Rotation of Coordinates Based On CORDEX Domains

If you have several CORDEX NetCDF files and you want to extract data from it so you can use two methods. Usually in these files you have lat/lon as two-dimensions variables and rlat/rlon as one-dimension variables so: Search your latitude and longitude and find the proper cell in two dimension variables(lat/lon) then find the grid number from two dimension variable. Then you can use rlat/rlon to read data from the main variable such as pr, tasmax, tas,... You can use Open Nc Fileand CORDEX Data Extractor to do this method and this process will be done in the behind code. But you can use another method for using Matlab or Netcdf Extractor. In this method, you should convert your regular coordinate to rotated coordinate and then find the Grid Numbers in rlat/rlon.

This tool can convert non-rotated coordinate to rotated coordinate and vice versa. The pivot of rotations is based on CORDEX domains(or you can change it). For extracting data from CORDEX netcdf files, you should convert your regular coordinate to rotated coordinate by this tool and then extract your data by using rotated coordinate and rlat/rlon variables in your file. You can extract data by any language programming or you can use Netcdf Extractor in this website.

For checking the correctness of the tool, you can use TLC(Top Left Corner) of each domain in CORDEX Domains and convert non-Rotated to rotated and vice versa.

If you want extract data from CORDEX NetCDF files you can do it simply by using CORDEX Data Extractor

If you have further question please contact with kolsoomi57@gmail.com

```
private Point convert2Rotated_S(Point your, bool direct, Point pole)
  {
    // Reference:
    // http://gis.stackexchange.com/questions/10808/lon-lat-transformation/14445
    var lon = your.X * Math.PI / 180; // Convert degrees to radians
    var lat = your.Y * Math.PI / 180;
    var theta = (-90 + pole.Y); // Rotation around y-axis
    var phi = pole.X; // Rotation around z-axis
    phi = (phi * Math.PI) / 180; // Convert degrees to radians
    theta = (theta * Math.PI) / 180;
    var x = Math.Cos(lon) * Math.Cos(lat); // Convert from spherical to cartesian coordinates
    var y = Math.Sin(lon) * Math.Cos(lat);
    var z = Math.Sin(lat);
    double x_new = 0;
    double y_new = 0;
    double z_new = 0;
    if (direct) // Regular -> Rotated
    {
      x_new = Math.Cos(theta) * Math.Cos(phi) * x + Math.Cos(theta) * Math.Sin(phi) * y + Math.Sin(theta) * z;
      y_new = -Math.Sin(phi) * x + Math.Cos(phi) * y;
      z_new = -Math.Sin(theta) * Math.Cos(phi) * x - Math.Sin(theta) * Math.Sin(phi) * y + Math.Cos(theta) * z;
    }
    else // Rotated -> Regular
    {
      phi = -phi;
      theta = -theta;
      x_new = Math.Cos(theta) * Math.Cos(phi) * x + Math.Sin(phi) * y + Math.Sin(theta) * Math.Cos(phi) * z;
      y_new = -Math.Cos(theta) * Math.Sin(phi) * x + Math.Cos(phi) * y - Math.Sin(theta) * Math.Sin(phi) * z;
      z_new = -Math.Sin(theta) * x + Math.Cos(theta) * z;
    }
    var lon_new = Math.Atan2(y_new, x_new); // Convert cartesian back to spherical coordinates
    var lat_new = Math.Asin(z_new);
    lon_new = (lon_new * 180) / Math.PI; // Convert radians back to degrees
    lat_new = (lat_new * 180) / Math.PI;
    lon_new = (lon_new + 360) % 360;
    Point grid_out = new Point() { X = lon_new, Y = lat_new };
    return grid_out;
  }
```