In [1]:
```
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 pyproj
from shapely.geometry import box
from shapely.ops import unary_union
```

In [2]:
```
omegas = pd.read_csv('data/poles.IGS08.txt', names=['LAT', 'LON', 'RATE', 'CODE'], header=None, delim_wh:
```

```
/var/folders/6l/7c4lc3h945qclbjjyb4wkqy80000gn/T/ipykernel_86140/3287462290.py:1: FutureWarning: The 'del
im_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_w
hitespace=True, keep_default_na=False)
```

In [3]:
```
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)
SECONDS_PER_YEAR = 365.25 * 24 * 3600
```

In [4]:
```
# 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
)

# Convert Ve, Vn, Vu to m/s
df_stations["Ve"] = df_stations["Ve"] / SECONDS_PER_YEAR
df_stations["Vn"] = df_stations["Vn"] / SECONDS_PER_YEAR
df_stations["Vu"] = df_stations["Vu"] / SECONDS_PER_YEAR
```

```
/var/folders/6l/7c4lc3h945qclbjjyb4wkqy80000gn/T/ipykernel_86140/3478071753.py:24: FutureWarning: The 'de
lim_whitespace' keyword in pd.read_csv is deprecated and will be removed in a future version. Use ``sep='
\s+'`` instead
  df_stations = pd.read_csv(
```

In [5]:
```
# 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 [6]:

```
# 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 [7]:

```
# 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 [8]:

```
# 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

def euler_vector_ecef(lat_deg, lon_deg, rate_deg_ma):

    # Convert degrees to radians
    lat_rad = np.radians(lat_deg)   # φ_p
    lon_rad = np.radians(lon_deg)   # λ_p

    # Convert rotation rate from deg/Myr to rad/s
    omega_rad_s = degMyr_to_rads(rate_deg_ma)
```

```python
    # Apply exactly the matrix/vector definition from the slides
    ox = omega_rad_s * np.cos(lat_rad) * np.cos(lon_rad)
    oy = omega_rad_s * np.cos(lat_rad) * np.sin(lon_rad)
    oz = omega_rad_s * np.sin(lat_rad)

    return np.array([ox, oy, oz])
```

In [10]:
```python
def station_ecef(lat_deg, lon_deg, h_m, radius=6378000):
    lat_rad = np.radians(lat_deg)
    lon_rad = np.radians(lon_deg)
    x = (radius + h_m) * np.cos(lat_rad) * np.cos(lon_rad)
    y = (radius + h_m) * np.cos(lat_rad) * np.sin(lon_rad)
    z = (radius + h_m) * np.sin(lat_rad)
    return np.array([x, y, z])

def velocity_ecef(lat_station, lon_station,
                  lat_pole, lon_pole, rate_deg_ma, h_m):
    # Euler vector in ECEF
    omega_ecef = euler_vector_ecef(lat_pole, lon_pole, rate_deg_ma)
    # Station ECEF
    xyz_station = station_ecef(lat_station, lon_station, h_m)
    # Cross product => velocity in ECEF (m/s)
    v_ecef = np.cross(omega_ecef, xyz_station)
    return v_ecef

def enu_rotation_matrix(lat_deg, lon_deg):
    lat_rad = np.radians(lat_deg)
    lon_rad = np.radians(lon_deg)
    # East unit vector
    e_east  = [-np.sin(lon_rad),  np.cos(lon_rad), 0.0]
    # North unit vector
    e_north = [-np.sin(lat_rad)*np.cos(lon_rad),
               -np.sin(lat_rad)*np.sin(lon_rad),
                np.cos(lat_rad)]
    # Up unit vector
    e_up    = [np.cos(lat_rad)*np.cos(lon_rad),
               np.cos(lat_rad)*np.sin(lon_rad),
               np.sin(lat_rad)]
    return np.array([e_east, e_north, e_up])

def velocity_enu(lat_station, lon_station,
                 lat_pole, lon_pole, rate_deg_ma, h_m):
    v_xyz = velocity_ecef(lat_station, lon_station,
                          lat_pole, lon_pole, rate_deg_ma, h_m)
    M_enu = enu_rotation_matrix(lat_station, lon_station)
    # local velocity = M_enu * v_xyz
    return M_enu @ v_xyz
```

In [11]:
```python
def get_plate_euler(plate_code):
    row = omegas.loc[omegas["CODE"] == plate_code]
    if row.empty:
        raise ValueError(f"No Euler pole found for plate code {plate_code}")
    return float(row["LAT"].values[0]), float(row["LON"].values[0]), float(row["RATE"].values[0])

def compute_all_predicted_velocities(df, exclude_plate_codes=None):
    if exclude_plate_codes is None:
        exclude_plate_codes = []

    df = df.copy()

    unique_plate_codes = [code for code in df['plate_code'].unique()
                          if code not in exclude_plate_codes]

    # Loop over each plate code to compute the velocities relative to the plates
    for code in unique_plate_codes:
        lat_pole, lon_pole, rate_deg_ma = get_plate_euler(code)

        # Actually compute the velocities with function below in mm/year
        def compute_velocity(row):
            v_enu_m_s = velocity_enu(row["lat"], row["lon"],
                                     lat_pole, lon_pole, rate_deg_ma, row["height"])
            return v_enu_m_s * (1000.0 * SECONDS_PER_YEAR)

        # Apply the function to each row
```

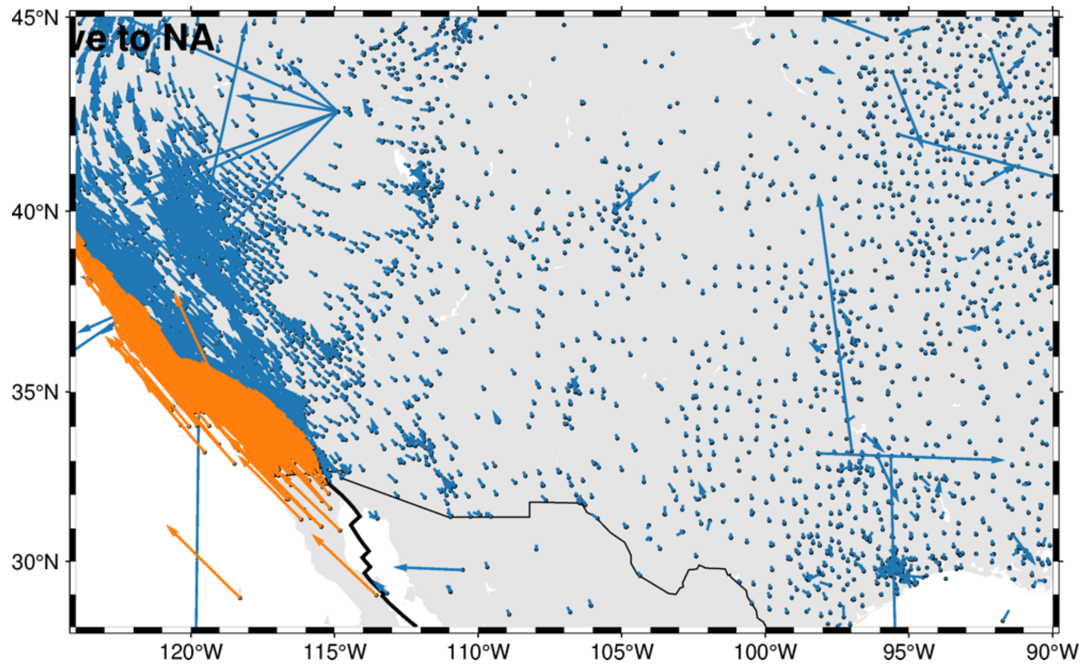
```
        velocities = df.apply(compute_velocity, axis=1)

        df[f"{code}_ve_pred"] = velocities.apply(lambda v: v[0])
        df[f"{code}_vn_pred"] = velocities.apply(lambda v: v[1])
        df[f"{code}_vu_pred"] = velocities.apply(lambda v: v[2])

    return df

# Apply to stations, filtering extra plate_codes not found in both files
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)]
df_with_velocities = compute_all_predicted_velocities(filtered_df, exclude_plate_codes)
```

In [13]:
```
# Data to mm/yr
df_with_velocities['Ve'] = df_with_velocities['Ve'] * 1000 * SECONDS_PER_YEAR
df_with_velocities['Vn'] = df_with_velocities['Vn'] * 1000 * SECONDS_PER_YEAR
df_with_velocities['Vu'] = df_with_velocities['Vu'] * 1000 * SECONDS_PER_YEAR
```

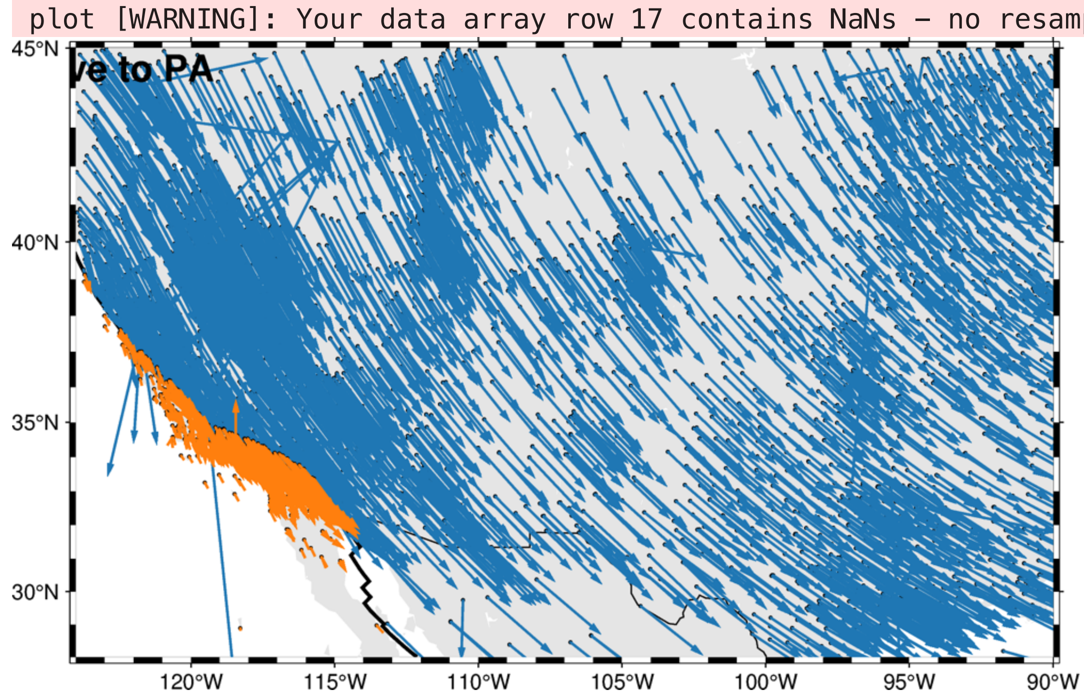
In [19]:
```
import pygmt
import pandas as pd
import numpy as np
import geopandas as gpd
from shapely.geometry import box
import matplotlib.colors as mcolors
import matplotlib.cm as cm

#  Define region (here california and us)
region = [-124, -90, 28, 45]

#  Define plate colors
unique_plates = df_with_velocities["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)}

scale_factor = 0.03

#  Compute velocity differences
# 1) Relative to NA
df_with_velocities["angle_deg_NA"] = np.degrees(np.arctan2(
    df_with_velocities["Vn"] - df_with_velocities["NA_vn_pred"],
    df_with_velocities["Ve"] - df_with_velocities["NA_ve_pred"]
))
df_with_velocities["mag_NA"] = np.hypot(
    df_with_velocities["Ve"] - df_with_velocities["NA_ve_pred"],
    df_with_velocities["Vn"] - df_with_velocities["NA_vn_pred"]
) * scale_factor

# 2) Relative to PA
df_with_velocities["angle_deg_PA"] = np.degrees(np.arctan2(
    df_with_velocities["Vn"] - df_with_velocities["PA_vn_pred"],
    df_with_velocities["Ve"] - df_with_velocities["PA_ve_pred"]
))
df_with_velocities["mag_PA"] = np.hypot(
    df_with_velocities["Ve"] - df_with_velocities["PA_ve_pred"],
    df_with_velocities["Vn"] - df_with_velocities["PA_vn_pred"]
) * scale_factor

# 3) Using Ve, Vn only
df_with_velocities["angle_deg_ITRF"] = np.degrees(np.arctan2(
    df_with_velocities["Vn"],
    df_with_velocities["Ve"]
))
df_with_velocities["mag_ITRF"] = np.hypot(
    df_with_velocities["Ve"],
    df_with_velocities["Vn"]
) * scale_factor

#  Load plate boundaries and crop to the region
plate_boundaries = gpd.read_file("data/tectonicplates-master/PB2002_boundaries.shp")
bbox = box(*region)
plate_boundaries_in_region = plate_boundaries[plate_boundaries.intersects(bbox)]

lines = []
```

```
for geom in plate_boundaries_in_region.geometry:
    if geom is None:
        continue
    if geom.geom_type == 'LineString':
        x, y = geom.xy
        lines.append(pd.DataFrame({'lon': x, 'lat': y}))
        # Insert NaNs to separate segments
        lines.append(pd.DataFrame({'lon': [np.nan], 'lat': [np.nan]}))
    elif geom.geom_type == 'MultiLineString':
        for line in geom.geoms:
            x, y = line.xy
            lines.append(pd.DataFrame({'lon': x, 'lat': y}))
            lines.append(pd.DataFrame({'lon': [np.nan], 'lat': [np.nan]}))
if lines:
    boundary_df = pd.concat(lines, ignore_index=True)

#  Function to plot a map for a given velocity field
def plot_full_map(title, angle_column, mag_column):
    fig = pygmt.Figure()
    fig.basemap(region=region, projection="M15c", frame=True)
    fig.coast(land="gray90", water="white", borders="1/0.5p", resolution="l")

    # Plot plate boundaries
    if lines:
        fig.plot(data=boundary_df, pen="1.5p,black")

    # For each plate, plot station locations and velocity vectors
    for plate in unique_plates:
        df_plate = df_with_velocities[df_with_velocities["plate_code"] == plate]
        color = plate_color_map[plate]

        # Plot station locations
        fig.plot(
            x=df_plate["lon"],
            y=df_plate["lat"],
            style="c0.05c",
            fill=color,
            pen="black"
        )

        # Plot velocity vectors
        fig.plot(
            x=df_plate["lon"],
            y=df_plate["lat"],
            style="v0.2c+eA",
            direction=[df_plate[angle_column], df_plate[mag_column]],
            pen=f"1p,{color}",
            fill=color
        )

    fig.text(text=title, x=region[0] + 0.5, y=region[3] - 0.5, font="16p,Helvetica-Bold,black")
    fig.show()

#  Plot figures
plot_full_map("Relative to NA", "angle_deg_NA", "mag_NA")
plot_full_map("Relative to PA", "angle_deg_PA", "mag_PA")
plot_full_map("Ve, Vn only (ITRF)", "angle_deg_ITRF", "mag_ITRF")
```

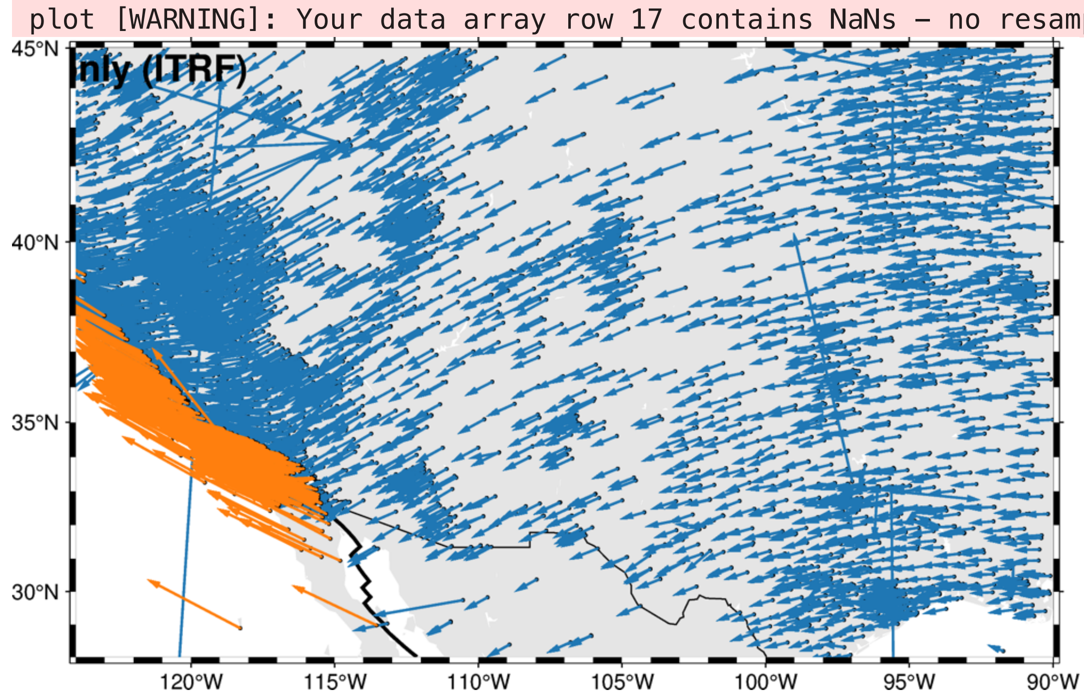
```
/var/folders/6l/7c4lc3h945qclbjjyb4wkqy80000gn/T/ipykernel_86140/1859130726.py:15: MatplotlibDeprecationW
arning: The get_cmap function was deprecated in Matplotlib 3.7 and will be removed in 3.11. Use ``matplot
lib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap()`` or ``pyplot.get_cmap()`` instead.
  colormap = cm.get_cmap("tab10", n_colors)
plot [WARNING]: Your data array row 17 contains NaNs - no resampling taken place!
```




```
plot [WARNING]: Your data array row 17 contains NaNs - no resampling taken place!
```




```
plot [WARNING]: Your data array row 17 contains NaNs - no resampling taken place!
```



Comment on how far the deformation zone of the plate boundary reaches. If you do not find velocities that approach near zero, can this be solved by taking a site at a larger distance?

I would imagine the deformation zone of the plate boundary is where the velocities relative to the plate motion of the opposite plate has the highest velocities, thus the longest arrows. This is from the plate boundary to around 115W and from 33N to 43N (see figure 1 above). If we dont see velocities approaching zero, it is likely because we have not yet reached a stable enough part of the NA or PA plate. We see very little velocities from 115W on eatwards (comparing wrt NA plate), with a few clusters shown. This can be seen as stable. However, as you move farther away to find more stable regions, the station's measured velocity will be influenced by other plate boundaries near it, so that would not be solved.

A similar analysis can be found when comparing to the PA plate, but as there are fewer stations on that side due to the presence of the Pacific ocean, there is less data available to commend. However, we see that in the second figure the velocities on the outskirt of the orange arrows are smaller.