial x}, \mathtt{uxy} = \frac{\partial u}{\partial y}, \mathtt{uxz} = \frac{\partial u}{\partial z} \\ \mathtt{uyx} &= \frac{\partial v}{\partial x}, \mathtt{uyy} = \frac{\partial v}{\partial y}, \mathtt{uyz} = \frac{\partial v}{\partial z} \\ \mathtt{uzx} &= \frac{\partial w}{\partial x}, \mathtt{uzy} = \frac{\partial w}{\partial y}, \mathtt{uzz} = \frac{\partial w}{\partial z} \end{aligned} \tag{1.12}$$

Now the point is that we want to convert all these variables which are based on `x'y'z'` to `UTM`. So the most important thing now is transforming the 1st derivatives to `UTM`.