Project2.0-3

September 10, 2021

[1]:
```python
import netCDF4 as nc
import numpy as np
from matplotlib import pyplot as plt
import xarray
import glob
from geopy.distance import geodesic
import datetime as dt
import matplotlib as mpl
from matplotlib.ticker import (MultipleLocator, AutoMinorLocator)
```

0.0.1 Funktionen zum einlesen von Daten

[2]:
```python
time_threshold = 3600 # in seconds
distance_threshold = 50 # in km
```

[3]:
```python
def cut_useless_variables(DS):
    """
    Preprocessing the IASI dataset before concatenation.
    Removing undesired dimensions and variables.

    Parameters:
    -----------
    ds : xarray dataset
        Dataset of IASI data.
    """

    # Remove some nasty variables:
    """
    Remaining variables are:
        pressure_levels_temp
        pressure_levels_humidity
        record_start_time
        record_stop_time
        lat
        lon
        atmospheric_temperature
        atmospheric_water_vapor
```

```
        surface_temperature
        surface_pressure
        instrument_mode
        flag_cldnes
        flag_iasibad
        flag_itconv
        flag_landsea
        error_data_index
            # temperature_error
            # water_vapour_error
        surface_z        (useful in combination with surface_pressure)
        co_qflag
        co_bdiv

    Remaining dimensions:
        nlt
        nlq
        along_track
        across_track
            # nerr
            # nerrt
            # nerrw
    """
    useless_vars = ['cloud_formation', 'pressure_levels_ozone',␣
↪'surface_emissivity_wavelengths',
                    'degraded_ins_MDR', 'degraded_proc_MDR', 'solar_zenith',␣
↪'satellite_zenith',
                    'solar_azimuth', 'satellite_azimuth',␣
↪'fg_atmospheric_temperature',
                    'fg_atmospheric_water_vapor', 'fg_atmospheric_ozone',␣
↪'fg_surface_temperature',
                    'atmospheric_ozone', 'integrated_water_vapor',␣
↪'integrated_ozone', 'integrated_n2o',
                    'integrated_co', 'integrated_ch4', 'integrated_co2',␣
↪'surface_emissivity',
                    'number_cloud_formations', 'fractional_cloud_cover',␣
↪'cloud_top_temperature',
                    'cloud_top_pressure', 'cloud_phase', 'spacecraft_altitude',␣
↪'flag_amsubad',
                    'flag_avhrrbad', 'flag_cdlfrm', 'flag_cdltst',␣
↪'flag_daynit', 'flag_dustcld',
                    'flag_fgcheck', 'flag_initia', 'flag_mhsbad', 'flag_numit',␣
↪'flag_nwpbad',
                    'flag_physcheck', 'flag_retcheck', 'flag_satman',␣
↪'flag_sunglnt', 'flag_thicir',
```

```
                    'nerr_values', 'ozone_error', 'co_npca', 'co_nfitlayers',␣
↪'co_nbr_values',
                    'co_cp_air', 'co_cp_co_a', 'co_x_co', 'co_h_eigenvalues',␣
↪'co_h_eigenvectors',
                    'temperature_error', 'water_vapour_error']
    DS = DS.drop_vars(useless_vars)

    # useless_dims = ['npct', 'npcw', 'npco', 'nl_co', 'nl_hno3', 'nl_o3',␣
↪'nl_so2', 'new', 'nlo',
                    # 'cloud_formations', 'nerro', 'co_nbr', 'neva_co',␣
↪'neve_co']
    # DS = DS.squeeze(useless_dims, drop=True)

    return DS
```

[4]:

```
def import_single_NYA_RS_radiosonde(
        filename,
        keys='all',
        verbose=0):

        """
        Imports single NYA-RS radiosonde data for Ny Alesund. Converts to SI␣
↪units
        and interpolates to a height grid with 5 m resolution from 0 to 15000 m.
↪

        Parameters:
        -----------
        filename : str
                Name (including path) of radiosonde data file.
        keys : list of str or str, optional
                This describes which variable(s) will be loaded. Specifying␣
↪'all' will import all variables.
                Specifying 'basic' will load the variables the author␣
↪consideres most useful for his current
                analysis.
                Default: 'all'
        verbose : int
                If 0, output is suppressed. If 1, basic output is printed. If␣
↪2, more output (more warnings,...)
                is printed.
        """

        """
                Loaded values are imported in the following units:
                T: in K
```

```
                P: in hPa, will be converted to Pa
                RH: in [0-1]
                Altitude: in m
                time: will be converted to sec since 1970-01-01 00:00:00 UTC
        """

        file_nc = nc.Dataset(filename)

        if (not isinstance(keys, str)) and (not isinstance(keys, list)):
                raise TypeError("Argument 'key' must be a list of strings or␣
↪'all'.")

        if keys == 'all':
                keys = file_nc.variables.keys()
        elif keys == 'basic':
                keys = ['time', 'temp', 'press', 'rh', 'alt']

        sonde_dict = dict()
        for key in keys:
                if not key in file_nc.variables.keys():
                        raise KeyError("I have no memory of this key: '%s'. Key␣
↪not found in radiosonde file." % key)

                sonde_dict[key] = np.asarray(file_nc.variables[key])
                if key != "IWV" and len(sonde_dict[key]) == 0: # 'and': second␣
↪condition only evaluated if first condition True
                        return None

                if key in ['lat', 'lon']:        # only interested in the first␣
↪lat, lon position
                        sonde_dict[key] = sonde_dict[key][0]

        # convert units:
        if 'P' in keys:                 # from hPa to Pa
                sonde_dict['P'] = sonde_dict['P']*100
        if 'time' in keys:        # from int64 to float64
                time_unit = file_nc.variables['time'].units
                time_offset = (dt.datetime.strptime(time_unit[-19:],␣
↪"%Y-%m-%dT%H:%M:%S") - dt.datetime(1970,1,1)).total_seconds()
                sonde_dict['time'] = np.float64(sonde_dict['time']) +␣
↪time_offset
                sonde_dict['launch_time'] = sonde_dict['time'][0]

        keys = [*keys]                  # converts dict_keys to a list
        for key in keys:
                if sonde_dict[key].shape == sonde_dict['time'].shape:
```

```
                        if key not in ['time', 'lat', 'lon', 'alt']:
                                sonde_dict[key + "_ip"] = np.interp(np.
↪arange(0,15001,5), sonde_dict['alt'], sonde_dict[key])
                        elif key == 'alt':
                                sonde_dict[key + "_ip"] = np.arange(0, 15001,5)


        # Renaming variables to a standard convention
        renaming = {'press': 'pres', 'alt': 'height', 'press_ip': 'pres_ip',␣
↪'alt_ip': 'height_ip'}
        for ren_key in renaming.keys():
                if ren_key in sonde_dict.keys():
                        sonde_dict[renaming[ren_key]] = sonde_dict[ren_key]

        return sonde_dict


def import_single_PS122_mosaic_radiosonde_level2(
        filename,
        keys='all',
        verbose=0):

        """
        Imports single level 2 radiosonde data created with PANGAEA_tab_to_nc.py
        ('PS122_mosaic_radiosonde_level2_yyyymmdd_hhmmssZ.nc'). Converts to SI␣
↪units
        and interpolates to a height grid with 5 m resolution from 0 to 15000 m.

        Parameters:
        -----------
        filename : str
                Name (including path) of radiosonde data file.
        keys : list of str or str, optional
                This describes which variable(s) will be loaded. Specifying␣
↪'all' will import all variables.
                Specifying 'basic' will load the variables the author␣
↪consideres most useful for his current
                analysis.
                Default: 'all'
        verbose : int
                If 0, output is suppressed. If 1, basic output is printed. If␣
↪2, more output (more warnings,...)
                is printed.
        """

        """
```

```
                Loaded values are imported in the following units:
                T: in deg C, will be converted to K
                P: in hPa, will be converted to Pa
                RH: in %, will be converted to [0-1]
                Altitude: in m
                q: in kg kg^-1 (water vapor specific humidity)
                time: in sec since 1970-01-01 00:00:00 UTC
        """

        file_nc = nc.Dataset(filename)

        if (not isinstance(keys, str)) and (not isinstance(keys, list)):
                raise TypeError("Argument 'key' must be a list of strings or␣
↪'all'.")

        if keys == 'all':
                keys = file_nc.variables.keys()
        elif keys == 'basic':
                keys = ['time', 'T', 'P', 'RH', 'q', 'Altitude']

        sonde_dict = dict()
        for key in keys:
                if not key in file_nc.variables.keys():
                        raise KeyError("I have no memory of this key: '%s'. Key␣
↪not found in radiosonde file." % key)

                sonde_dict[key] = np.asarray(file_nc.variables[key])
                if key != "IWV" and len(sonde_dict[key]) == 0: # 'and': second␣
↪condition only evaluated if first condition True
                        return None

                if key in ['Latitude', 'Longitude']:        # only interested␣
↪in the first lat, lon position
                        sonde_dict[key] = sonde_dict[key][0]
                if key == 'IWV':
                        sonde_dict[key] = np.float64(sonde_dict[key])

        # convert units:
        if 'RH' in keys:        # from percent to [0, 1]
                sonde_dict['RH'] = sonde_dict['RH']*0.01
        if 'T' in keys:                 # from deg C to K
                sonde_dict['T'] = sonde_dict['T'] + 273.15
        if 'P' in keys:                 # from hPa to Pa
                sonde_dict['P'] = sonde_dict['P']*100
        if 'time' in keys:      # from int64 to float64
                sonde_dict['time'] = np.float64(sonde_dict['time'])
                sonde_dict['launch_time'] = sonde_dict['time'][0]
```

```
        keys = [*keys]                  # converts dict_keys to a list
        for key in keys:
                if sonde_dict[key].shape == sonde_dict['time'].shape:
                        if key not in ['time', 'Latitude', 'Longitude', 'ETIM',␣
 ↪'Altitude']:
                                sonde_dict[key + "_ip"] = np.interp(np.
 ↪arange(0,15001,5), sonde_dict['Altitude'], sonde_dict[key])
                        elif key == 'Altitude':
                                sonde_dict[key + "_ip"] = np.arange(0, 15001,5)


        # Renaming variables: ['Lat', 'Lon', 'p', 'T', 'RH', 'GeopHgt', 'qv',␣
 ↪'time', ...]
        renaming = {'T': 'temp',         'P': 'pres',         'RH': 'rh',
                                'Altitude': 'height', 'h_geom': 'height_geom',
                                'Latitude': 'lat',         'Longitude': 'lon',
                                'T_ip': 'temp_ip', 'P_ip': 'pres_ip', 'RH_ip':␣
 ↪'rh_ip',
                                'Altitude_ip': 'height_ip', 'h_geom_ip':␣
 ↪'height_geom_ip',
                                'IWV': 'iwv'}
        for ren_key in renaming.keys():
                if ren_key in sonde_dict.keys():
                        sonde_dict[renaming[ren_key]] = sonde_dict[ren_key]

        return sonde_dict

##!!!

def import_radiosonde_daterange(
        path_data,
        date_start,
        date_end,
        s_version='level_2',
        with_wind=False,
        verbose=0):

        """
        Imports radiosonde data 'mossonde-curM1' and concatenates the files␣
 ↪into time series x height.
        E.g. temperature profile will have the dimension: n_sondes x n_height

        Parameters:
        -----------
        path_data : str
```

```
                Path of radiosonde data.
        date_start : str
                Marks the first day of the desired period. To be specified in␣
↪yyyy-mm-dd (e.g. 2021-01-14)!
        date_end : str
                Marks the last day of the desired period. To be specified in␣
↪yyyy-mm-dd (e.g. 2021-01-14)!
        s_version : str, optional
                Specifies the radiosonde version that is to be imported.␣
↪Possible options: 'mossonde',
                'psYYMMDDwHH', 'level_2'. Default: 'level_2' (published by␣
↪Marion Maturilli)
        with_wind : bool, optional
                This describes if wind measurements are included (True) or not␣
↪(False). Does not work with
                s_version='psYYMMDDwHH'. Default: False
        verbose : int, optional
                If 0, output is suppressed. If 1, basic output is printed. If␣
↪2, more output (more warnings,...)
                is printed.
        """

        if not isinstance(s_version, str): raise TypeError("s_version in␣
↪import_radiosonde_daterange must be a string.")

        # extract day, month and year from start date:
        date_start = dt.datetime.strptime(date_start, "%Y-%m-%d")
        date_end = dt.datetime.strptime(date_end, "%Y-%m-%d")

        if s_version == 'level_2':
                all_radiosondes_nc = sorted(glob.glob(path_data +␣
↪"PS122_mosaic_radiosonde_level2*.nc"))

                # inquire the number of radiosonde files (date and time of␣
↪launch is in filename):
                # And fill a list which will include the relevant radiosonde␣
↪files.
                radiosondes_nc = []
                for rs_nc in all_radiosondes_nc:
                        rs_date = rs_nc[-19:-3]                 # date of␣
↪radiosonde from filename
                        yyyy = int(rs_date[:4])
                        mm = int(rs_date[4:6])
                        dd = int(rs_date[6:8])
                        rs_date_dt = dt.datetime(yyyy,mm,dd)
                        if rs_date_dt >= date_start and rs_date_dt <= date_end:
```

```
                                radiosondes_nc.append(rs_nc)

        elif s_version == 'nya-rs':
                all_radiosondes_nc = sorted(glob.glob(path_data + "NYA-RS_*.
↪nc"))

                # inquire the number of radiosonde files (date and time of␣
↪launch is in filename):
                # And fill a list which will include the relevant radiosonde␣
↪files.
                radiosondes_nc = []
                for rs_nc in all_radiosondes_nc:
                        rs_date = rs_nc[-15:-3]                # date of␣
↪radiosonde from filename
                        yyyy = int(rs_date[:4])
                        mm = int(rs_date[4:6])
                        dd = int(rs_date[6:8])
                        rs_date_dt = dt.datetime(yyyy,mm,dd)
                        if rs_date_dt >= date_start and rs_date_dt <= date_end:
                                radiosondes_nc.append(rs_nc)


        # number of sondes:
        n_sondes = len(radiosondes_nc)

        # count the number of days between start and end date as max. array␣
↪size:
        n_days = (date_end - date_start).days

        # basic variables that should always be imported:
        if s_version == 'level_2':
                geoinfo_keys = ['lat', 'lon', 'launch_time', 'iwv']
                time_height_keys = ['pres', 'temp', 'rh', 'height', 'rho_v',␣
↪'q']
                if with_wind: time_height_keys = time_height_keys + ['wspeed',␣
↪'wdir']

        elif s_version == 'nya-rs':
                geoinfo_keys = ['lat', 'lon', 'launch_time']
                time_height_keys = ['pres', 'temp', 'rh', 'height']
                if with_wind: time_height_keys = time_height_keys + ['wspeed',␣
↪'wdir']
        else:
                raise ValueError("s_version in import_radiosonde_daterange must␣
↪be 'nya-rs' or 'level_2'.")
        all_keys = geoinfo_keys + time_height_keys
```

```
	# sonde_master_dict (output) will contain all desired variables on specific axes:
	# Time axis (one sonde = 1 timestamp) = axis 0; height axis = axis 1
	n_height = len(np.arange(0,15001,5))		# length of the interpolated height grid
	sonde_master_dict = dict()
	for gk in geoinfo_keys: sonde_master_dict[gk] = np.full((n_sondes,), np.nan)
	for thk in time_height_keys: sonde_master_dict[thk] = np.full((n_sondes, n_height), np.nan)

	if s_version == 'level_2':
		all_keys_import = ['Latitude', 'Longitude', 'P', 'T', 'RH', 'Altitude', 'rho_v', 'q', 'time', 'IWV']
		if with_wind: all_keys_import = all_keys_import + ['wdir', 'wspeed']


		# cycle through all relevant sonde files:
		for rs_idx, rs_nc in enumerate(radiosondes_nc):

			if verbose >= 1:
				# rs_date = rs_nc[-19:-3]
				print("Working on Radiosonde, " + rs_nc)

			sonde_dict = import_single_PS122_mosaic_radiosonde_level2(rs_nc, keys=all_keys_import)

			# save to sonde_master_dict:
			for key in all_keys:
				if key in geoinfo_keys:
					sonde_master_dict[key][rs_idx] = sonde_dict[key]

				elif key in time_height_keys:
					sonde_master_dict[key][rs_idx, :] = sonde_dict[key + "_ip"]		# must use the interpolated versions!

				else:
					raise KeyError("Key '" + key + "' not found in radiosonde dictionary after importing it with " +
											"import_single_PS122_m

	if s_version == 'nya-rs':
```

```
                all_keys_import = ['lat', 'lon', 'press', 'temp', 'rh', 'alt',␣
↪'time']
                if with_wind: all_keys_import = all_keys_import + ['wdir',␣
↪'wspeed']


                # cycle through all relevant sonde files:
                for rs_idx, rs_nc in enumerate(radiosondes_nc):

                        if verbose >= 1:
                                # rs_date = rs_nc[-19:-3]
                                print("Working on Radiosonde, " + rs_nc)

                        sonde_dict = import_single_NYA_RS_radiosonde(rs_nc,␣
↪keys=all_keys_import)

                        # save to sonde_master_dict:
                        for key in all_keys:
                                if key in geoinfo_keys:
                                        sonde_master_dict[key][rs_idx] =␣
↪sonde_dict[key]

                                elif key in time_height_keys:
                                        sonde_master_dict[key][rs_idx, :] =␣
↪sonde_dict[key + "_ip"]                # must use the interpolated versions!

                                else:
                                        raise KeyError("Key '" + key + "' not␣
↪found in radiosonde dictionary after importing it with " +
                                                                                                "import_single_NYA_RS_

        return sonde_master_dict
```

```
[5]: def numpydatetime64_to_epochtime(npdt_array):

    """
    Converts numpy datetime64 array to array in seconds since 1970-01-01 00:00:
↪00 UTC (type:
    float).

    Parameters:
    -----------
    npdt_array : numpy array of type np.datetime64 or np.datetime64 type
        Array (1D) or directly a np.datetime64 type variable.
    """
```

```
    sec_epochtime = npdt_array.astype(np.timedelta64) / np.timedelta64(1, 's')

    return sec_epochtime
```

[6]:
```
def create_launch_time(DS):

    time_dif = np.diff(DS.time.values)
    where_jump = np.argwhere(np.abs(time_dif) > time_threshold).flatten()
    launch_time = np.concatenate((np.array([DS.time.values[0]]), DS.time.
 ↪values[where_jump+1]))

    return xarray.DataArray(launch_time)
```

0.0.2 Daten einlesen

[7]:
```
#NyAlesund
path_data = "/Users/charlottebaur/Desktop/Project work/NyAlesund/"
NyAl_sonde_dict = import_radiosonde_daterange(path_data,␣
 ↪date_start="2020-08-01", date_end="2020-09-30", s_version='nya-rs',␣
 ↪with_wind=False)

#IASI
IASI_DS = xarray.open_mfdataset('/Users/charlottebaur/Desktop/Project work/
 ↪METRS_IASI/*.nc',concat_dim='along_track', combine='nested',␣
 ↪decode_times=False, preprocess=cut_useless_variables)

#Polarstern
path_data = "/Users/charlottebaur/Desktop/Project work/Polarstern/"
date_start = "2020-08-01"                # in yyyy-mm-dd
date_end = "2020-09-30"                        # in yyyy-mm-dd
s_version = 'level_2'

PS_sonde_dict = import_radiosonde_daterange(path_data, date_start, date_end,␣
 ↪s_version, with_wind=False)
```

```
<ipython-input-4-3e5033df8d41>:59: DeprecationWarning: tostring() is deprecated.
Use tobytes() instead.
  time_unit = file_nc.variables['time'].units
/Users/charlottebaur/opt/anaconda3/lib/python3.8/site-
packages/xarray/conventions.py:512: SerializationWarning: variable
'temperature_error' has multiple fill values {-2147483646, 4294967295}, decoding
all values to NaN.
  new_vars[k] = decode_cf_variable(
/Users/charlottebaur/opt/anaconda3/lib/python3.8/site-
packages/xarray/conventions.py:512: SerializationWarning: variable
'water_vapour_error' has multiple fill values {-2147483646, 4294967295},
decoding all values to NaN.
```

```
  new_vars[k] = decode_cf_variable(
/Users/charlottebaur/opt/anaconda3/lib/python3.8/site-
packages/xarray/conventions.py:512: SerializationWarning: variable 'ozone_error'
has multiple fill values {-2147483646, 4294967295}, decoding all values to NaN.
  new_vars[k] = decode_cf_variable(
/Users/charlottebaur/opt/anaconda3/lib/python3.8/site-
packages/xarray/conventions.py:512: SerializationWarning: variable 'surface_z'
has multiple fill values {-32767, 32767}, decoding all values to NaN.
  new_vars[k] = decode_cf_variable(
/Users/charlottebaur/opt/anaconda3/lib/python3.8/site-
packages/xarray/conventions.py:512: SerializationWarning: variable 'co_cp_air'
has multiple fill values {-2, 65535}, decoding all values to NaN.
  new_vars[k] = decode_cf_variable(
/Users/charlottebaur/opt/anaconda3/lib/python3.8/site-
packages/xarray/conventions.py:512: SerializationWarning: variable 'co_cp_co_a'
has multiple fill values {-2, 65535}, decoding all values to NaN.
  new_vars[k] = decode_cf_variable(
```

0.0.3 Zeit filtern

[8]:
```
#IASI in Sekunden seit 01.01.2000 > umwandeln in Zeit wie bei NyAlesund

record_start_time = np.zeros(IASI_DS['record_start_time'].shape)
time_diff = (dt.datetime(2000,1,1) - dt.datetime(1970,1,1)).total_seconds()
n_time = IASI_DS['record_start_time'].shape[0]
iasi_record_start_time = IASI_DS.record_start_time.values
n_time_print = int(0.15*n_time)
for i in range(n_time):
    if i%n_time_print == 0: print(i)
    record_start_time[i] = iasi_record_start_time[i] + time_diff


IASI_DS['record_start_time'] = xarray.DataArray(record_start_time,␣
 ↪dims=['along_track'])
```

```
0
14857
29714
44571
59428
74285
89142
```

[9]:
```
n_time_NyAl = len(NyAl_sonde_dict['launch_time'])
time_idx = np.full((n_time_NyAl,),np.nan)
for idx in range(n_time_NyAl):
    if np.any(np.abs(record_start_time - NyAl_sonde_dict['launch_time'][idx])␣
 ↪<= time_threshold):
```

```
        time_idx[idx] = np.argmin(np.abs(record_start_time -␣
↪NyAl_sonde_dict['launch_time'][idx]))

time_idx = time_idx.astype(np.int32)
```

0.0.4 Koordinaten filtern

[10]:
```
coords_nyal = (78.924444, 11.928611)
```

[11]:
```
#Für jeden IASI Pixel (along und across track )
#<-> 2 for loops: berechne Distanz mit geopy Functions
n_along = len(IASI_DS.along_track)
n_across = len(IASI_DS.across_track)
distance_iasi_nyal = np.full((n_along, n_across), np.nan)
IASI_lon = IASI_DS.lon.values
IASI_lat = IASI_DS.lat.values
for j in range(n_across):
    if j%20 == 0: print(j)
    for i in range(n_along):
        if np.abs(IASI_lat[i,j]) <= 90:
            distance_iasi_nyal[i,j] = geodesic(coords_nyal, (IASI_lat[i,j],␣
↪IASI_lon[i,j])).km
        else:
            distance_iasi_nyal[i,j] = 999999.99

IASI_DS['distance_iasi_nyal'] = xarray.DataArray(distance_iasi_nyal,␣
↪dims=['along_track', 'across_track'])
```

```
0
20
40
60
80
100
```

[12]:
```
#NyAlesund from rh to IWP

# constants:
R_d = 287.04          # gas constant of dry air, in J kg^-1 K^-1
R_v = 461.5           # gas constant of water vapour, in J kg^-1 K^-1
M_dv = R_d / R_v # molar mass ratio , in ()
e_0 = 611                 # saturation water vapour pressure at freezing point␣
↪(273.15 K), in Pa
T0 = 273.15                 # freezing temperature, in K
g = 9.80665           # gravitation acceleration, in m s^-2

def e_sat(temp):
```

```
        """
        Calculates the saturation pressure over water after Goff and Gratch␣
↪(1946).
        It is the most accurate that you can get for a temperture range from␣
↪-90°C to +80°C.
        Source: Smithsonian Tables 1984, after Goff and Gratch 1946
        http://cires.colorado.edu/~voemel/vp.html
        http://hurri.kean.edu/~yoh/calculations/satvap/satvap.html

        e_sat_gg_water in Pa.

        Parameters:
        -----------
        temp : array of floats
                Array of temperature (in K).
        """

        e_sat_gg_water = 100 * 1013.246 * 10**(-7.90298*(373.16/temp-1) + 5.
↪02808*np.log10(
            373.16/temp) - 1.3816e-7*(10**(11.344*(1-temp/373.16))-1) + 8.
↪1328e-3 * (10**(-3.49149*(373.16/temp-1))-1))

        return e_sat_gg_water

#e_sat(NyAl_temp)

def convert_rh_to_spechum(
        temp,
        pres,
        relhum):

        """
        Convert array of relative humidity (between 0 and 1) to specific␣
↪humidity
        in kg kg^-1.

        Saturation water vapour pressure computation is based on: see␣
↪e_sat(temp).

        Parameters:
        -----------
        temp : array of floats
                Array of temperature (in K).
        pres : array of floats
                Array of pressure (in Pa).
```

```
        relhum : array of floats
                Array of relative humidity (between 0 and 1).
        """

        e_sat_water = e_sat(temp)

        e = e_sat_water * relhum
        q = M_dv * e / (e*(M_dv - 1) + pres)

        return q
```

[13]:
```
#n_time_ps = PS_sonde_dict['launch_time'].shape[0]
```

[14]:
```
#IASI_T zu radiosonden(PS)-pixel-selektieren

n_time_ps = len(PS_sonde_dict['launch_time'])

distance_iasi_ps = np.full((n_along, n_across), 999999.99)
record_start_time = IASI_DS.record_start_time.values
which_PS_sonde = np.full((n_along, n_across), np.nan)
for i in range(n_along):
        if i%1000 == 0: print(i)
        # find Polarstern radiosonde launch that has temporally closest to IASI␣
 ↪along track scan (pixel):
        launch_time_dif = np.abs(PS_sonde_dict['launch_time'] -␣
 ↪record_start_time[i])
        idx_ps_time = np.argmin(launch_time_dif)

        if np.abs(PS_sonde_dict['launch_time'][idx_ps_time] -␣
 ↪record_start_time[i]) <= 3600:
                # which_PS_sonde[i] = idx_ps_time
                for j in range(n_across):

                        if np.abs(IASI_lat[i,j]) <= 90:
                                disdis =␣
 ↪geodesic((PS_sonde_dict['lat'][idx_ps_time],␣
 ↪PS_sonde_dict['lon'][idx_ps_time]), (IASI_lat[i,j], IASI_lon[i,j])).km
                                if disdis < 50:
                                        which_PS_sonde[i,j] = idx_ps_time
                                        distance_iasi_ps[i,j] = disdis

IASI_DS['distance_iasi_ps'] = xarray.DataArray(distance_iasi_ps,␣
 ↪dims=['along_track', 'across_track'])

# remove nans from which_PS_sonde:
which_PS_sonde = which_PS_sonde.astype(np.int32)
```

```python
which_PS_sonde_nonnan = np.unique(which_PS_sonde[which_PS_sonde >= 0]) #␣
↪explained below



# what do we have now:
# - distance_iasi_ps (n_along, n_across): distance of IASI to Polarstern for␣
↪each IASI pixel (value is only non-nan if that pixel is temporally within␣
↪3600 sec of a
#         Polarstern radiosonde launch AND if it is within 50 km of Polarstern)
# - which_PS_sonde_nonnan (varying dimension): this now indicates which␣
↪Polarstern sondes have IASI pixels that are within 3600 sec of a sonde␣
↪launch and where IASI
#         pixel is within 50 km of Polarstern

# Later, we want to know if there is a IASI pixel for a given Polarstern launch␣
↪which is close enough (i.e. < 50 km) and within
# 3600 sec of that sonde launch. So, we would like to have an array with the␣
↪shape (n_sondes_ps,2) (or (n_time_ps,2)) that tells
# us the exact along track and across track coordinate of IASI that fulfills␣
↪these conditions.
# So, we can now run through all Polarstern sondes again, and check, if the␣
↪indicated along track coordinate has one or more IASI_DS['distance_iasi_ps']␣
↪< 50 km
iasi_pixels_for_nya = np.full((n_time_ps,2), 0)

ps_iasi_overlap_pixels = list()
for idx in range(n_time_ps):
    if idx in which_PS_sonde_nonnan: # then it's not a fill value and a IASI␣
↪pixel with time offset < 3600 sec exists for the current Polarstern sonde␣
↪launch:

               # find lines where which_PS_sonde is equal to idx:
        ps_iasi_overlap_pixels.append(np.argwhere(which_PS_sonde == idx))

"""
ps_iasi_overlap_pixels must always be considered together with␣
↪which_PS_sonde_nonnan:
Example: which_PS_sonde_nonnan[0] is 16: then Polarstern sonde number 16 of␣
↪your array is within temporal and spatial range of IASI overpasses
Then, ps_iasi_overlap_pixels[0] tells you which coordinates of IASI_DS overlaps␣
↪with Polarstern: For example,
ps_iasi_overlap_pixels[0] can be

array([[372, 108],
      [372, 109],
```

```
       [372, 110],
       [372, 111],
       [373, 108],
       [373, 111]])

--> first column: along_track coordinate that fulfills our requirements ;␣
 ↪second column: across_track coordinate that fulfills requirements
--> 6 pixels of IASI fulfill our requirements in this case. You may now select␣
 ↪the Polarstern temperature profile via:

PS_sonde_dict['temp'][which_PS_sonde_nonnan[0], :];

and the IASI temperature profile(s) (dimensions: along_track, across_track,␣
 ↪height):
"""
```

0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
16000
17000
18000
19000
20000
21000
22000
23000
24000
25000
26000
27000
28000
29000
30000

31000
32000
33000
34000
35000
36000
37000
38000
39000
40000
41000
42000
43000
44000
45000
46000
47000
48000
49000
50000
51000
52000
53000
54000
55000
56000
57000
58000
59000
60000
61000
62000
63000
64000
65000
66000
67000
68000
69000
70000
71000
72000
73000
74000
75000
76000
77000
78000

```
79000
80000
81000
82000
83000
84000
85000
86000
87000
88000
89000
90000
91000
92000
93000
94000
95000
96000
97000
98000
99000
```

[14]: "\nps_iasi_overlap_pixels must always be considered together with which_PS_sonde_nonnan:\nExample: which_PS_sonde_nonnan[0] is 16: then Polarstern sonde number 16 of your array is within temporal and spatial range of IASI overpasses\nThen, ps_iasi_overlap_pixels[0] tells you which coordinates of IASI_DS overlaps with Polarstern: For example,\nps_iasi_overlap_pixels[0] can be\n\narray([[372, 108],\n   [372, 109],\n   [372, 110],\n   [372, 111],\n   [373, 108],\n   [373, 111]])\n\n--> first column: along_track coordinate that fulfills our requirements ; second column: across_track coordinate that fulfills requirements\n--> 6 pixels of IASI fulfill our requirements in this case. You may now select the Polarstern temperature profile via:\n\nPS_sonde_dict['temp'][which_PS_sonde_nonnan[0], :];\n\nand the IASI temperature profile(s) (dimensions: along_track, across_track, height):\n"

[15]:
```
#n_height_iasi = len(IASI_DS.nlt)
#n_detected_pixels = len(ps_iasi_overlap_pixels[0])
#IASI_T = np.zeros((n_detected_pixels, n_height_iasi))                # here,␣
↪the T profiles for the current Polarstern sonde are saved to
#for idx, ps_ol in enumerate(ps_iasi_overlap_pixels[0]):
#       IASI_T[idx,:] = IASI_DS.atmospheric_temperature[ps_ol[0], ps_ol[1],:]

##IASI_T might include many nans because not all pixels of IASI actually have a␣
↪temperature profile!
##You might now either average over the number of detected pixels (ignore nans):
#IASI_T_avg = np.nanmean(IASI_T, axis=0)                # might produce␣
↪warnings (RuntimeWarning)
```

```
<ipython-input-15-a0b374def292>:9: RuntimeWarning: Mean of empty slice
  IASI_T_avg = np.nanmean(IASI_T, axis=0)                # might produce
warnings (RuntimeWarning)
```

[16]:

```
#IASI_T zu radiosonden(NyAl)-pixel-selektieren

n_time_NyAl = len(NyAl_sonde_dict['launch_time'])

distance_iasi_NyAl = np.full((n_along, n_across), 999999.99)
record_start_time = IASI_DS.record_start_time.values
which_NyAl_sonde = np.full((n_along, n_across), np.nan)
for i in range(n_along):
        if i%1000 == 0: print(i)
        # find NyAlesund radiosonde launch that has temporally closest to IASI␣
 ↪along track scan (pixel):
        launch_time_dif = np.abs(NyAl_sonde_dict['launch_time'] -␣
 ↪record_start_time[i])
        idx_NyAl_time = np.argmin(launch_time_dif)

        if np.abs(NyAl_sonde_dict['launch_time'][idx_NyAl_time] -␣
 ↪record_start_time[i]) <= 3600:
                # which_NyAl_sonde[i] = idx_NyAl_time
                for j in range(n_across):

                        if np.abs(IASI_lat[i,j]) <= 90:
                                disdis =␣
 ↪geodesic((NyAl_sonde_dict['lat'][idx_NyAl_time],␣
 ↪NyAl_sonde_dict['lon'][idx_NyAl_time]), (IASI_lat[i,j], IASI_lon[i,j])).km
                                if disdis < 50:
                                        which_NyAl_sonde[i,j] = idx_NyAl_time
                                        distance_iasi_NyAl[i,j] = disdis

IASI_DS['distance_iasi_NyAl'] = xarray.DataArray(distance_iasi_NyAl,␣
 ↪dims=['along_track', 'across_track'])

# remove nans from which_NyAl_sonde:
which_NyAl_sonde = which_NyAl_sonde.astype(np.int32)
which_NyAl_sonde_nonnan = np.unique(which_NyAl_sonde[which_NyAl_sonde >= 0]) #␣
 ↪explained below



# what do we have now:
# - distance_iasi_NyAl (n_along, n_across): distance of IASI to NyAl for each␣
 ↪IASI pixel (value is only non-nan if that pixel is temporally within 3600␣
 ↪sec of a
#          NyAl radiosonde launch AND if it is within 50 km of NyAl)
```

```
# - which_NyAl_sonde_nonnan (varying dimension): this now indicates which NyAl␣
↪sondes have IASI pixels that are within 3600 sec of a sonde launch and where␣
↪IASI
#         pixel is within 50 km of NyAl

# Later, we want to know if there is a IASI pixel for a given NyAl launch which␣
↪is close enough (i.e. < 50 km) and within
# 3600 sec of that sonde launch. So, we would like to have an array with the␣
↪shape (n_sondes_NyAl,2) (or (n_time_ps,2)) that tells
# us the exact along track and across track coordinate of IASI that fulfills␣
↪these conditions.
# So, we can now run through all NyAl sondes again, and check, if the indicated␣
↪along track coordinate has one or more IASI_DS['distance_iasi_NyAl'] < 50 km
iasi_pixels_for_NyAl = np.full((n_time_ps,2), 0)

NyAl_iasi_overlap_pixels = list()
for idx in range(n_time_NyAl):
    if idx in which_NyAl_sonde_nonnan: # then it's not a fill value and a IASI␣
↪pixel with time offset < 3600 sec exists for the current NyAl sonde launch:

            # find lines where which_NyAl_sonde is equal to idx:
        NyAl_iasi_overlap_pixels.append(np.argwhere(which_NyAl_sonde == idx))

"""
NyAl_iasi_overlap_pixels must always be considered together with␣
↪which_NyAl_sonde_nonnan:
Example: which_NyAl_sonde_nonnan[0] is 16: then NyAl sonde number 16 of your␣
↪array is within temporal and spatial range of IASI overpasses
Then, NyAl_iasi_overlap_pixels[0] tells you which coordinates of IASI_DS␣
↪overlaps with NyAl: For example,
NyAl_iasi_overlap_pixels[0] can be

array([[372, 108],
      [372, 109],
      [372, 110],
      [372, 111],
      [373, 108],
      [373, 111]])

--> first column: along_track coordinate that fulfills our requirements ;␣
↪second column: across_track coordinate that fulfills requirements
--> 6 pixels of IASI fulfill our requirements in this case. You may now select␣
↪the NyAl temperature profile via:

NyAl_sonde_dict['temp'][which_NyAl_sonde_nonnan[0], :];
```

```
and the IASI temperature profile(s) (dimensions: along_track, across_track,␣
 ↪height):
"""
```

0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
16000
17000
18000
19000
20000
21000
22000
23000
24000
25000
26000
27000
28000
29000
30000
31000
32000
33000
34000
35000
36000
37000
38000
39000
40000
41000
42000

43000
44000
45000
46000
47000
48000
49000
50000
51000
52000
53000
54000
55000
56000
57000
58000
59000
60000
61000
62000
63000
64000
65000
66000
67000
68000
69000
70000
71000
72000
73000
74000
75000
76000
77000
78000
79000
80000
81000
82000
83000
84000
85000
86000
87000
88000
89000
90000

```
91000
92000
93000
94000
95000
96000
97000
98000
99000
```

[16]:

```
"\nNyAl_iasi_overlap_pixels must always be considered together with
which_NyAl_sonde_nonnan:\nExample: which_NyAl_sonde_nonnan[0] is 16: then NyAl
sonde number 16 of your array is within temporal and spatial range of IASI
overpasses\nThen, NyAl_iasi_overlap_pixels[0] tells you which coordinates of
IASI_DS overlaps with NyAl: For example,\nNyAl_iasi_overlap_pixels[0] can
be\n\narray([[372, 108],\n       [372, 109],\n       [372, 110],\n       [372,
111],\n       [373, 108],\n       [373, 111]])\n\n--> first column: along_track
coordinate that fulfills our requirements ; second column: across_track
coordinate that fulfills requirements\n--> 6 pixels of IASI fulfill our
requirements in this case. You may now select the NyAl temperature profile
via:\n\nNyAl_sonde_dict['temp'][which_NyAl_sonde_nonnan[0], :];\n\nand the IASI
temperature profile(s) (dimensions: along_track, across_track, height):\n"
```

[17]:

```
#n_height_iasi = len(IASI_DS.nlt)
#n_detected_pixels = len(NyAl_iasi_overlap_pixels[0])
#IASI_T = np.zeros((n_detected_pixels, n_height_iasi))                # here,␣
↪the T profiles for the current NyAl sonde are saved to
#for idx, NyAl_ol in enumerate(NyAl_iasi_overlap_pixels[0]):
#       IASI_T[idx,:] = IASI_DS.atmospheric_temperature[NyAl_ol[0], NyAl_ol[1],:
↪]

##IASI_T might include many nans because not all pixels of IASI actually have a␣
↪temperature profile!
##You might now either average over the number of detected pixels (ignore nans):
#IASI_T_avg = np.nanmean(IASI_T, axis=0)                # might produce␣
↪warnings (RuntimeWarning)
```

```
<ipython-input-17-69479153df2f>:9: RuntimeWarning: Mean of empty slice
  IASI_T_avg = np.nanmean(IASI_T, axis=0)                # might produce
warnings (RuntimeWarning)
```

[18]:

```
IASI_ST = IASI_DS.surface_temperature.values
IASI_SP = IASI_DS.surface_pressure.values
```

[19]:

```
# height grid auf eine Größe gebracht werden / interpolieren darauf
NyAl_sonde_dict['q'] = convert_rh_to_spechum(NyAl_sonde_dict['temp'],␣
↪NyAl_sonde_dict['pres'], NyAl_sonde_dict['rh'])
```

```
IASI_T = IASI_DS.atmospheric_temperature.values
IASI_q = IASI_DS.atmospheric_water_vapor.values
```

[20]:
```
# for schleife für unterschiedliche shapes IASI_T.shape=(2980, 120, 101) along,␣
↪across, vertical IASI_DS.pressure_levels_temp.shape = (101,)

#IASI_alt = np.zeros(IASI_T.shape)
IASI_DS_pressure_levels_temp = IASI_DS.pressure_levels_temp.values
#for i in range(n_along):
#    for j in range(n_across):
#        IASI_alt[i,j,:] = -(R_d / g) * (IASI_T[i,j,:]*np.
↪log(IASI_DS_pressure_levels_temp) - IASI_ST[i,j]*np.log(IASI_SP[i,j]))
```

[45]:
```
n_sondes_PS = len(which_PS_sonde_nonnan)
n_height_PS = PS_sonde_dict['temp'].shape[1]

PS_T = np.zeros((n_sondes_PS,n_height_PS))
PS_P = np.zeros((n_sondes_PS,n_height_PS))
PS_q = np.zeros((n_sondes_PS,n_height_PS))

for i, which_PS in enumerate(which_PS_sonde_nonnan):
  #print(i,which_PS)
    PS_T[i,:] = PS_sonde_dict['temp'][which_PS,:]
    PS_P[i,:] = PS_sonde_dict['pres'][which_PS,:]
    PS_q[i,:] = PS_sonde_dict['q'][which_PS,:]

n_sondes_NyAl = len(which_NyAl_sonde_nonnan)
n_height_NyAl = NyAl_sonde_dict['temp'].shape[1]

NyAl_T = np.zeros((n_sondes_NyAl,n_height_NyAl))
NyAl_P = np.zeros((n_sondes_NyAl,n_height_NyAl))
NyAl_q = np.zeros((n_sondes_NyAl,n_height_NyAl))

for i, which_NyAl in enumerate(which_NyAl_sonde_nonnan):
  #  print(i,which_NyAl)
    NyAl_T[i,:] = NyAl_sonde_dict['temp'][which_NyAl,:]
    NyAl_P[i,:] = NyAl_sonde_dict['pres'][which_NyAl,:]*100
    NyAl_q[i,:] = NyAl_sonde_dict['q'][which_NyAl,:]*0.001
```

[46]:
```
#len(ps_iasi_overlap_pixels)#IASI überlappt mit PS

n_height_IASI = len(IASI_DS_pressure_levels_temp)
IASI_T_PS = np.zeros((n_sondes_PS,n_height_IASI))
IASI_q_PS = np.zeros((n_sondes_PS,n_height_IASI))
for idx, ps_iasi_overlap in enumerate(ps_iasi_overlap_pixels):
        along_chosen = ps_iasi_overlap[:,0]
```

```
        across_chosen = ps_iasi_overlap[:,1]
        IASI_T_PS[idx,:] = np.nanmean(IASI_T[along_chosen, across_chosen,:],␣
↪axis=0)
        IASI_q_PS[idx,:] = np.nanmean(IASI_q[along_chosen, across_chosen,:],␣
↪axis=0)
```

```
<ipython-input-46-93e1fcba0ccf>:9: RuntimeWarning: Mean of empty slice
  IASI_T_PS[idx,:] = np.nanmean(IASI_T[along_chosen, across_chosen,:], axis=0)
<ipython-input-46-93e1fcba0ccf>:10: RuntimeWarning: Mean of empty slice
  IASI_q_PS[idx,:] = np.nanmean(IASI_q[along_chosen, across_chosen,:], axis=0)
```

[47]:
```
#len(NyAl_iasi_overlap_pixels)#IASI überlappt mit NyAl

n_height_IASI = len(IASI_DS_pressure_levels_temp)
IASI_T_NyAl = np.zeros((n_sondes_NyAl,n_height_IASI))
IASI_q_NyAl = np.zeros((n_sondes_NyAl,n_height_IASI))
for idx, NyAl_iasi_overlap in enumerate(NyAl_iasi_overlap_pixels):
        along_chosen = NyAl_iasi_overlap[:,0]
        across_chosen = NyAl_iasi_overlap[:,1]
        IASI_T_NyAl[idx,:] = np.nanmean(IASI_T[along_chosen, across_chosen,:],␣
↪axis=0)
        IASI_q_NyAl[idx,:] = np.nanmean(IASI_q[along_chosen, across_chosen,:],␣
↪axis=0)
```

```
<ipython-input-47-da914ae8c662>:9: RuntimeWarning: Mean of empty slice
  IASI_T_NyAl[idx,:] = np.nanmean(IASI_T[along_chosen, across_chosen,:], axis=0)
<ipython-input-47-da914ae8c662>:10: RuntimeWarning: Mean of empty slice
  IASI_q_NyAl[idx,:] = np.nanmean(IASI_q[along_chosen, across_chosen,:], axis=0)
```

[48]:
```
#interpolation über druck koordinaten
IASI_DS_pressure_levels_humidity = IASI_DS.pressure_levels_humidity.values
IASI_T_PS_grid = np.zeros((n_sondes_PS,n_height_PS))
IASI_q_PS_grid = np.zeros((n_sondes_PS,n_height_PS))

for idx in range(n_sondes_PS):
    IASI_T_PS_grid[idx,:] = np.interp(PS_P[idx,:
↪],IASI_DS_pressure_levels_temp,IASI_T_PS[idx,:])
    IASI_q_PS_grid[idx,:] = np.interp(PS_P[idx,:
↪],IASI_DS_pressure_levels_humidity,IASI_q_PS[idx,:])
```

[49]:
```
#interpolation über druck koordinaten
IASI_DS_pressure_levels_humidity = IASI_DS.pressure_levels_humidity.values
IASI_T_NyAl_grid = np.zeros((n_sondes_NyAl,n_height_NyAl))
IASI_q_NyAl_grid = np.zeros((n_sondes_NyAl,n_height_NyAl))

for idx in range(n_sondes_NyAl):
```

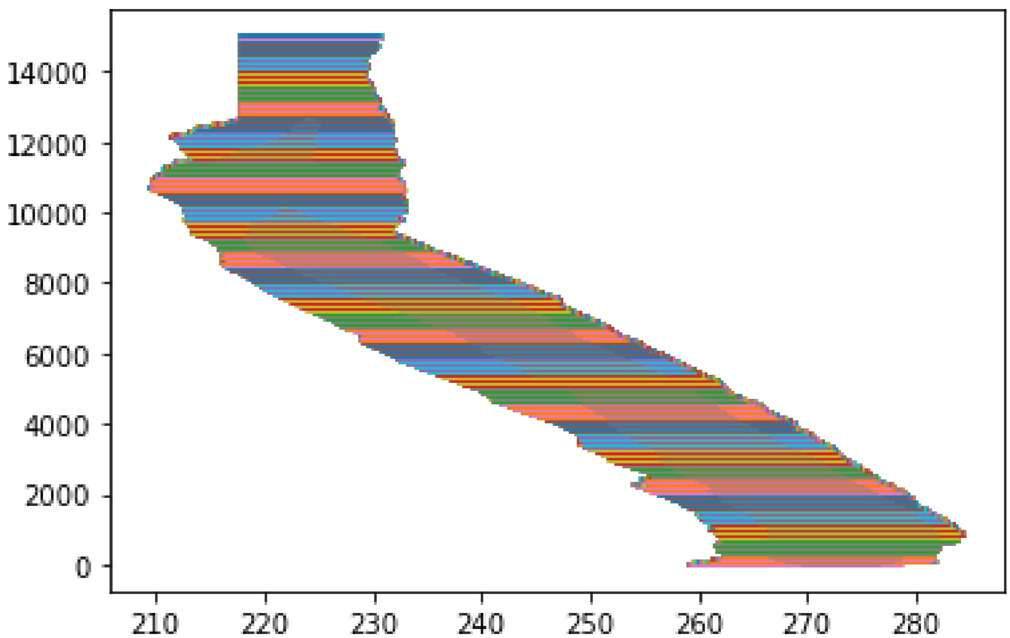
```
    IASI_T_NyAl_grid[idx,:] = np.interp(NyAl_P[idx,:
↪],IASI_DS_pressure_levels_temp,IASI_T_NyAl[idx,:])
    IASI_q_NyAl_grid[idx,:] = np.interp(NyAl_P[idx,:
↪],IASI_DS_pressure_levels_humidity,IASI_q_NyAl[idx,:])
```

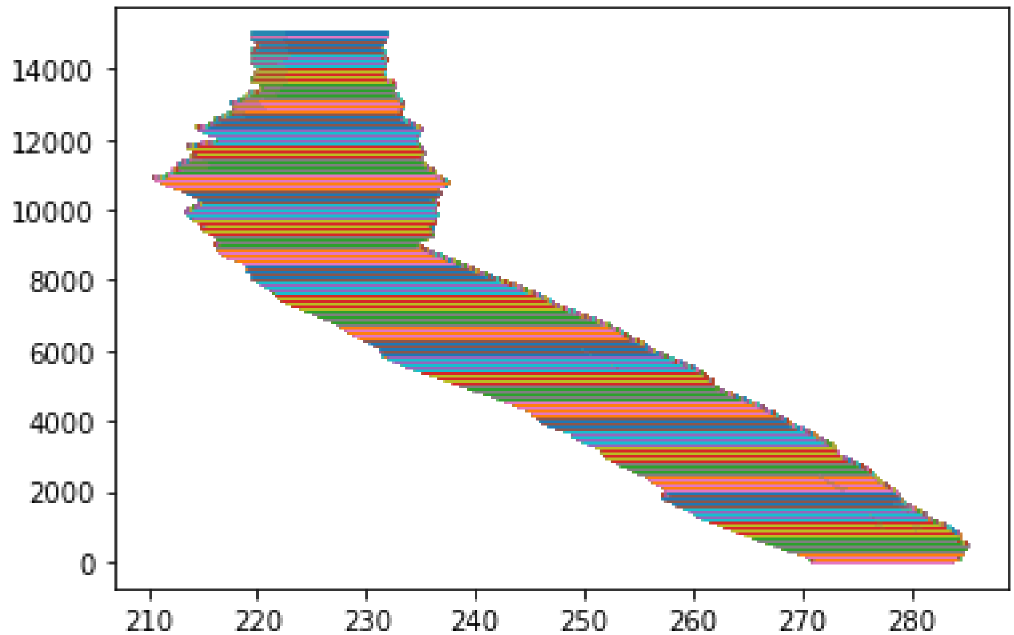
[50]:
```
#print(NyAl_P)
```

[51]:
```
#Kontrolle nur IASI geht nicht

plt.plot(PS_sonde_dict['temp'], PS_sonde_dict['height'])
plt.show()

plt.plot(NyAl_sonde_dict['temp'], NyAl_sonde_dict['height'])
plt.show()
```





0.0.5 Plot - Vergleich an dem gleichen Punkten

[52]:
```
#fig, ax = plt.subplots(constrained_layout=True, sharey=True)
#ax.plot(PS_sonde_dict['temp'][which_PS_sonde_nonnan[8],:],␣
↪PS_sonde_dict['pres'][which_PS_sonde_nonnan[8],:])
#ax.set_xlabel('T')
#ax.set_ylabel('pressure level [Pa]')
```

Vergleich von IASI am selben Ort [16,:] wie PS

[85]:
```
#print(IASI_T_PS[x,:],IASI_q_PS[x,:])
#print(PS_T[[16],:])
```

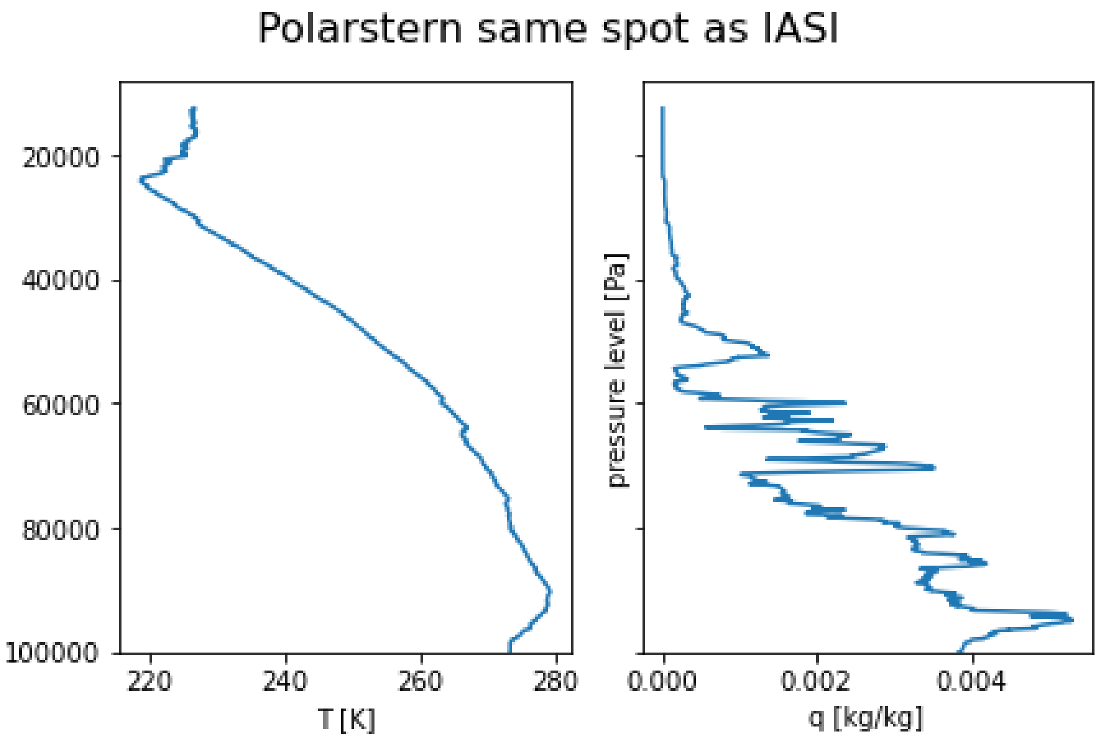
[93]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, constrained_layout=True, sharey=True)

ax1.plot(PS_T[16,:],PS_P[16,:])#ax1.set_xlabel('pressure level [Pa]')
ax1.set_xlabel('T [K]')
ax1.invert_yaxis()
ax1.set_ylim(100000)

ax2.plot(PS_q[16,:],PS_P[16,:])
ax2.set_xlabel('q [kg/kg]')
ax2.set_ylabel('pressure level [Pa]')

fig.suptitle('Polarstern same spot as IASI', fontsize=16)
```

[93]: Text(0.5, 0.98, 'Polarstern same spot as IASI')



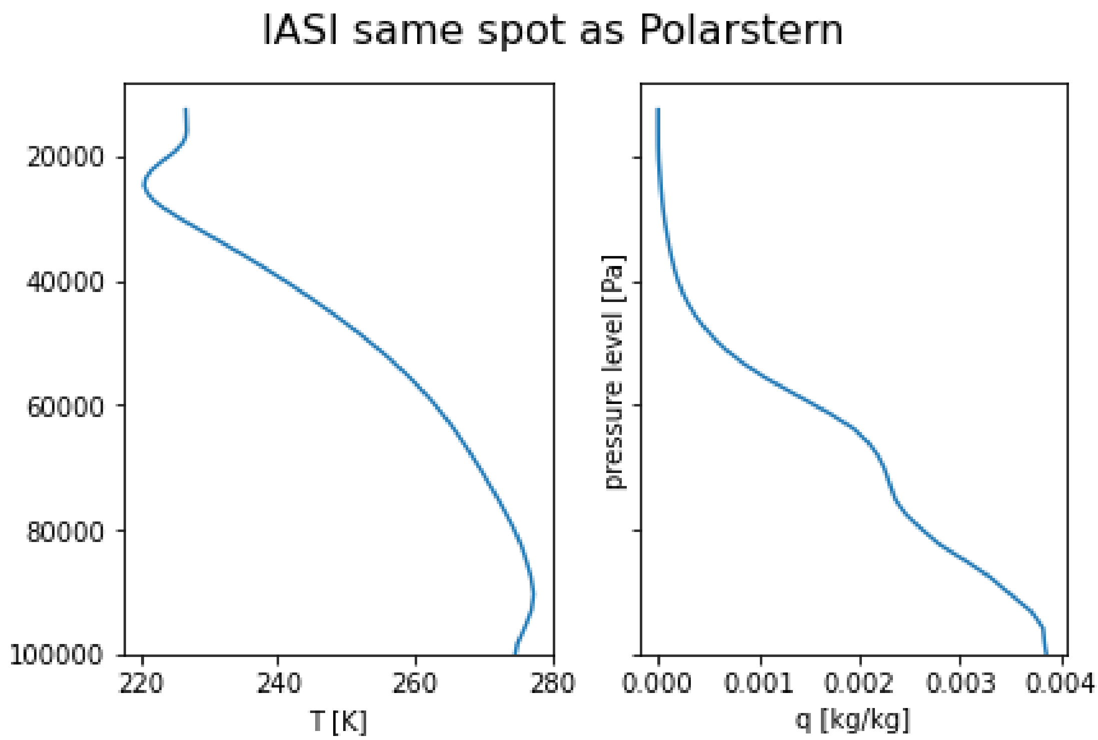
[77]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, constrained_layout=True, sharey=True)

ax1.plot(IASI_T_PS_grid[16,:], PS_P[16,:])#ax1.set_xlabel('pressure level [Pa]')
ax1.set_xlabel('T [K]')
ax1.invert_yaxis()
ax1.set_ylim(100000)

ax2.plot(IASI_q_PS_grid[16,:], PS_P[16,:])
ax2.set_xlabel('q [kg/kg]')
ax2.set_ylabel('pressure level [Pa]')

fig.suptitle('IASI same spot as Polarstern', fontsize=16)
```

[77]: Text(0.5, 0.98, 'IASI same spot as Polarstern')



Vergleich von IASI am selben Ort [12,:] wie NyAlesund

[98]:
```
#print(IASI_T_NyAl_grid[11,:],NyAl_T[11,:])
```

```
[          nan           nan           nan … 226.94075532 226.93746096
 226.93457832] [278.6499939 278.6499939 278.6499939 … 226.6499939 226.6499939
 226.6499939]
```

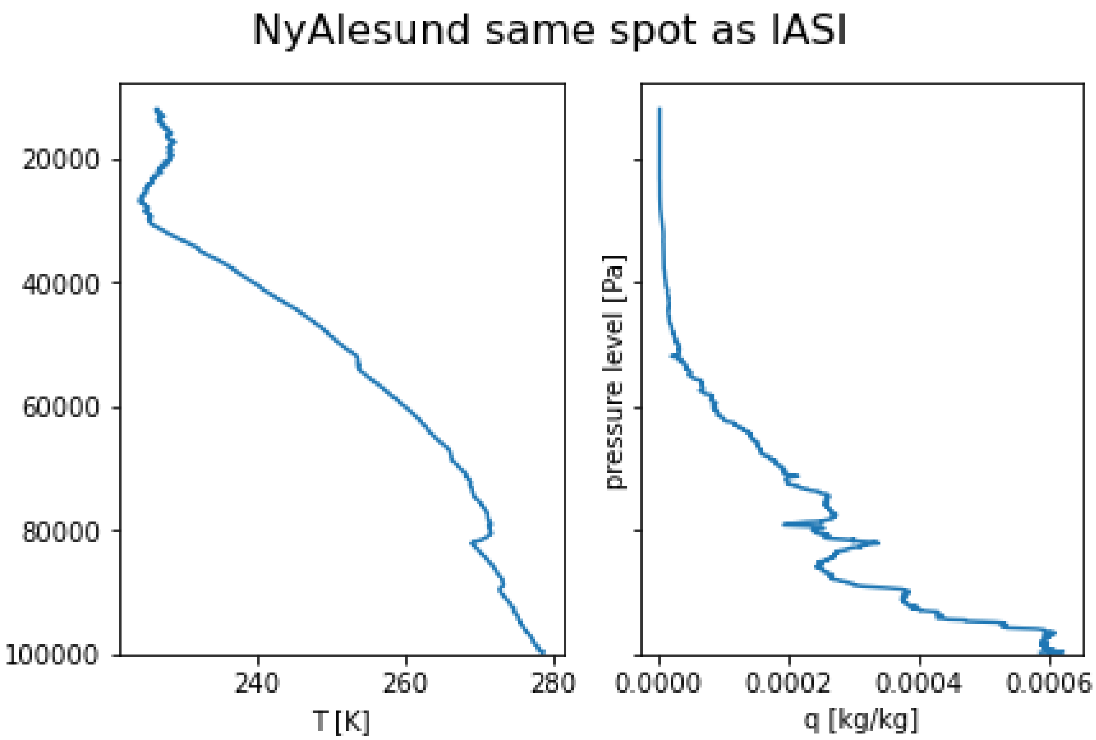
[99]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, constrained_layout=True, sharey=True)

ax1.plot(NyAl_T[11,:],NyAl_P[11,:])
ax1.set_xlabel('T [K]')
ax1.invert_yaxis()
ax1.set_ylim(100000)

ax2.plot(NyAl_q[11,:],NyAl_P[11,:])
ax2.set_xlabel('q [kg/kg]')
ax2.set_ylabel('pressure level [Pa]')

fig.suptitle('NyAlesund same spot as IASI', fontsize=16)
```

[99]:
```
Text(0.5, 0.98, 'NyAlesund same spot as IASI')
```



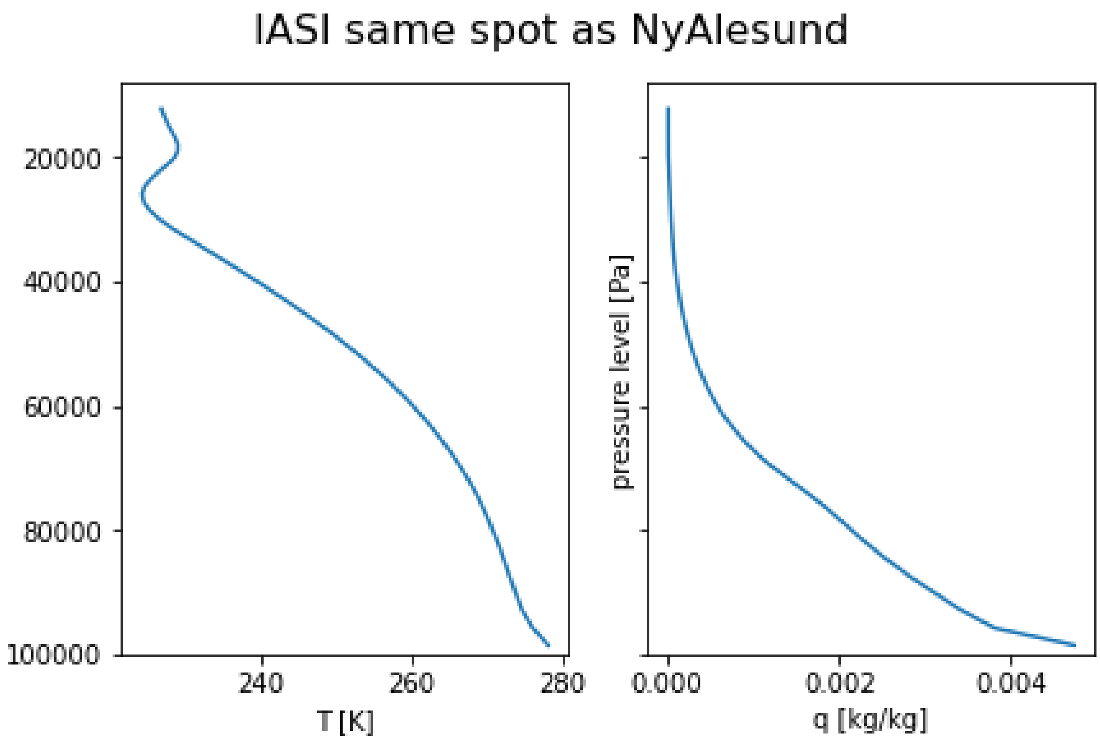
[100]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, constrained_layout=True, sharey=True)

ax1.plot(IASI_T_NyAl_grid[11,:], NyAl_P[11,:])
ax1.set_xlabel('T [K]')
ax1.invert_yaxis()
ax1.set_ylim(100000)

ax2.plot(IASI_q_NyAl_grid[11,:], NyAl_P[11,:])
ax2.set_xlabel('q [kg/kg]')
ax2.set_ylabel('pressure level [Pa]')

fig.suptitle('IASI same spot as NyAlesund', fontsize=16)
```

[100]: Text(0.5, 0.98, 'IASI same spot as NyAlesund')



0.0.6 RMSE

[35]:
```
# Polarstern und IASI

def RMSE(X_sonde,X_IASI):

    RMSE_X = np.sqrt( np.nanmean((X_IASI - X_sonde)**2, axis=0) )  # for RMSE
 ↪PROFILE, we average over time (which is axis 0)

    return RMSE_X

RMSE_T_IASI_PS = RMSE(IASI_T_PS_grid, PS_T)
RMSE_pres_PS = np.nanmean(PS_P, axis=0) # zugehöriger Druck
RMSE_q_IASI_PS = RMSE(IASI_q_PS_grid,PS_q)

RMSE_T_IASI_NyAl = RMSE(IASI_T_NyAl_grid, NyAl_T)
RMSE_pres_NyAl = np.nanmean(NyAl_P, axis=0) # zugehöriger Druck
RMSE_q_IASI_NyAl = RMSE(IASI_q_NyAl_grid,NyAl_q)
```

0.0.7 Biased

[36]:
```
# Bias von T IASI für eine bestimmte Höhe


def BIAS(X_sonde,X_IASI):

    BIAS_X = np.nanmean((X_IASI - X_sonde), axis=0)  # for BIAS PROFILE, we␣
 ↪average over time (which is axis 0)

    return BIAS_X

BIAS_T_IASI_PS = BIAS(IASI_T_PS_grid, PS_T)
BIAS_pres_PS = np.nanmean(PS_P, axis=0) # zugehöriger Druck
BIAS_q_IASI_PS = BIAS(IASI_q_PS_grid,PS_q)

BIAS_T_IASI_NyAl = BIAS(IASI_T_NyAl_grid, NyAl_T)
BIAS_pres_NyAl = np.nanmean(NyAl_P, axis=0) # zugehöriger Druck
BIAS_q_IASI_NyAl = BIAS(IASI_q_NyAl_grid,NyAl_q)
```

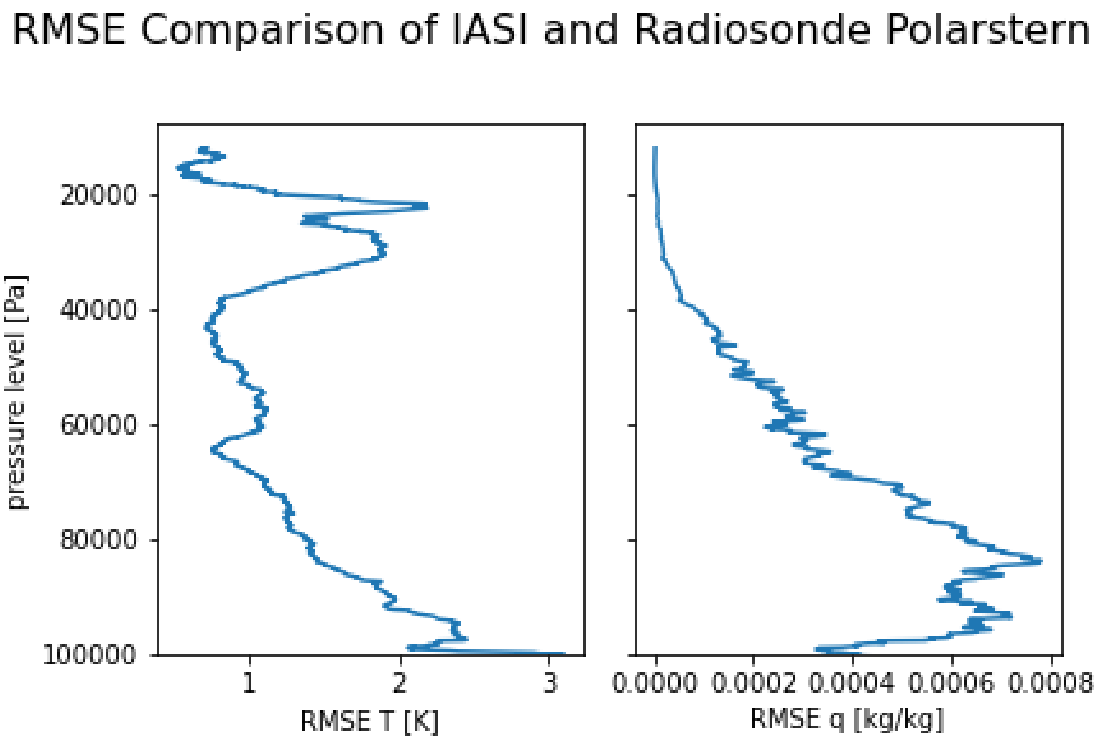
0.0.8 Plot - Vergleich der beiden Profile mit RMSE und BIAS

[81]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, constrained_layout=True, sharey=True)
ax1.plot(RMSE_T_IASI_PS, RMSE_pres_PS)
ax1.set_title('')
ax1.set_xlabel('RMSE T [K]')
ax1.set_ylabel('pressure level [Pa]')
ax1.invert_yaxis()
ax1.set_ylim(100000)

ax2.plot(RMSE_q_IASI_PS, RMSE_pres_PS)
ax2.set_xlabel('RMSE q [kg/kg]')
ax2.set_title(' ')

fig.suptitle('RMSE Comparison of IASI and Radiosonde Polarstern', fontsize=16)
```

[81]:
```
Text(0.5, 0.98, 'RMSE Comparison of IASI and Radiosonde Polarstern')
```

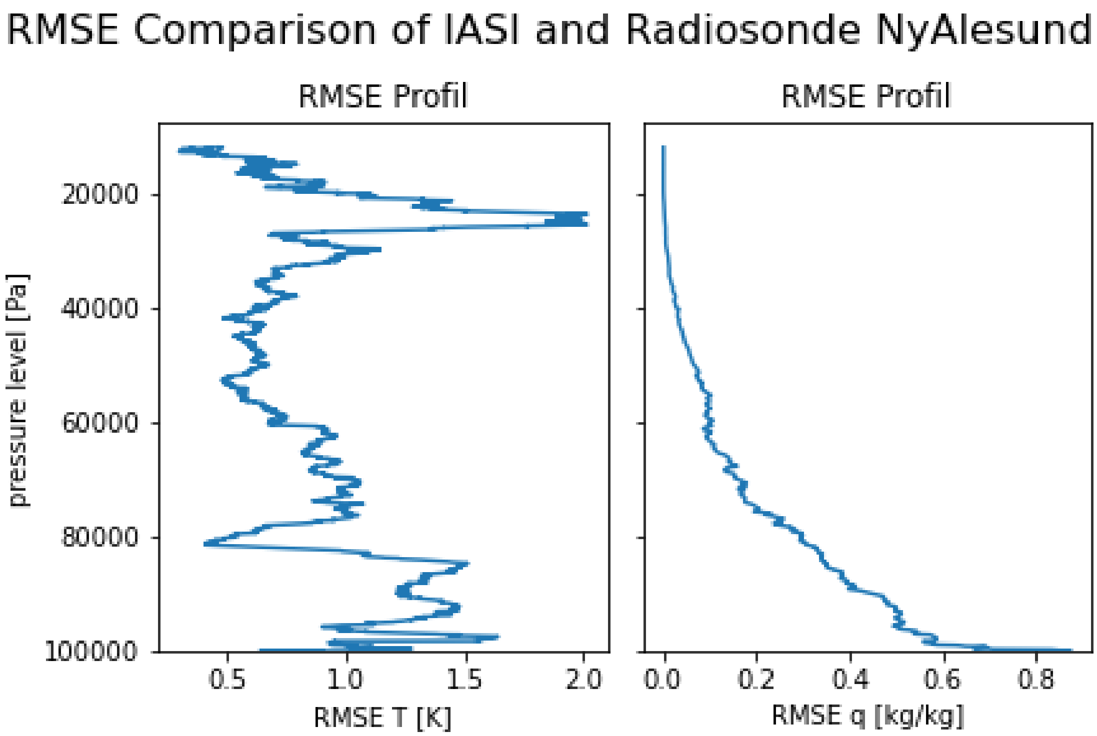


[82]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, constrained_layout=True, sharey=True)
ax1.plot(RMSE_T_IASI_NyAl, RMSE_pres_NyAl)
ax1.set_title('RMSE Profil')
ax1.set_xlabel('RMSE T [K]')
ax1.set_ylabel('pressure level [Pa]')
ax1.invert_yaxis()
ax1.set_ylim(100000)

ax2.plot(RMSE_q_IASI_NyAl, RMSE_pres_NyAl)
ax2.set_xlabel('RMSE q [kg/kg]')
ax2.set_title('RMSE Profil')

fig.suptitle('RMSE Comparison of IASI and Radiosonde NyAlesund', fontsize=16)
```

[82]:
```
Text(0.5, 0.98, 'RMSE Comparison of IASI and Radiosonde NyAlesund')
```



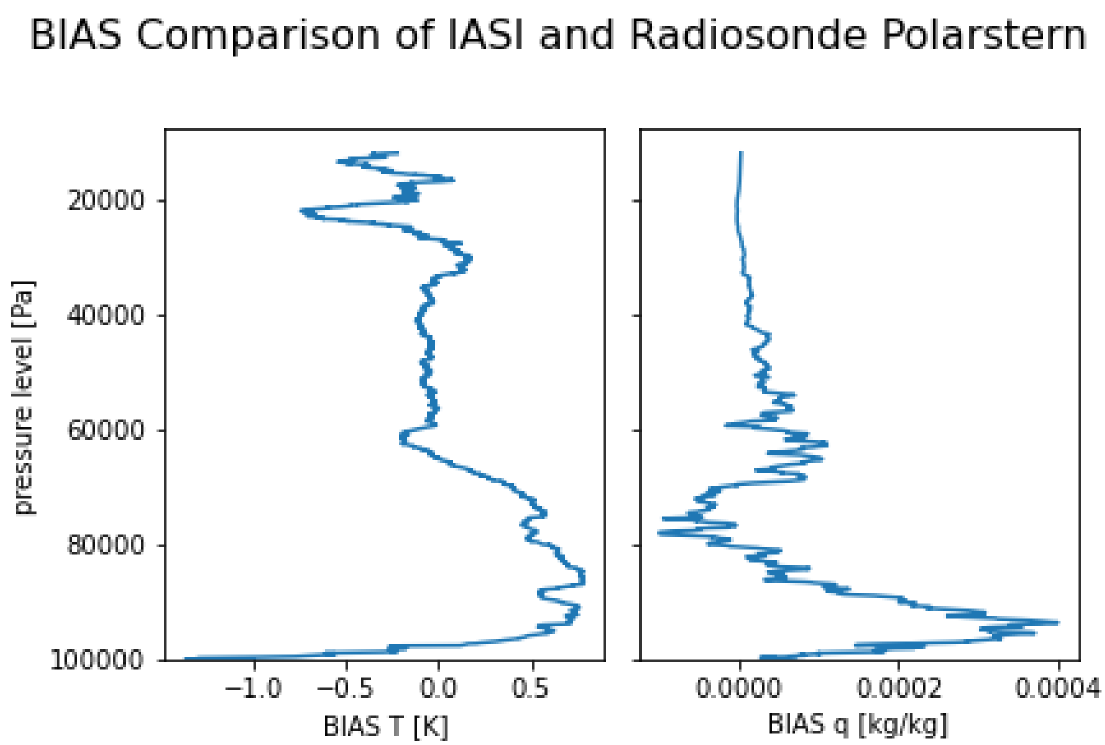
[39]:
```
#IASI_DS.pressure_levels_temp.values.shape
#print(IASI_T_PS_grid.shape)
```

[83]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, constrained_layout=True, sharey=True)
ax1.plot(BIAS_T_IASI_PS, BIAS_pres_PS)
ax1.set_title('')
ax1.set_xlabel('BIAS T [K]')
ax1.set_ylabel('pressure level [Pa]')
ax1.invert_yaxis()
ax1.set_ylim(100000)

ax2.plot(BIAS_q_IASI_PS, BIAS_pres_PS)
ax2.set_xlabel('BIAS q [kg/kg]')
ax2.set_title(' ')

fig.suptitle('BIAS Comparison of IASI and Radiosonde Polarstern', fontsize=16)
```

[83]:
```
Text(0.5, 0.98, 'BIAS Comparison of IASI and Radiosonde Polarstern')
```

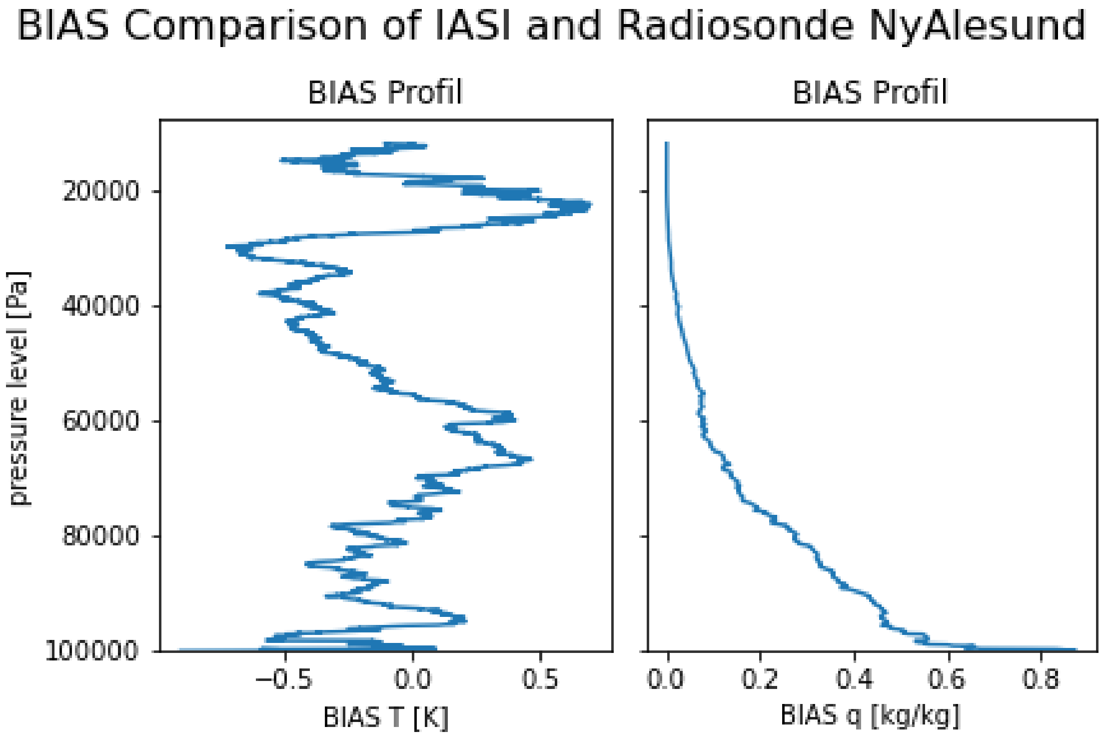



```
[84]: fig, (ax1, ax2) = plt.subplots(1, 2, constrained_layout=True, sharey=True)
ax1.plot(BIAS_T_IASI_NyAl, BIAS_pres_NyAl)
ax1.set_title('BIAS Profil')
ax1.set_xlabel('BIAS T [K]')
ax1.set_ylabel('pressure level [Pa]')
ax1.invert_yaxis()
ax1.set_ylim(100000)

ax2.plot(BIAS_q_IASI_NyAl, BIAS_pres_NyAl)
ax2.set_xlabel('BIAS q [kg/kg]')
ax2.set_title('BIAS Profil')

fig.suptitle('BIAS Comparison of IASI and Radiosonde NyAlesund', fontsize=16)
```

```
[84]: Text(0.5, 0.98, 'BIAS Comparison of IASI and Radiosonde NyAlesund')
```


BIAS Comparison of IASI and Radiosonde NyAlesund
BIAS Profil
BIAS Profil
pressure level [Pa]
20000
40000
60000
80000
100000
−0.5
0.0
0.5
BIAS T [K]
0.0
0.2
0.4
0.6
0.8
BIAS q [kg/kg]

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```