In [2]:
```
import geopandas as gpd
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.colors as mcolors
import numpy as np
import pandas as pd
import pygmt
import pyproj
from shapely.geometry import box
from shapely.ops import unary_union
```

In [5]:
```
omegas = pd.read_csv('data/poles.IGS08.txt', names=['LAT', 'LON', 'RATE', 'CODE'], header=None, delim_whitespace=True, keep_default_na=False)
```

```
/var/folders/6l/7c4lc3h945qclbjjyb4wkqy80000gn/T/ipykernel_60462/3287462290.py:1: FutureWarning: The 'delim_whitespace' keyword in pd.read_csv is deprecated and will be removed in a future version. Use ``sep='\s+'`` instead
  omegas = pd.read_csv('data/poles.IGS08.txt', names=['LAT', 'LON', 'RATE', 'CODE'], header=None, delim_whitespace=True, keep_default_na=False)
```

In [6]:
```
crs_lla  = pyproj.CRS("EPSG:4326")  # WGS84 lat,lon
crs_ecef = pyproj.CRS("EPSG:4978")  # ECEF (X,Y,Z)
transform_ll_to_ecef = pyproj.Transformer.from_crs(crs_lla, crs_ecef, always_xy=True)
```

In [8]:
```
SECONDS_PER_YEAR = 365.25 * 24 * 3600

# Converting to units we can use for the euler polar equation
def degMyr_to_rads(rate_deg_ma):
    return rate_deg_ma * np.pi / 180.0 / (1e6 * SECONDS_PER_YEAR)
```

In [9]:
```
# Implementing the euler pole equation and converting it to the units we need

def make_euler_vector(plate_code):
    pole_row = omegas.loc[omegas["CODE"] == plate_code]

    lat_deg = pole_row["LAT"].values[0]
    lon_deg = pole_row["LON"].values[0]
    rate_deg_ma = pole_row["RATE"].values[0]

    X, Y, Z = transform_ll_to_ecef.transform(lon_deg, lat_deg, 0.0)
    mag = np.sqrt(X*X + Y*Y + Z*Z)
    unit_dir = np.array([X/mag, Y/mag, Z/mag])
    rate_rads = degMyr_to_rads(rate_deg_ma)
    return unit_dir * rate_rads
```

In [10]:
```
# Naming the dataset columns we had to download

col_names = [
    "station_id",       # (col 1)
    "midas_version",    # (col 2)
```

```
    "tstart",           # (col 3)
    "tend",             # (col 4)
    "duration",         # (col 5)
    "num_epochs_total", # (col 6)
    "num_epochs_used",  # (col 7)
    "num_velocity_pairs", # (col 8)
    "Ve", "Vn", "Vu",   # (cols 9,10,11) east,north,up velocity (m/yr)
    "Se", "Sn", "Su",   # (cols 12,13,14) velocity uncertainties
    "offsetE", "offsetN", "offsetU", # (cols 15,16,17)
    "outliersE", "outliersN", "outliersU", # (cols 18,19,20)
    "std_velE", "std_velN", "std_velU",    # (cols 21,22,23)
    "num_steps",        # (col 24)
    "lat", "lon", "height"  # (cols 25,26,27)
]

midas_file = "data/midas.IGS14.txt"

# Loading data into df
df_stations = pd.read_csv(
    midas_file,
    delim_whitespace=True,
    comment="#",
    header=None,
    names=col_names,
    usecols=range(27)
)

# Setting lon from -180 to 180, a more standard version than what the dataset has (-360 to 0)
df_stations["lon"] = df_stations["lon"].apply(
    lambda x: x + 360 if x < -180 else x
)
```

```
/var/folders/6l/7c4lc3h945qclbjjyb4wkqy80000gn/T/ipykernel_60462/1182422711.py:24: FutureWarning: The 'delim_whitespace' keyword in pd.read_csv is deprecated an
d will be removed in a future version. Use ``sep='\s+'`` instead
  df_stations = pd.read_csv(
```

In [11]:

```
# Mapping for the dataset I had (Bird 2003) for figuring out which station is on which plate
plate_mapping = {
    "AF": "Africa",
    "AN": "Antarctica",
    "AR": "Arabia",
    "AU": "Australia",
    "BU": "Burma",
    "CA": "Caribbean",
    "CO": "Cocos",
    "EU": "Eurasian",
    "IN": "Indian",
    "MA": "Mariana",
    "NA": "North America",
    "NB": "North Bismark",
```

```
    "NZ": "Nazca",
    "OK": "Okhotsk",
    "ON": "Okinawa",
    "PA": "Pacific",
    "PM": "Panama",
    "PS": "Philippine Sea",
    "SA": "South America",
    "SB": "South Bismark",
    "SC": "Scotia",
    "SL": "Shetland",
    "SO": "Somalia",
    "SU": "Sunda",
    "WL": "Woodlark"
}
```

In [12]:

```
# Using the shapefile (Bird 2003) to add column for plate name and code for each station

gd_df_stations = gpd.GeoDataFrame(
    df_stations,
    geometry=gpd.points_from_xy(df_stations["lon"], df_stations["lat"]),
    crs="EPSG:4326"
)

plates = gpd.read_file("data/tectonicplates-master/PB2002_plates.shp").to_crs(epsg=4326)

gd_df_stations_joined = gpd.sjoin(
    gd_df_stations,
    plates,
    how="left",
    predicate="within"
)

# Rename Code to plate_code and PlateName to plate_name
gd_df_stations_joined.rename(columns={"Code": "plate_code", "PlateName": "plate_name"}, inplace=True)
```

In [13]:

```
# Define the needed cross product for the equation we must implement

def cross_product(omega, xyz):
    return np.cross(omega, xyz)

def get_station_coordinates(station_id):
    station_data = df_stations[df_stations['station_id'] == station_id]
    if not station_data.empty:
        lat_lon_height = station_data[['lat', 'lon', 'height']].values[0]
        return lat_lon_height
    else:
        return None
```

```
#### Examples ####

station_id = "0ABI"
plate = "EU"

lat_deg, lon_deg, h_m = get_station_coordinates(station_id) # Example of how to use the functions to get the velocity of a station
lat_deg, lon_deg, h_m = [51.986, 4.388, 0] # Delft lat, lon, h that we needed to check. Assumign 0 here for height


X, Y, Z = transform_ll_to_ecef.transform(lon_deg, lat_deg, h_m)
x_ecef = np.array([X, Y, Z])
omega_ecef = make_euler_vector(plate)
v_ecef_m_s = cross_product(omega_ecef, x_ecef)
v_ecef_mm_yr = v_ecef_m_s * SECONDS_PER_YEAR * 1000.0

print(f"Delft: lat={lat_deg:.5f}, lon={lon_deg:.5f} -> vECEF={v_ecef_mm_yr} mm/yr")
```

```
Delft: lat=51.98600, lon=4.38800 -> vECEF=[-13.5382852   16.46864103   9.63111986] mm/yr
```

In [14]:

```
# This cell will convert the ECEF velocity to ENU velocity based on the station's latitude and longitude
lat_rad = np.radians(lat_deg)
lon_rad = np.radians(lon_deg)

# East unit vector (ECEF)
east = np.array([
    -np.sin(lon_rad),
     np.cos(lon_rad),
     0.0
])

# North unit vector (ECEF)
north = np.array([
    -np.sin(lat_rad)*np.cos(lon_rad),
    -np.sin(lat_rad)*np.sin(lon_rad),
     np.cos(lat_rad)
])

# Up unit vector (ECEF)
up = np.array([
     np.cos(lat_rad)*np.cos(lon_rad),
     np.cos(lat_rad)*np.sin(lon_rad),
     np.sin(lat_rad)
])

# Build rotation matrix
R = np.vstack([east, north, up])

# Transform ECEF velocity to ENU
v_enu_mm_yr = R @ v_ecef_mm_yr
```

```
vE = v_enu_mm_yr[0]
vN = v_enu_mm_yr[1]
vU = v_enu_mm_yr[2]

#### Delft check: ####
print(f"Velocity East = {vE:.4f} mm/yr")
print(f"Velocity North = {vN:.4f} mm/yr")
print(f"Velocity Up = {vU:.4f} mm/yr")

### We see that East = 17.4562, North is 15.5737 (both mm/yr). Not exactly the same but very close.
### Is this due to the rotation / conversion above?
```

```
Velocity East = 17.4562 mm/yr
Velocity North = 15.5737 mm/yr
Velocity Up = 0.0508 mm/yr
```

In [15]:

```
# Make a dictionary for the euler vectors of the plates

plate_euler = {}
for idx, row in omegas.iterrows():
    code = row["CODE"]
    plate_euler[code] = make_euler_vector(code)

#### Example ####
omega_NA = plate_euler["EU"]
omega_PA = plate_euler["IN"]

print("NA plate euler vector (rad/s):", omega_NA)
print("PA plate euler vector (rad/s):", omega_PA)
```

```
NA plate euler vector (rad/s): [-1.37246595e-17 -7.88159612e-17  1.15478099e-16]
PA plate euler vector (rad/s): [ 1.75750376e-16 -1.92930573e-17  2.16943287e-16]
```

In [16]:

```
#### This combines all of the above into a function that can be used to predict the velocity of a station (a row in a df)
#### Resulting in both ECEF and ENU appended
#### Everything in m/s or rad/s

def station_predicted_velocity(row):
    lat_deg = row["lat"]
    lon_deg = row["lon"]
    h_m     = row["height"]
    plate_code = row["plate_code"]

    # Station position in ECEF
    X, Y, Z = transform_ll_to_ecef.transform(lon_deg, lat_deg, h_m)  # (m)
    xyz = np.array([X, Y, Z])

    # Euler component
    omega_xyz = plate_euler[plate_code]  # rad/s

    # Cross product => velocity in ECEF (m/s)
```

```python
    v_ecef_m_s = np.cross(omega_xyz, xyz)

    # Rotate ECEF -> local ENU
    lat_rad = np.radians(lat_deg)
    lon_rad = np.radians(lon_deg)

    # East unit vector (ECEF)
    east = np.array([
        -np.sin(lon_rad),
         np.cos(lon_rad),
         0.0
    ])

    # North unit vector (ECEF)
    north = np.array([
        -np.sin(lat_rad)*np.cos(lon_rad),
        -np.sin(lat_rad)*np.sin(lon_rad),
         np.cos(lat_rad)
    ])

    # Up unit vector (ECEF)
    up = np.array([
         np.cos(lat_rad)*np.cos(lon_rad),
         np.cos(lat_rad)*np.sin(lon_rad),
         np.sin(lat_rad)
    ])

    # Rotation matrix
    R = np.vstack([east, north, up])

    # Multiply => local ENU components (m/s)
    v_enu_m_s = R @ v_ecef_m_s
    vE, vN, vU = v_enu_m_s

    # Return both ECEF and ENU velocity
    return pd.Series({
        "vX_ecef_m_s": v_ecef_m_s[0],
        "vY_ecef_m_s": v_ecef_m_s[1],
        "vZ_ecef_m_s": v_ecef_m_s[2],
        "vE_pred": vE,  # m/yr
        "vN_pred": vN,
        "vU_pred": vU
    })
```

In [17]:

```python
### Apply function above to calculate the velocities for each point
### Added filtering to exclude certain plate codes as they were in the Bird 2003 dataset but not in the poles.IGS08 dataset
exclude_plate_codes = ['ND', 'AT', 'AP', 'KE', 'BS', 'TI', 'NH']
filtered_df = gd_df_stations_joined[~gd_df_stations_joined['plate_code'].isin(exclude_plate_codes)]
velocity_results = filtered_df.apply(station_predicted_velocity, axis=1)
filtered_df = filtered_df.join(velocity_results)
```

In [18]:
```
### Quick check

filtered_df.columns
filtered_df.head()
```

Out[18]:

	station_id	midas_version	tstart	tend	duration	num_epochs_total	num_epochs_used	num_velocity_pairs	Ve	Vn	...	index_right	LAYER	plate_code
0	00NA	MIDAS5	2008.2355	2018.7324	10.4969	3190	2968	5171	0.036213	0.058799	...	4	plate	AU
1	01NA	MIDAS5	2008.2683	2019.7426	11.4743	2362	2362	3599	0.035826	0.059595	...	4	plate	AU
2	02NA	MIDAS5	2008.7255	2016.9993	8.2738	1913	1913	3392	0.036171	0.059940	...	4	plate	AU
3	0ABI	MIDAS5	2009.4428	2024.7420	15.2992	5567	5532	10338	0.015129	0.014934	...	5	plate	EU
4	0ABN	MIDAS5	2023.2580	2024.7420	1.4840	540	284	284	0.015129	0.017643	...	5	plate	EU

5 rows × 38 columns

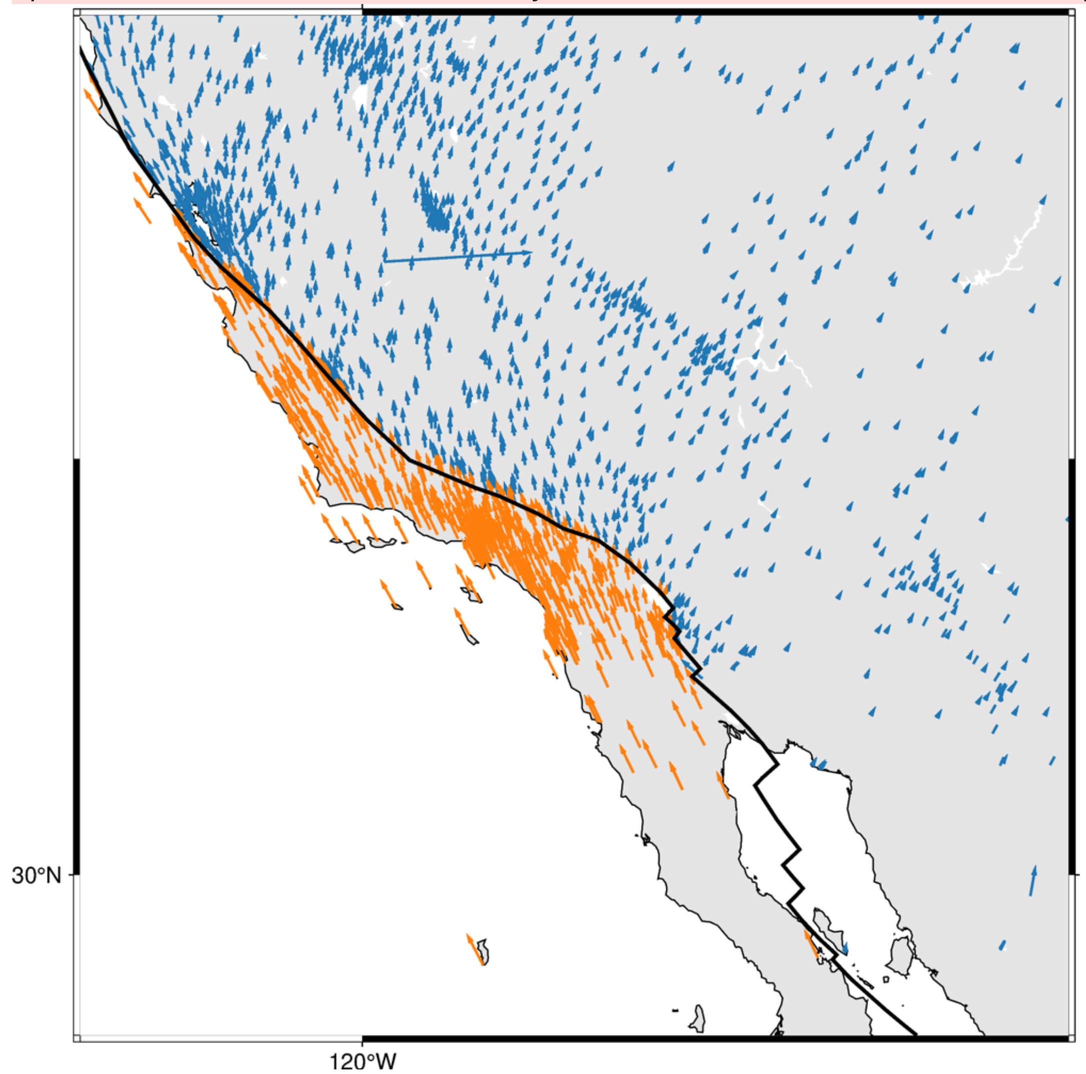
In [19]:
```
### Compute relative velocities to the plate, as asked
### This is to the own plate, not to the one opposite / other plate

filtered_df["resE"] = filtered_df["vE_pred"] - filtered_df["Ve"]
filtered_df["resN"] = filtered_df["vN_pred"] - filtered_df["Vn"]
```

In [24]:
```
### Plotting code ###

# Colours for each plate
unique_plates = filtered_df["plate_code"].unique()
n_colors = len(unique_plates)
colormap = cm.get_cmap("tab10", n_colors)
plate_color_map = {plate: mcolors.to_hex(colormap(i)) for i, plate in enumerate(unique_plates)}

# Region to plot [west, east, south, north]
region = [-124, -110, 28, 40]

# For visibility of each arrow
scale_factor = 10

# Calculating the angle and length of the relative velocities
filtered_df["angle_deg"] = np.degrees(np.arctan2(filtered_df["resN"], filtered_df["resE"]))
filtered_df["length"] = np.hypot(filtered_df["resE"], filtered_df["resN"]) * scale_factor

### Plotting the data ###
```

```
fig = pygmt.Figure()

fig.basemap(
    region=region,
    projection="M15c",
    frame=["xa30f10", "ya15f5", "WSen"]
)

fig.coast(shorelines="1/0.5p", land="gray90", water="white")

# Plot velocity vectors for each plate_code with its assigned color and calculated direction / magnitude
for plate in unique_plates:
    df_plate = filtered_df[filtered_df["plate_code"] == plate]
    color = plate_color_map[plate]
    fig.plot(
        x=df_plate["lon"],
        y=df_plate["lat"],
        style="V0.15c+eA",
        pen=f"1p,{color}",
        fill=color,
        direction=[
            df_plate["angle_deg"],   # the computed angle
            df_plate["length"],      # the computed length
        ],
    )

### Here we add the plate boundaries to the plot
plate_boundaries = gpd.read_file("data/tectonicplates-master/PB2002_boundaries.shp")

# Filter geometries to those that intersect with the region in the map
bbox = box(*region)
plate_boundaries_in_region = plate_boundaries[plate_boundaries.intersects(bbox)]

# Convert each LineString/MultiLineString to lon/lat dfs to plot them
lines = []
for geom in plate_boundaries_in_region.geometry:
    if geom is None:
        continue
    if geom.geom_type == 'LineString':
        x, y = geom.xy
        df_line = pd.DataFrame({'lon': x, 'lat': y})
        lines.append(df_line)
        lines.append(pd.DataFrame({'lon': [np.nan], 'lat': [np.nan]}))
    elif geom.geom_type == 'MultiLineString':
        for line in geom.geoms:
            x, y = line.xy
            df_line = pd.DataFrame({'lon': x, 'lat': y})
            lines.append(df_line)
            lines.append(pd.DataFrame({'lon': [np.nan], 'lat': [np.nan]}))
```

```
if lines:
    boundary_df = pd.concat(lines, ignore_index=True)
    fig.plot(data=boundary_df, pen="1.5p,black")

### Display plot
fig.show()
```

```
/var/folders/6l/7c4lc3h945qclbjjyb4wkqy80000gn/T/ipykernel_60462/1056575990.py:6: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap()`` or ``pyplot.get_cmap()`` instead.
  colormap = cm.get_cmap("tab10", n_colors)
plot [WARNING]: Your data array row 17 contains NaNs - no resampling taken place!
```



In [ ]:

In [ ]: