Imports

In [5]:
```
import numpy as np

import pandas as pd
import datetime as dt
from datetime import datetime
import xarray as xr

import matplotlib.pyplot as plt
from matplotlib import cm
import matplotlib.dates as mdates
import cartopy.crs as ccrs

import random
import decimal
import aerobulk
from aerobulk.flux import noskin_np, skin_np, noskin, skin
```

Create Data

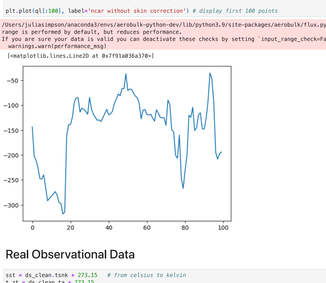
In [6]:
```
ds = xr.load_dataset('../../pl-fluxes/data/fluxes_all_cruises_compilation.nc')
ds_clean = ds.dropna(dim="time", how="any",
                     subset=['tauxx','tauyy','hsc','hlc','U','tsnk','ta','qa'])
```

```
/Users/juliasimpson/anaconda3/envs/aerobulk-python-dev/lib/python3.9/site-packages/xarray/coding/times.py:254: RuntimeWarning: invalid value encountered in cast
  flat_num_dates_ns_int = (flat_num_dates * _NS_PER_TIME_DELTA[delta]).astype(
/Users/juliasimpson/anaconda3/envs/aerobulk-python-dev/lib/python3.9/site-packages/xarray/coding/times.py:254: RuntimeWarning: invalid value encountered in cast
  flat_num_dates_ns_int = (flat_num_dates * _NS_PER_TIME_DELTA[delta]).astype(
```

In [7]:
```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1, projection=ccrs.Robinson())
ax.set_global()
ax.stock_img()
ax.coastlines()
colors = cm.tab10(np.linspace(0, 1, 10))

ax.scatter(ds_clean.lon,ds_clean.lat,s=ds_clean.tsnk,transform=ccrs.PlateCarree())
```

Out[7]:
```
<matplotlib.collections.PathCollection at 0x7f919288f730>
```

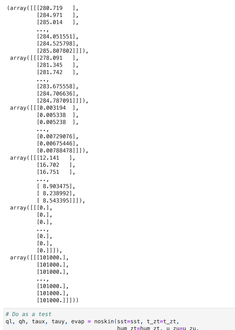


In [8]:
```
# Do as a test
ql, qh, taux, tauy, evap = noskin(sst=ds_clean.tsnk + 273.15, t_zt=ds_clean.ta + 273.15,
                                  hum_zt=ds_clean.qa/1000., u_zu=ds_clean.U,
                                  v_zu=ds_clean.U*0, slp=ds_clean.U/ds_clean.U*101000.0, algo='ncar', zt=ds_clean.zt.to_numpy(),
                                  zu=ds_clean.zu.to_numpy())

plt.plot(ql[:100], label='ncar without skin correction!') # display first 100 points
```

```
/Users/juliasimpson/anaconda3/envs/aerobulk-python-dev/lib/python3.9/site-packages/aerobulk/flux.py:280: UserWarning: Checking for misaligned nans and values outside of the valid
range is performed by default, but reduces performance.
If you are sure your data is valid you can deactivate these checks by setting `input_range_check=False`
  warnings.warn(performance_msg)
```

Out[8]:
```
[<matplotlib.lines.Line2D at 0x7f91a036a370>]
```



Real Observational Data

In [9]:
```
sst = ds_clean.tsnk + 273.15   # from celsius to kelvin
t_zt = ds_clean.ta + 273.15
hum_zt = ds_clean.qa / 1000.  # from g/kg to kg/kg
u_zu = ds_clean.U
v_zu = ds_clean.U * 0
zt = ds_clean.zt.to_numpy()
zu = ds_clean.zu.to_numpy()
slp=ds_clean.U/ds_clean.U*101000.0
```

In [10]:
```
ocean_index = np.where(~np.isnan(sst))
# Shrink the input data (i.e, remove all land points)
args_shrunk = tuple(np.atleast_3d(a[ocean_index]) for a in (sst, t_zt, hum_zt, u_zu, v_zu, slp))
```

In [11]:
```
ocean_index
```

Out[11]:
```
(array([    0,     1,     2, ..., 10076, 10077, 10078]),)
```

In [12]:
```
args_shrunk
```

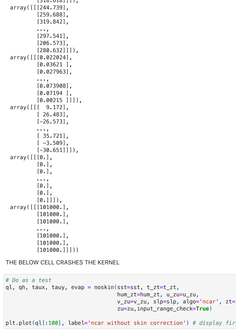
Out[12]:
```
(array([[[280.719   ],
         [284.971   ],
         [285.014   ],
         ...,
         [284.455551],
         [284.525798],
         [285.007802]]],
       dtype=float32),
 array([[[278.891   ],
         [281.345   ],
         [281.742   ],
         ...,
         [283.675558],
         [284.706636],
         [284.787091]]],
       dtype=float32),
 array([[[0.003194  ],
         [0.005338  ],
         [0.005238  ],
         ...,
         [0.00729076],
         [0.006754461],
         [0.007884781]]],
       dtype=float32),
 array([[[12.141   ],
         [16.702   ],
         [16.751   ],
         ...,
         [ 8.980475],
         [ 8.238992],
         [ 8.543395]]]],
       dtype=float32),
 array([[[0.],
         [0.],
         [0.],
         ...,
         [0.],
         [0.],
         [0.]]]),
 array([[[1101000.],
         [101000.],
         [101000.],
         ...,
         [101000.],
         [101000.],
         [101000.]]]))
```

In [13]:
```
# Do as a test
ql, qh, taux, tauy, evap = noskin(sst=sst, t_zt=t_zt,
                                  hum_zt=hum_zt, u_zu=u_zu,
                                  v_zu=v_zu, slp=slp, algo='ncar', zt=zt,
                                  zu=zu,input_range_check=True)

plt.plot(ql[:100], label='ncar without skin correction!') # display first 100 points
```

```
/Users/juliasimpson/anaconda3/envs/aerobulk-python-dev/lib/python3.9/site-packages/aerobulk/flux.py:280: UserWarning: Checking for misaligned nans and values outside of the valid
range is performed by default, but reduces performance.
If you are sure your data is valid you can deactivate these checks by setting `input_range_check=False`
  warnings.warn(performance_msg)
```

Out[13]:
```
[<matplotlib.lines.Line2D at 0x7f91703993c40>]
```



Synthetic Data

In [14]:
```
# Valid data range according to documentation
# https://github.com/xgcm/aerobulk-python/blob/main/source/aerobulk/flux.py
VALID_VALUE_RANGES = {'sst': [270, 320],
                      't_zt': [180, 330],
                      'hum_zt': [0, 0.08],
                      'u_zu': [-50, 50],
                      'v_zu': [-50, 50],
                      'slp': [80000, 110000],
                      'rad_sw': [0, 1500],
                      'rad_lw': [0, 750]
                     }

sst = []
t_zt = []
hum_zt = []
u_zu = []
v_zu = []
slp = []
zt = []
zu = []

data_points = 10000

#FINITAL: don't specify any kind of distribution. Later, look at choices/ways in random to specify
#Later, also experiment with specifying a random difference, random sst, and adding for air temp
for data_point in range (0, data_points):
    sst.append(float(decimal.Decimal(random.randrange(VALID_VALUE_RANGES['sst'][0]*1000, VALID_VALUE_RANGES['sst'][1]*1000))/1000)) #did *10 and then /10 to get 0.1 precision
    t_zt.append(float(decimal.Decimal(random.randrange(VALID_VALUE_RANGES['t_zt'][0]*1000, VALID_VALUE_RANGES['t_zt'][1]*1000))/1000))
    #t_zt.append(sst[data_point]+10) #instead of 10 do a random range of difference between 270-180 and 320-330
    hum_zt.append(float(decimal.Decimal(random.randrange(VALID_VALUE_RANGES['hum_zt'][0]*1000000, VALID_VALUE_RANGES['hum_zt'][1]*1000000))/1000000))
    u_zu.append(float(decimal.Decimal(random.randrange(VALID_VALUE_RANGES['u_zu'][0]*1000, VALID_VALUE_RANGES['u_zu'][1]*1000))/1000))
    v_zu.append(float(0))
    slp.append(101000.0)

    zt.append(float(decimal.Decimal(random.randrange(12*10,19*10))/10)) #use 12 to 19, mirroring other dataset
    zu.append(float(decimal.Decimal(random.randrange(15*10,22*10))/10)) #use 15 to 22, mirroring other dataset
```

In [15]:
```
time = ds.time[:10000].data

ds_new = xr.Dataset(
    data_vars=dict(
        sst=("time",  sst),
        t_zt=("time", t_zt),
        hum_zt=("time"), hum_zt),
        u_zu=("time", u_zu),
        v_zu=("time"), v_zu),
        slp=("time", slp),
        zt=("time"), zt),
        zu=("time"), zu),
    ),
    coords=dict(
        time=time
    ),
    attrs=dict(description="Organizing synthetic data into a set."),
)
```

In [16]:
```
sst = ds_new.sst
t_zt = ds_new.t_zt
hum_zt = ds_new.hum_zt
u_zu = ds_new.u_zu
v_zu = ds_new.v_zu
slp = ds_new.slp
zt = ds_new.zt.to_numpy()
zu = ds_new.zu.to_numpy()

# Also tried the below, to completely mimic the real data
#sst = ds_new.sst
#t_zt = ds_new.t_zt
#hum_zt = ds_new.hum_zt
#u_zu = ds_new.u_zu
#v_zu = ds_new.v_zu = 0
#zt = ds_new.zt.to_numpy()
#zu = ds_new.zu.to_numpy()
#slp=ds_new.u_zu/ds_new.o_zu*101000.0
```

In [17]:
```
ocean_index = np.where(~np.isnan(sst))
# Shrink the input data (i.e, remove all land points)
args_shrunk = tuple(np.atleast_3d(a[ocean_index]) for a in (sst, t_zt, hum_zt, u_zu, v_zu, slp))
```

In [18]:
```
ocean_index
```

Out[18]:
```
(array([   0,    1,    2, ..., 9997, 9998, 9999]),)
```

In [19]:
```
args_shrunk
```

Out[19]:
```
(array([[[303.514],
         [305.133],
         [283.737],
         ...,
         [314.369],
         [316.807],
         [318.618]]]),
 array([[[264.739],
         [259.688],
         [319.842],
         ...,
         [297.561],
         [286.573],
         [280.632]]]),
 array([[[0.022024],
         [0.03621 ],
         [0.027963],
         ...,
         [0.073988],
         [0.07194 ],
         [0.00215 ]]]),
 array([[[  9.172],
         [ 26.483],
         [-26.573],
         ...,
         [ 35.721],
         [ -3.589],
         [-30.655]]]),
 array([[[0.],
         [0.],
         [0.],
         ...,
         [0.],
         [0.],
         [0.]]]),
 array([[[101000.],
         [101000.],
         [101000.],
         ...,
         [101000.],
         [101000.],
         [101000.]]]))
```

THE BELOW CELL CRASHES THE KERNEL

In [ ]:
```
# Do as a test
ql, qh, taux, tauy, evap = noskin(sst=sst, t_zt=t_zt,
                                  hum_zt=hum_zt, u_zu=u_zu,
                                  v_zu=v_zu, slp=slp, algo='ncar', zt=zt,
                                  zu=zu,input_range_check=True)

plt.plot(ql[:100], label='ncar without skin correction!') # display first 100 points
```

In [ ]:
```

```