CTD_dfo-walle652-20200616

November 10, 2025

1 CTD corrections applied to delayed-mode data

This notebook performs analysis and correction of GPCTD data from the following C-PROOF glider deployment:

[1]:
```
glider_name = 'dfo-walle652'
deploy_name = f'{glider_name}-20200616'
deploy_prefix = f'./glider/{glider_name}/{deploy_name}/'
filepath = f'deployments/{glider_name}/{deploy_name}/' # having this is␣
 ↪important later for functions that auto-load data
openfile = f'{filepath}/{deploy_name}_delayed.nc'
opengridfile = f'{filepath}/{deploy_name}_grid_delayed.nc'
deployfile = f'{filepath}/{deploy_name}.yml'

description = 'Line P'
initials = 'LT'

# CTD specs:
sensor = 'CTD_9409'

# For conductivity filter:
accuracy = 0.0003 #accuracy of the sensor is 0.0003 S/m, used as a cutoff on␣
 ↪the exclusion criterion

from datetime import date
processing_date = date.today().strftime('%Y%m%d')
processing_protocol = 'C-PROOF_SBE_CTDProcessingReport_v0.2.pdf'
processing_report = 'TBD' # 'CTD_dfo-bb046-20220707_v3'


# Import module for loading .md files
from IPython.display import Markdown, display
import os

os.chdir(f'/Users/Lauryn/Documents/processing/')
```

[2]:
```
# Summarize info for report:
print(f'** {description}:  glider {glider_name}**')
print(f'************')
print(f'* Deployment: {deploy_name}')
print(f'* Sensor: {sensor}')
print(f'')

# print(f'* Protocols are detailed in: {processing_protocol}')
print(f'* Processing steps will be saved in: CTD_{deploy_name}.html')
# print(f'* Files will be located in: {deploy_prefix}')
print(f'* Processed by {initials}, Ocean Sciences Division, Fisheries and␣
 ↪Oceans Canada')
print(f'* Processing date: {processing_date}')
```

```
** Line P:  glider dfo-walle652**
************
* Deployment: dfo-walle652-20200616
* Sensor: CTD_9409

* Processing steps will be saved in: CTD_dfo-walle652-20200616.html
* Processed by LT, Ocean Sciences Division, Fisheries and Oceans Canada
* Processing date: 20251107
```

[3]:
```
display(Markdown("./docs/CTD_1_Preamble.md"))
```

2 1.0 Preamble

This document describes conductivity, temperature, and pressure data processing steps applied to delayed mode data collected using Sea-Bird Scientific Glider Payload Conductivity Temperature Depth (GPCTD) sensors mounted on C-PROOF Slocum and SeaExplorer autonomous ocean gliders. This sensor has a nominal sampling rate of 1 Hz and was designed specifically for Slocum gliders. This document covers the application of the sensor alignment correction and the thermal lag correction, as well as removal of questionable conductivity values and salinity profiles.

2.1 1.1 Set up the processing

The processing steps below are applied to delayed mode data stored in a single netcdf timeseries file created using the Pyglider data processing package (https://github.com/c-proof/pyglider).

The metadata and sensor calibration sheets are available via the deployment page on the C-PROOF website at: **https://cproof.uvic.ca/gliderdata/deployments/dfo-bb046/dfo-bb046-20220707/**

[4]:
```
import warnings
warnings.filterwarnings('ignore')

import xarray as xr
import numpy as np
```

```
import pathlib
import pyglidersensor as pgs
from pyglider.ncprocess import make_gridfiles

from datetime import datetime, date
%matplotlib ipympl


import scipy.stats as stats

import seawater
import gsw

%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from matplotlib.dates import DateFormatter
import cmocean
import cartopy.crs as ccrs
import cartopy.feature as cfeature

import pandas as pd

%load_ext autoreload
%autoreload 2

from scipy import signal
import seawater as sw
```

[5]:
```
%matplotlib ipympl
```

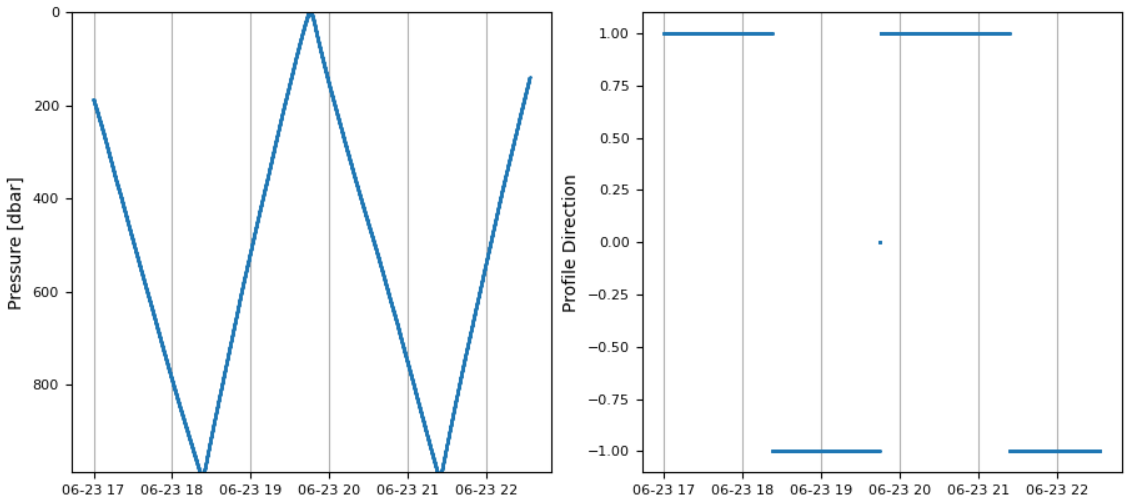
2.2 1.2 Profile Check

Check that upcasts and downcasts are being properly identified. **Negative values should be associated with upcasts**.

[6]:
```
print(f'Loading: {openfile}')

caption = ('Identifying upcasts and downcasts. The left panel shows '
           'pressure vs. time and the right panel shows profile direction vs. '
           'time for a small subset of the time series:')


fname = openfile

with xr.open_dataset(fname) as ds0:
```

```
    # SAVE SOME PARAMS FOR PLOTTING DOWN BELOW!
    N = len(ds0.time)
    MAX_DEPTH = np.nanmax(ds0.depth)
    NUM_PROFILES = np.nanmax(ds0.profile_index)

    print('************')
    print(f'* There are {N} data points in total, with {NUM_PROFILES} profiles')
    print(f'* Time period: {pd.to_datetime(np.nanmin(ds0.time)).
↪strftime("%Y-%m-%d")} to {pd.to_datetime(np.nanmax(ds0.time)).
↪strftime("%Y-%m-%d")}')
    print(f'* Depth range: {round(np.nanmin(ds0.depth))} - {round(MAX_DEPTH)}␣
↪metres')
    print('************')
    if N > 50000:
        todo = slice(int(N/2)-5000, int(N/2)+5000)
    else:
        todo = slice(int(N/3), int(2*N/3))

    fig, axs = plt.subplots(nrows=1, ncols=2,
                            constrained_layout=True,
                            figsize=(9, 4))

    ds = ds0.isel(time=todo)
    axs[0].plot(ds.time, ds.pressure, '.', markersize=1)
    axs[0].set_ylim([MAX_DEPTH, 0])
    axs[0].set_ylabel('Pressure [dbar]')
    axs[0].tick_params(axis='both', labelsize=8)
    axs[0].grid(axis='x')

    axs[1].plot(ds.time, ds.profile_direction, '.', markersize=1)
    axs[1].set_ylabel('Profile Direction')
    axs[1].tick_params(axis='both', labelsize=8)
    axs[1].grid(axis='x')
    print(caption)
```

```
Loading: deployments/dfo-walle652/dfo-walle652-20200616//dfo-
walle652-20200616_delayed.nc
************
* There are 567914 data points in total, with 330.0 profiles
* Time period: 2020-06-16 to 2020-07-01
* Depth range: 0 - 987 metres
************
Identifying upcasts and downcasts. The left panel shows pressure vs. time and
the right panel shows profile direction vs. time for a small subset of the time
series:
```

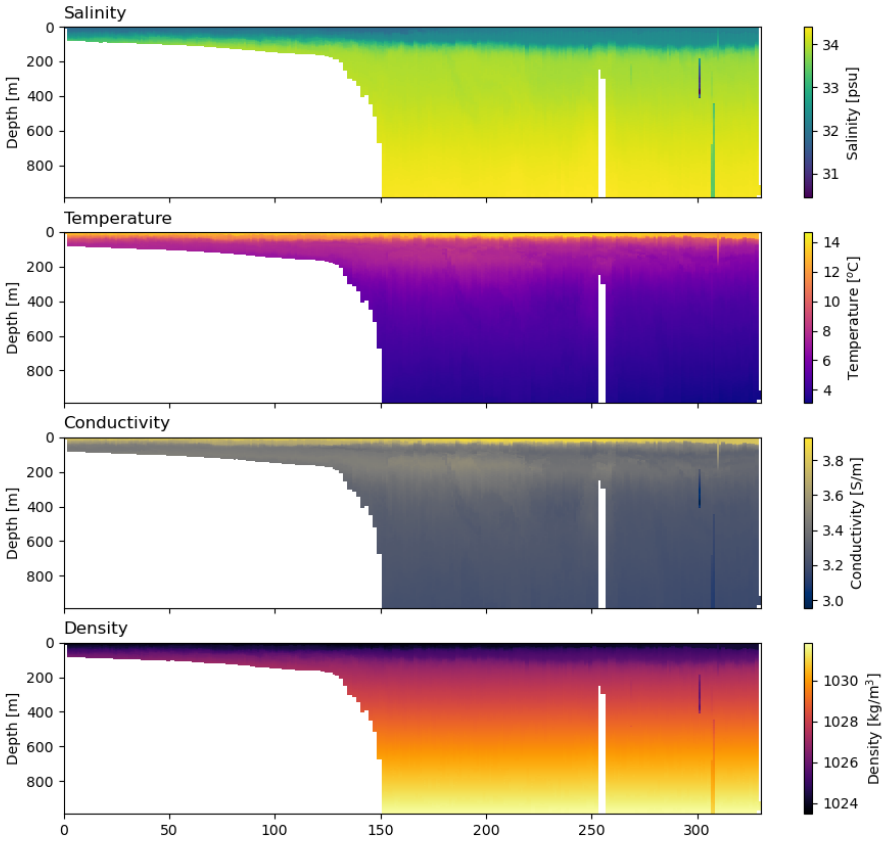


2.3 1.3 Delayed-mode data prior to corrections

Checking fields (temperature, salinity, conductivity and density) in the delayed-mode data, before any CTD corrections:

[7]:
```
tds = opengridfile
ds = xr.open_dataset(tds)
list(ds.keys())

fig, axs = plt.subplots(4, 1, figsize=(11, 10), sharey=True, sharex=True)

xlims = [0, NUM_PROFILES]
ylims=[MAX_DEPTH,0]
# ylims=[50,0]

pc = axs[0 ].pcolormesh(ds.profile, ds.depth, ds['salinity'],rasterized=True)
axs[0].set_ylim(ylims)
axs[1].set_xlim(xlims)
fig.colorbar(pc, ax=axs[0], label = 'Salinity [psu]')
axs[0].set_title('Salinity',loc='left')

pc = axs[1].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['temperature'],rasterized=True,cmap='plasma')

fig.colorbar(pc, ax=axs[1], label = 'Temperature [$^o$C]')
axs[1].set_title('Temperature',loc='left')
pc = axs[2].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['conductivity'],rasterized=True,cmap='cividis')
fig.colorbar(pc, ax=axs[2], label = 'Conductivity [S/m]')
axs[2].set_title('Conductivity',loc='left')
```

```
# pc = axs[3].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['oxygen_concentration'],rasterized=True,cmap='inferno')
# fig.colorbar(pc, ax=axs[3])
# axs[3].set_title('Oxygen Concentration',loc='left')

pc = axs[3].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['density'],rasterized=True,cmap='inferno')
fig.colorbar(pc, ax=axs[3], label = 'Density [kg/m$^3$]')
axs[3].set_title('Density',loc='left')

axs[0].set_ylabel('Depth [m]');
axs[1].set_ylabel('Depth [m]');
axs[2].set_ylabel('Depth [m]');
axs[3].set_ylabel('Depth [m]');
```



[8]:
```
display(Markdown("./docs/CTD_2_Steps.md"))
```

3 2.0 Corrections applied to delayed mode data for this mission

Processing steps:

1. Identification of anomalous conductivity values
2. Identification of questionable salinity profiles
3. Sensor alignment correction
4. Thermal lag correction

3.1 2.1.1 Identification and removal of anomalous conductivity values

We identify and remove any conductivity values that are obviously unphysical, which is typically caused by air bubbles in the conductivity cell. We use a simple criterion applied to the raw conductivity data. The criterion temporarily flags any data points that are more than **5 standard deviations** away from the overall time series mean for a given depth bin and profile bin, then recomputes the mean and standard deviation, excluding the temporarily flagged values. Conductivity values that still differ from the mean by more than **3 standard deviations** are flagged as 'bad' and set to NaN in the time series. If the difference between the 'bad' values and the mean is less than the accuracy of the sensor, which is 0.0003 S/m for the GPCTD, then those points are not excluded.

This criterion is applied to data binned first by profile index, in increments of **50 profiles**, then binned by depth, in increments of **5 m**. The use of profile index bins rather than time or temperature bins is designed to allow for the removal of unphysical values.

Adjustments to this correction are based on examining the data and making a judgment call about which conductivity values are undeniably 'bad'. In this case, we want to exclude the **extremely low values occurring at the surface** consistent with air bubbles in the cell. Some unphysical values are missed by this correction, and may be caught during the removal of unphysical salinity profiles in further stepsbelow.

Note that for this mission:

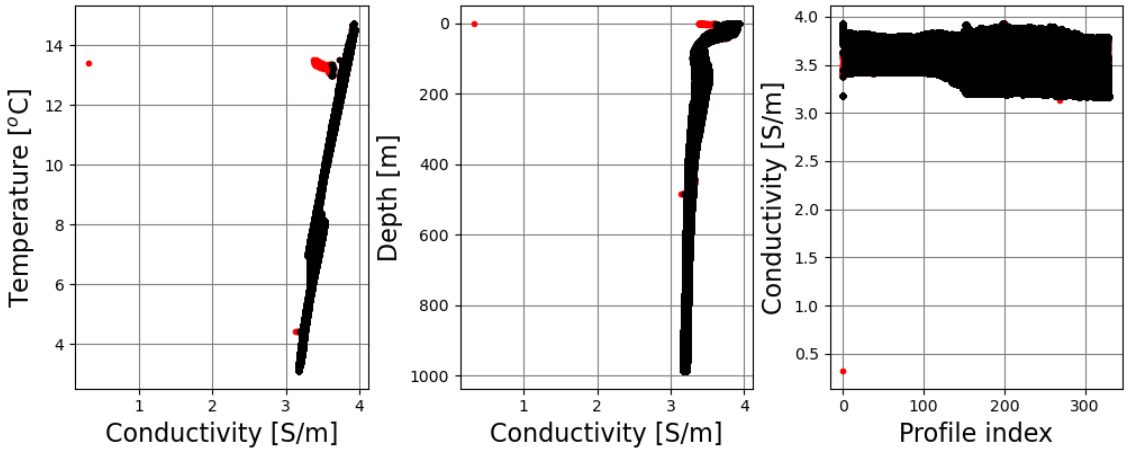
[9]:
```
###open the spike free dataset
ds0 = xr.open_dataset(f'{filepath}/{deploy_name}_spikeClean.nc')
ds = xr.open_dataset(f'{filepath}/{deploy_name}_grid_spikeClean.nc')

srate = stats.mode((np.diff(ds0.time)).astype('timedelta64[s]')).mode
fs = 1/srate.astype(float) #the sampling frequency = 1/(delta t)
print('************')
print(f'The mode of the sampling rate for the GPCTD is one sample every {srate}.
  ↪')
print('************')
```

```
************
The mode of the sampling rate for the GPCTD is one sample every 2 seconds.
************
```

[10]:
```
# Identify the questionable conductivity values
flag_stdev = 5 #number of standard deviations to temporarily flag bad salinity␣
 ↪values
clean_stdev = 3 #number of standard deviations to flag bad conductivity values,␣
 ↪after removing the temporary bad values from the calc
dT = 50 #size of the profile bins
dz = 5 #size of the depth bins

ts0 = ds0.copy() #make a copy of the spike-free timeseries
ts0.conductivity[ts0.conductivity<0.1] = np.nan
ts = pgs.get_conductivity_clean(ts0, dT, dz, flag_stdev, clean_stdev, accuracy)

# Figures to look at the comparison
fig, ax = plt.subplots(1,3,figsize=(10,4), constrained_layout=True)

ax[0].plot(ts.conductivity, ts.temperature, color='r', marker='.',␣
 ↪linestyle='none', label='Removed')
ax[0].plot(ts.conductivityClean, ts.temperature, color='k', marker='.',␣
 ↪linestyle='none', label='Retained')
ax[0].set_ylabel('Temperature [$^o$C]', fontsize=16)
ax[0].set_xlabel('Conductivity [S/m]', fontsize=16)
ax[0].grid(axis='both', color='0.5')

ax[1].plot(ts.conductivity, ts.depth, color='r', marker='.', linestyle='none')
ax[1].plot(ts.conductivityClean, ts.depth, color='k', marker='.',␣
 ↪linestyle='none')
ax[1].set_xlabel('Conductivity [S/m]', fontsize=16)
ax[1].set_ylabel('Depth [m]', fontsize=16)
ax[1].invert_yaxis()
ax[1].grid(axis='both', color='0.5')

ax[2].plot(ts.profile_index, ts.conductivity, color='r', marker='.',␣
 ↪linestyle='none')
ax[2].plot(ts.profile_index, ts.conductivityClean, color='k', marker='.',␣
 ↪linestyle='none')
ax[2].set_xlabel('Profile index', fontsize=16)
ax[2].set_ylabel('Conductivity [S/m]', fontsize=16)
ax[2].grid(axis='both', color='0.5')

print('Fig 2: Temperature vs. conductivity (left), depth vs. conductivity␣
 ↪(middle), '
      'and conductivity vs. profile index (right), '
      'with the red dots showing the unphysical values flagged as bad and␣
 ↪removed:')
```

Fig 2: Temperature vs. conductivity (left), depth vs. conductivity (middle), and conductivity vs. profile index (right), with the red dots showing the unphysical

values flagged as bad and removed:



Adjustments to this correction are based on examining the data and making a judgment call about which conductivity values are undeniably 'bad'. In this case, we want to exclude the **extremely low values occurring at the surface** (Fig. 2) consistent with air bubbles in the cell. Some unphysical values are missed by this correction, and may be caught during the removal of unphysical salinity profiles below.

3.2 2.1.2 Remove spikes from biofouling

Manually remove spikes in the data. Sometimes **biofouling** occurs causing unphysical values.

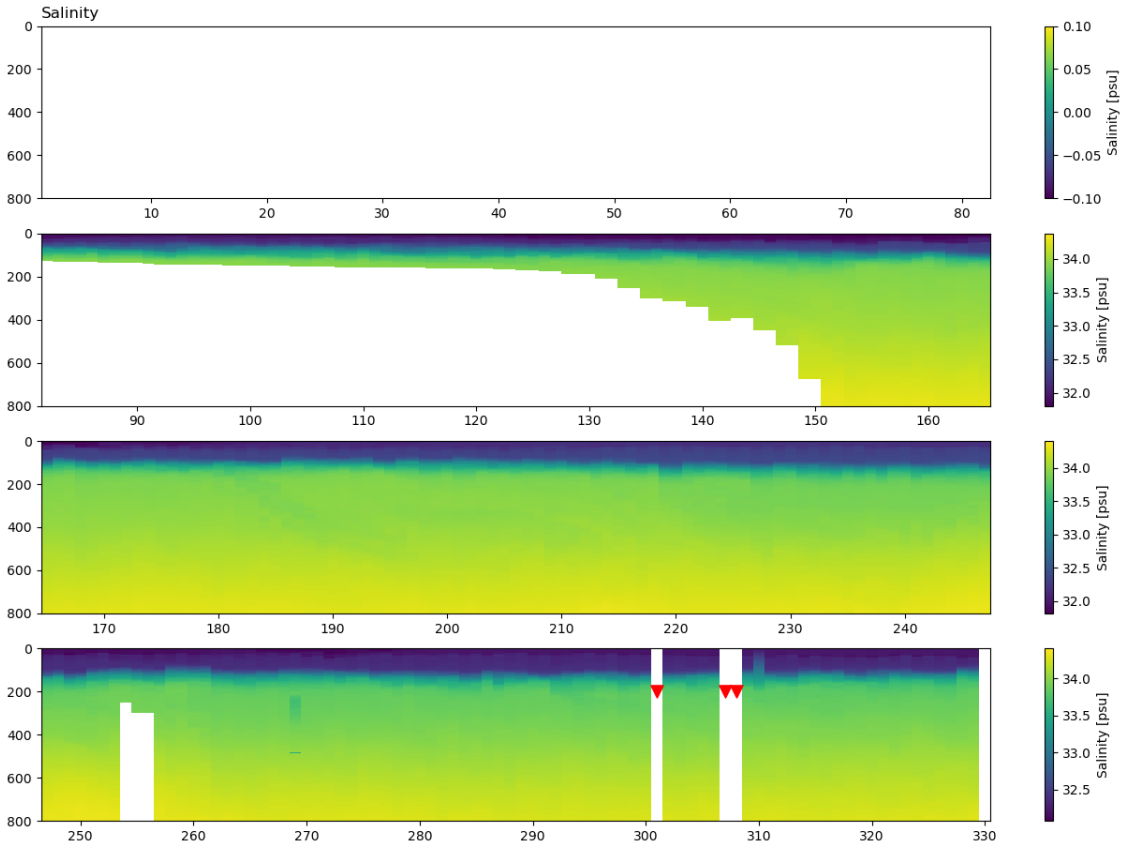
[11]:
```
# Now adding zoomed plot
xlim_1 = [0, int(NUM_PROFILES/4)]
xlim_2 = [int(NUM_PROFILES/4), int(NUM_PROFILES/4*2)]
xlim_3 = [int(NUM_PROFILES/4*2), int(NUM_PROFILES/4*3)]
xlim_4 = [int(NUM_PROFILES/4*3), NUM_PROFILES]

Y_LIMS = [800, 0] #######modified

fig, axs = plt.subplots(4, 1, #height_ratios=[1, 4],
                        figsize = [12,9],
                        layout='constrained', sharex=False)



profile_lims = xlim_1
ds_sub = ds.where((ds.profile >=profile_lims[0]) & (ds.profile <=␣
 ↪profile_lims[1]), drop=True)
ds_sub = ds_sub.isel(depth=range(200,800))

ax = axs[0]
```

```
pc = ax.pcolormesh(ds_sub.profile, ds_sub.depth,␣
 ↪ds_sub['salinity'],rasterized=True)
axs[0].set_ylim(Y_LIMS)

fig.colorbar(pc, ax=ax, label = 'Salinity [psu]')
axs[0].set_title('Salinity',loc='left')


########
profile_lims = xlim_2
ds_sub = ds.where((ds.profile >=profile_lims[0]) & (ds.profile <=␣
 ↪profile_lims[1]), drop=True)

ax = axs[1]
pc = ax.pcolormesh(ds_sub.profile, ds_sub.depth,␣
 ↪ds_sub['salinity'],rasterized=True)
ax.set_ylim(Y_LIMS)

fig.colorbar(pc, ax=ax, label = 'Salinity [psu]')
axs[0].set_title('Salinity',loc='left')

######
profile_lims = xlim_3
ds_sub = ds.where((ds.profile >=profile_lims[0]) & (ds.profile <=␣
 ↪profile_lims[1]), drop=True)

ax = axs[2]
pc = ax.pcolormesh(ds_sub.profile, ds_sub.depth,␣
 ↪ds_sub['salinity'],rasterized=True)
ax.set_ylim(Y_LIMS)

fig.colorbar(pc, ax=ax, label = 'Salinity [psu]')
axs[0].set_title('Salinity',loc='left')

#########
profile_lims = xlim_4
ds_sub = ds.where((ds.profile >=profile_lims[0]) & (ds.profile <=␣
 ↪profile_lims[1]), drop=True)

ax = axs[3]
pc = ax.pcolormesh(ds_sub.profile, ds_sub.depth,␣
 ↪ds_sub['salinity'],rasterized=True)
ax.set_ylim(Y_LIMS)

fig.colorbar(pc, ax=ax, label = 'Salinity [psu]')
axs[0].set_title('Salinity',loc='left')
```

```
print('Zooming in along the glider deployment to visualize salinity spikes.␣
 ↪Spikes manually identified are labeled with a red arrow.')

############
#Manually identify
axs[3].scatter(301,200,90,marker='v',color='r',zorder=2)
axs[3].scatter(307,200,90,marker='v',color='r',zorder=2)
axs[3].scatter(308,200,90,marker='v',color='r',zorder=2)
```

```
Zooming in along the glider deployment to visualize salinity spikes. Spikes
manually identified are labeled with a red arrow.
```

[11]: <matplotlib.collections.PathCollection at 0x33b5a5090>



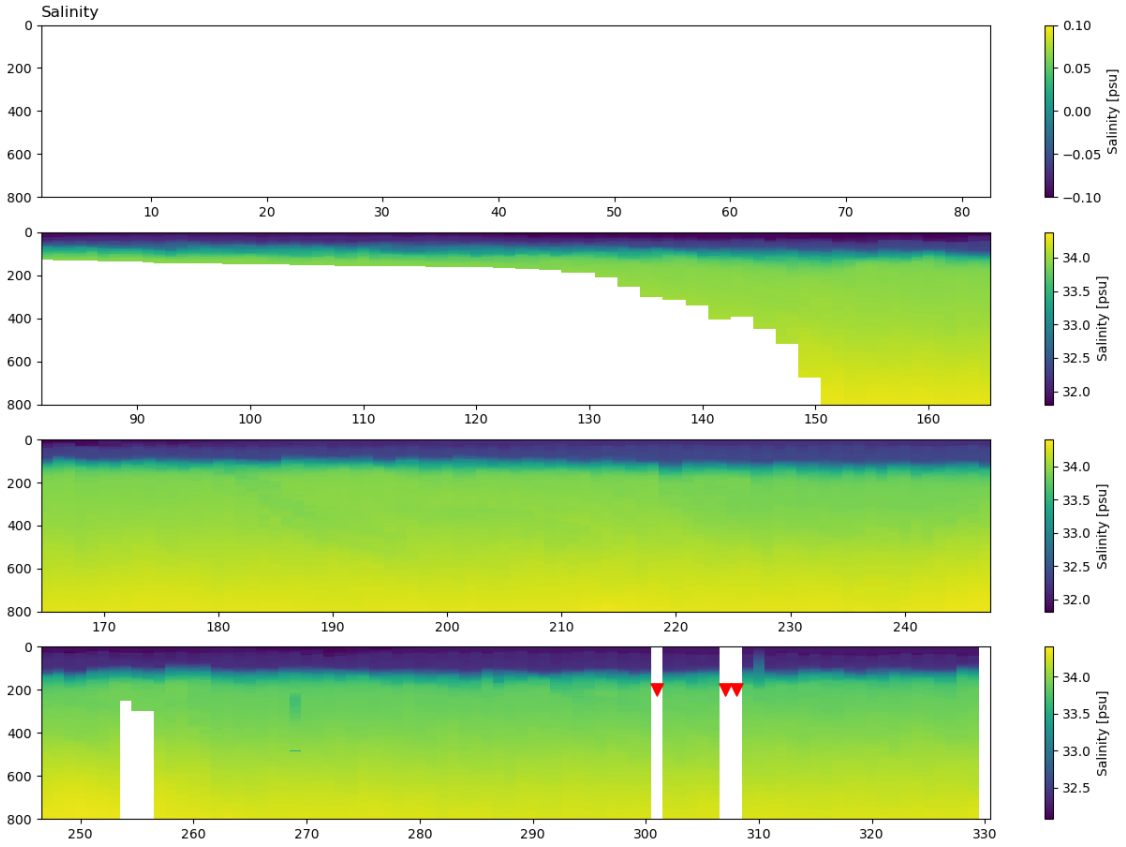
[12]:
```
#### mask these indices
spiky_profiles = [301,307,308]
ds_no_spike = ds.copy()
ds_no_spike['salinity'] = ds.salinity.where(~ds.profile_index.
 ↪isin(spiky_profiles))
```

```
spike_profiles = xr.DataArray([301,307,308], dims="bad_profiles") ##needed in␣
 ↪thermal lag correction

# Now adding zoomed plot
xlim_1 = [0, int(NUM_PROFILES/4)]
xlim_2 = [int(NUM_PROFILES/4), int(NUM_PROFILES/4*2)]
xlim_3 = [int(NUM_PROFILES/4*2), int(NUM_PROFILES/4*3)]
xlim_4 = [int(NUM_PROFILES/4*3), NUM_PROFILES]

Y_LIMS = [800, 0] #######modified

fig, axs = plt.subplots(4, 1, #height_ratios=[1, 4],
                        figsize = [12,9],
                        layout='constrained', sharex=False)



profile_lims = xlim_1
ds_sub = ds_no_spike.where((ds_no_spike.profile >=profile_lims[0]) &␣
 ↪(ds_no_spike.profile <= profile_lims[1]), drop=True)
ds_sub = ds_sub.isel(depth=range(200,800))

ax = axs[0]
pc = ax.pcolormesh(ds_sub.profile, ds_sub.depth,␣
 ↪ds_sub['salinity'],rasterized=True)
axs[0].set_ylim(Y_LIMS)

fig.colorbar(pc, ax=ax, label = 'Salinity [psu]')
axs[0].set_title('Salinity',loc='left')


########
profile_lims = xlim_2
ds_sub = ds_no_spike.where((ds_no_spike.profile >=profile_lims[0]) &␣
 ↪(ds_no_spike.profile <= profile_lims[1]), drop=True)

ax = axs[1]
pc = ax.pcolormesh(ds_sub.profile, ds_sub.depth,␣
 ↪ds_sub['salinity'],rasterized=True)
ax.set_ylim(Y_LIMS)

fig.colorbar(pc, ax=ax, label = 'Salinity [psu]')
axs[0].set_title('Salinity',loc='left')

######
profile_lims = xlim_3
```

```
ds_sub = ds_no_spike.where((ds_no_spike.profile >=profile_lims[0]) &␣
 ↪(ds_no_spike.profile <= profile_lims[1]), drop=True)

ax = axs[2]
pc = ax.pcolormesh(ds_sub.profile, ds_sub.depth,␣
 ↪ds_sub['salinity'],rasterized=True)
ax.set_ylim(Y_LIMS)

fig.colorbar(pc, ax=ax, label = 'Salinity [psu]')
axs[0].set_title('Salinity',loc='left')

#########
profile_lims = xlim_4
ds_sub = ds_no_spike.where((ds_no_spike.profile >=profile_lims[0]) &␣
 ↪(ds_no_spike.profile <= profile_lims[1]), drop=True)

ax = axs[3]
pc = ax.pcolormesh(ds_sub.profile, ds_sub.depth,␣
 ↪ds_sub['salinity'],rasterized=True)
ax.set_ylim(Y_LIMS)

fig.colorbar(pc, ax=ax, label = 'Salinity [psu]')
axs[0].set_title('Salinity',loc='left')

print('Zooming in along the glider deployment to see if spikes were removed')
axs[3].scatter(301,200,90,marker='v',color='r',zorder=2)
axs[3].scatter(307,200,90,marker='v',color='r',zorder=2)
axs[3].scatter(308,200,90,marker='v',color='r',zorder=2)



###########
```

```
Zooming in along the glider deployment to see if spikes were removed
```

[12]: <matplotlib.collections.PathCollection at 0x33ea12210>



Mask the spikes in the time series too

[13]:
```
### now we need to mask the values in the timeseries too
mask = ts['profile_index'].isin(spiky_profiles)
ts = ts.where(~mask,other=np.nan)
```

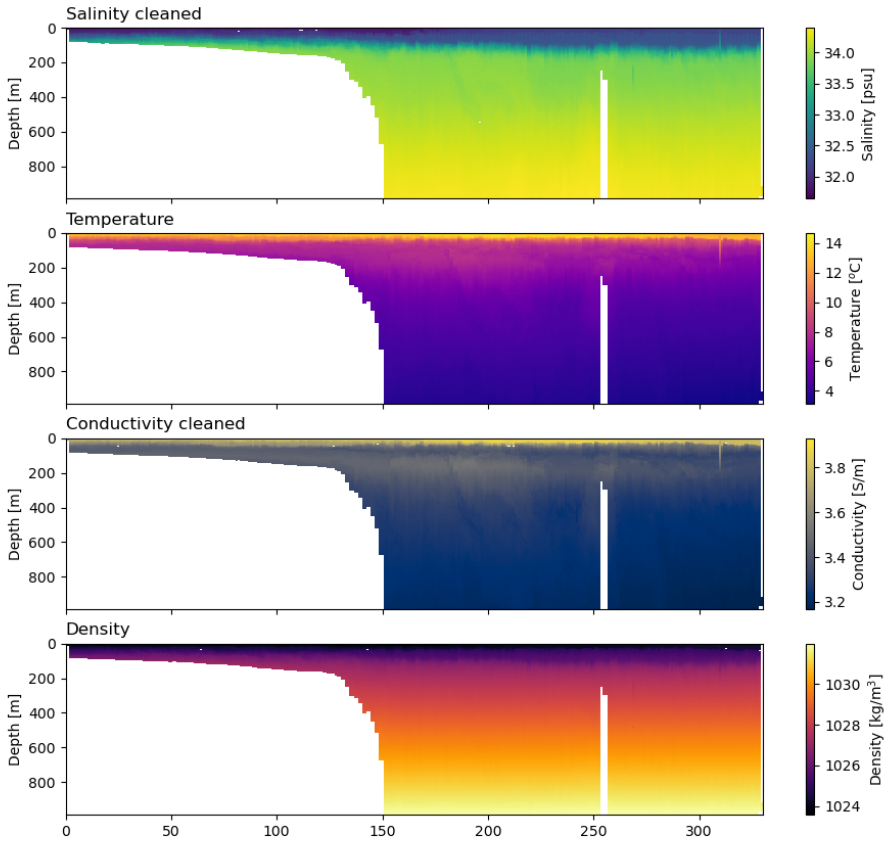
[14]:
```
###### grid to make finding bad profiles easier
ts.to_netcdf(f'{filepath}/{deploy_name}_conductivityClean.nc')
# Save a gridded version as well
outfile = make_gridfiles(f'{filepath}/{deploy_name}_conductivityClean.nc',␣
 ↪f'{filepath}', deployfile, fnamesuffix='condfilter')
```

Check to see if final plots look reasonable.

[15]:
```
# Open the grid file
print('Salinity, temperature, conductivity and density shown, with conductivity␣
 ↪outliers removed from the salinity, conductivity and density fields:')

ds=xr.open_dataset(f'{filepath}/{deploy_name}_gridcondfilter.nc')
# list(ds.keys())
```

```
# RE-PLOTTING WITH THE COND FILTER!
fig, axs = plt.subplots(4, 1, figsize=(11, 10), sharey=True, sharex=True)

xlims = [0, NUM_PROFILES]
ylims = [MAX_DEPTH,0]

pc = axs[0].pcolormesh(ds.profile, ds.depth, ds['salinity'],rasterized=True)
axs[0].set_ylim(ylims)
axs[0].set_xlim(xlims)
fig.colorbar(pc, ax=axs[0], label = 'Salinity [psu]')
axs[0].set_title('Salinity cleaned',loc='left')

pc = axs[1].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['temperature'],rasterized=True,cmap='plasma')
fig.colorbar(pc, ax=axs[1], label = 'Temperature [$^o$C]')
axs[1].set_title('Temperature',loc='left')

# from matplotlib import colors as c
# pc = axs[2].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['conductivity'],rasterized=True,cmap=c.ListedColormap(['r']))
pc = axs[2].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['conductivityClean'],rasterized=True,cmap='cividis')
fig.colorbar(pc, ax=axs[2], label = 'Conductivity [S/m]')
axs[2].set_title('Conductivity cleaned',loc='left')

# pc = axs[3].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['oxygen_concentration'],rasterized=True,cmap='inferno')
# fig.colorbar(pc, ax=axs[3])
# axs[3].set_title('Oxygen Concentration',loc='left')

pc = axs[3].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['density'],rasterized=True,cmap='inferno')
fig.colorbar(pc, ax=axs[3], label = 'Density [kg/m$^3$]')
axs[3].set_title('Density',loc='left')

# fig.supylabel('Depth [m]')
axs[0].set_ylabel('Depth [m]');
axs[1].set_ylabel('Depth [m]');
axs[2].set_ylabel('Depth [m]');
axs[3].set_ylabel('Depth [m]');
```

Salinity, temperature, conductivity and density shown, with conductivity outliers removed from the salinity, conductivity and density fields:



When plotting the fields shown above with the conductivity filter, we can see that the salinity and conductivity fields now have "speckles" showing data removed. This filtering is applied to remove values from the following fields:

- conductivity
- salinity (re-calculated)
- density (re-calculated)

The new field is called **conductivityClean**. The other fields mentioned were replaced.

3.3 2.2 Identifying questionable salinity profiles

Here, potentially suspicious salinity profiles are identified in order to prevent them from being used in the thermal lag correction. While these questionable salinity profiles are not included in the following steps, these profiles **are not removed** from the final corrected salinity product.

We identify any salinity profiles that are obviously unphysical, which is typically caused by something (usually biology) getting caught in the conductivity cell, and set all values within those

profiles to NaN. We use a simple criterion applied to the salinity data, binned by temperature, with bin sizes based on the time series mean temperature profile. The criterion temporarily flags any data points that are more than **4 standard deviations** away from the overall mean for the salinity time series within a given temperature bin, then recomputes the mean and standard deviation, excluding the temporarily flagged values. Salinity values that still differ from the mean by more than **4 standard deviations** are flagged as 'bad'. Finally, any profile where more than **10% of the salinity values** have been flagged as 'bad' using this criterion is removed. The number of standard deviations used and the percent of flagged required to flag a profile as 'bad' can be adjusted.

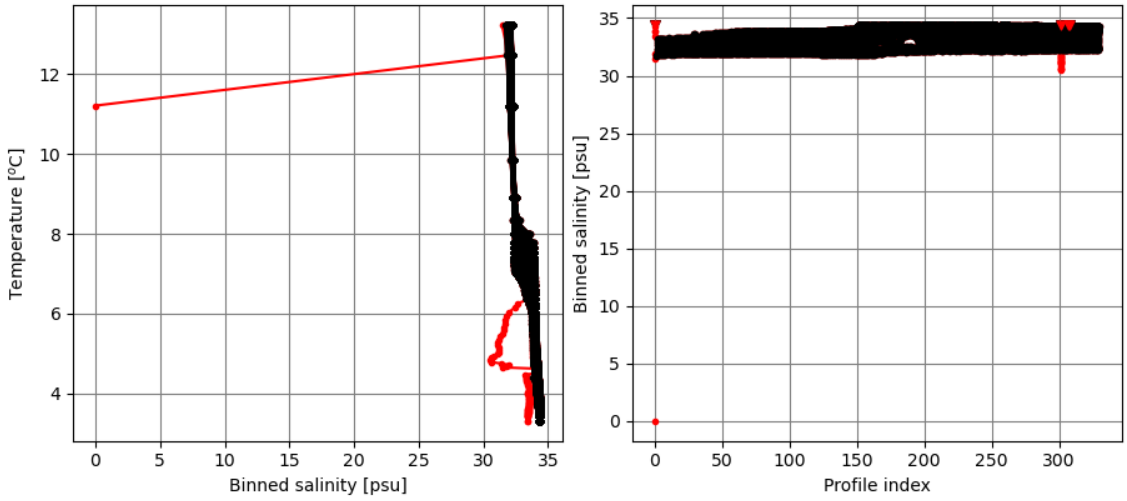
[62]:
```
###### code with spike free timeseries

# Determine time series mean temperature profile
fname = f'{filepath}/{deploy_name}_conductivityClean.nc'
gridfname = f'{filepath}/{deploy_name}_gridcondfilter.nc'

ds=xr.open_dataset(gridfname) # Not loading delayed mode data - instead loading␣
 ↪from intermediate files saved.
ds0 = xr.open_dataset(openfile) ##comparing to the original timeseries

####### find mean temperature profile of the timeseries
Tmean = ds['temperature'].mean(dim='time')
Tmean = Tmean.sortby(Tmean, ascending=True).where(np.isfinite(Tmean), drop=True)

# Identify the questionable salinity values
clean_profs = 0 #number of profiles to exclude from the start and end of the␣
 ↪time series
flag_stdev = 4 #number of standard deviations to temporarily flag bad salinity␣
 ↪values
clean_stdev = 4 #number of standard deviations to flag bad salinity values,␣
 ↪after removing the temporary bad values
clean_cutoff = 0.1 #fraction of bad salinity values required to label a profile␣
 ↪as bad
dtbin = 10 #number of temperature bins

#######trying modified code to exlude questionable profiles on the ends
sal = pgs.get_salinity_grid(ds0, Tmean, clean_profs, flag_stdev, clean_stdev,␣
 ↪clean_cutoff, dtbin)

sal.to_netcdf(f'{filepath}/SalinityGrid.nc')

bad_profiles = sal.profiles.where(sal.bad >= clean_cutoff, drop=True)
print('Profiles flagged as bad due to questionable salinity values:',␣
 ↪bad_profiles.values)
```

```
Profiles flagged as bad due to questionable salinity values: [  0. 301. 307.
308.]
```

4 were flagged as bad and had their values set to NaN

[63]:
```
caption = ('Binned salinity plotted as a function of temperature '
'(left) and vs. profile index (right), \n with the salinity profiles identified␣
 ↪'
'as bad due to questionable values and set to NaN shown in red and indicated by␣
 ↪'
'the red arrows at the top of the panel on the right:')

# Function to plot the salinity field with the values identified as bad
with xr.open_dataset(f'{filepath}/SalinityGrid.nc') as sal:
    fig, ax = plt.subplots(1,2,figsize=(9,4),
                           constrained_layout=True)

    sal.salinity.plot.line(ax=ax[0],
                           y='temperature',
                           color='r',
                           marker='.',
                           add_legend=False)
    sal.salinityGood.plot.line(ax=ax[0],
                               y='temperature',
                               color='k',
                               marker='.',
                               add_legend=False)
    ax[0].set_ylabel('Temperature [$^o$C]')
    ax[0].set_xlabel('Binned salinity [psu]')
    ax[0].grid(axis='both', color='0.5')

    sal.salinity.plot.line(ax=ax[1],
                           x='profiles',
                           color='r',
                           marker='.',
                           add_legend=False)
    sal.salinityGood.plot.line(ax=ax[1],
                               x='profiles',
                               color='k',
                               marker='.',
                               add_legend=False)
    ax[1].set_ylabel('Binned salinity [psu]')
    ax[1].set_xlabel('Profile index')
    ax[1].grid(axis='both', color='0.5')
    x = bad_profiles
    y = np.nanmax(sal.salinity.values) + np.zeros_like(bad_profiles)
    ax[1].scatter(x,y,30,marker='v',color='k',zorder=1)
    ax[1].scatter(x,y,25,marker='v',color='r',zorder=2)

    print(caption)
```

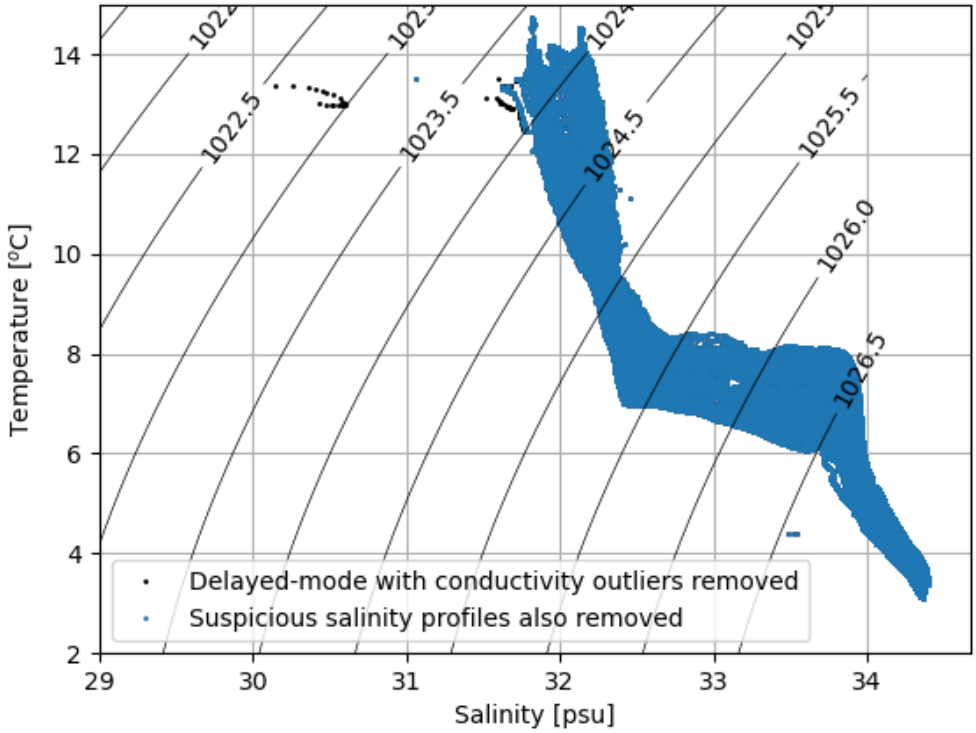
Binned salinity plotted as a function of temperature (left) and vs. profile index (right),
 with the salinity profiles identified as bad due to questionable values and set to NaN shown in red and indicated by the red arrows at the top of the panel on the right:



[64]:
```
# T-S diagram to select near-surface water density range to exclude

ts = xr.open_dataset(f'{filepath}/{deploy_name}_conductivityClean.nc')
ts = ts.assign_coords(pind=ts.profile_index)
srate = stats.mode((np.diff(ts.time)).astype('timedelta64[s]')).mode
fig,ax=plt.subplots()

ax.plot(ts.salinity,ts.temperature,'k.',markersize=2, label = 'Delayed-mode␣
 ↪with conductivity outliers removed')
# Now remove the bad profiles
idx = ~ts.profile_index.isin(bad_profiles)
ax.plot(ts.salinity[idx], ts.temperature[idx], '.', markersize = 2, label =␣
 ↪'Suspicious salinity profiles also removed')

#Create a density grid to contour plot isopycnals
S_range = np.linspace(int(np.min(ts.salinity)-0.5),
                      int(np.max(ts.salinity)+0.5), 1000)
T_range = np.linspace(int(np.min(ts.temperature)-1),
                      int(np.max(ts.temperature)+1), 1000)
S_grid, T_grid = np.meshgrid(S_range, T_range)
density_grid = seawater.eos80.dens0(S_grid, T_grid)

CS = ax.contour(S_range, T_range, density_grid,
                np.arange(1014,
```

```
                          np.round(np.max(density_grid)),0.5),
                colors='k', linewidths=0.5);
ax.clabel(CS, CS.levels, inline=True, fontsize=10)
ax.set_xlabel('Salinity [psu]')
ax.set_ylabel('Temperature [$^o$C]')
# plt.xlim(28,35)
ax.grid()
ax.legend(prop={'size': 10})
print('Temperature vs. salinity diagram. ',
     'Black contours give density in kg/m^3:')
```

```
Temperature vs. salinity diagram.  Black contours give density in kg/m^3:
```



[65]:
```
display(Markdown('./docs/CTD_2_Sensor_lag_LT.md'))
```

3.4 2.3 Sensor alignment correction

We now test application of a sensor alignment correction. In the literature this correction is often used to align the temperature and conductivity in time, relative to the pressure. This correction reduces the occurrence of salinity spikes near sharp gradients in T and S, and ensures calculations are made using the same parcel of water for all variables. The misalignment between the sensors is

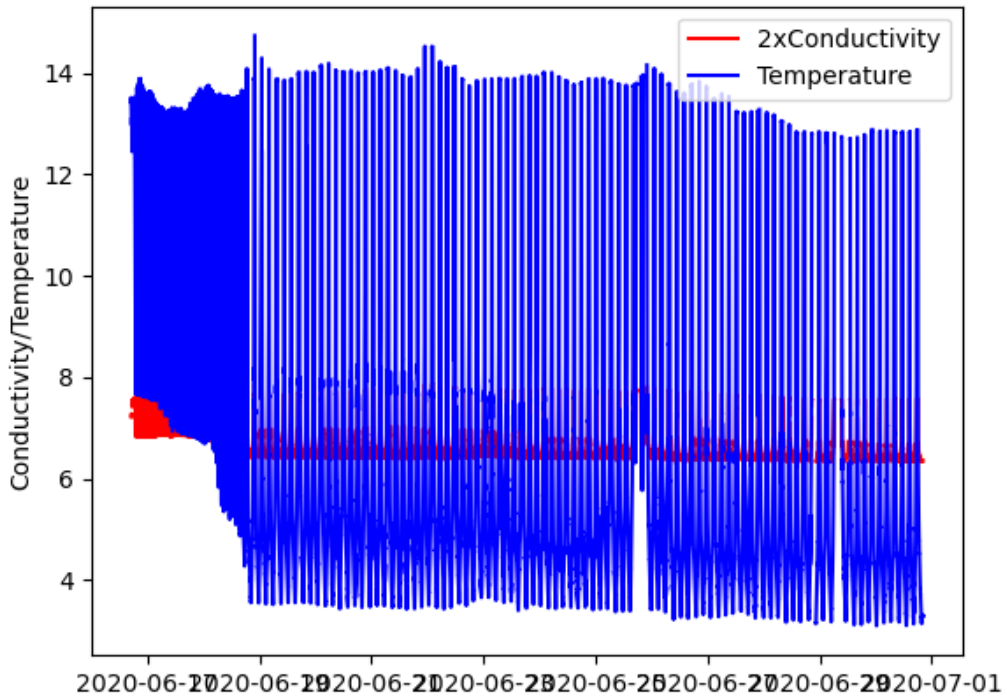
caused by: 1. The physical separation between sensors causing a transit time delay for water being pumped through the CTD, and, 2. Different sensor response times

We follow the SeaBird Electronics Data Processing Manual (page 80) to determine if there is any time lag between the temperature and conductivity sensors on our pumped CTD.

Sources: Sea-Bird Electronics, Inc. SEASOFT V2: SBE Data Processing (https://misclab.umeoce.maine.edu/ftp/instruments/CTD%2037SI%20June%202011%20disk/website/pdf_docun

[66]:
```
####open cleaned timeseries file
ts_clean = xr.open_dataset(f'{filepath}/{deploy_name}_conductivityClean.nc')
fig,ax=plt.subplots(sharex=True)

ax.plot(ts_clean.time,2*ts_clean.conductivityClean, c='red',␣
 ↪label='2xConductivity')
ax.set_ylabel('Conductivity/Temperature',)

ax.plot(ts_clean.time,ts_clean.temperature ,c='blue',label='Temperature')
ax.legend()

print('Plot timeseries of temperature and conductivity to observe any time␣
 ↪offset.')
```

```
Plot timeseries of temperature and conductivity to observe any time offset.
```



There is **no significant lag between the temperature (T) and conductivity (C) signals**. Examination of individual casts revealed no measurable offset between T and C. No visible lag from a 2 Hz sensor is reasonable. Furthermore, the T and C sensors on the Glider Pumped CTD (CPCTD) are spatially co-located, ebsuring both sensors sample the same parcel of water simultaneously. Therefore, **no sensor offset correction** was applied to the data.

[67]:
```
display(Markdown('./docs/CTD_2_Thermal_lag_LineP.md'))
```

3.5 2.4 Thermal lag correction

The thermal lag effect is caused by the thermal inertia of the conductivity cell affecting the temperature of the water as it passes through the cell. To determine the thermal lag correction, the temperature inside the conductivity cell is estimated, then salinity is recalculated using the estimated temperature and the measured conductivity. To estimate the temperature, a recursive filter is applied to the temperature field with parameters (the amplitude of the error), and (the time constant for the thermal lag). Two methods for this are mentioned below.

Sea-Bird GPCTDs are pumped with a constant flow rate. As such, we expect the thermal lag to be approximately constant over the full mission, and it is sufficient to find a single value of and for the entire mission. It is ideal to use profile pairs from regions with large temperature gradients, but small conductivity gradients, when comparing up- and down-casts.

Janzen and Creed (2011) determined a cell thermal mass correction for the GPCTD using data from a prototype CTD that sampled twice as rapidly as the GPCTD nominally samples, with a pumped flow rate of 10 ml/s. **They found = 0.06 and = 10s.** These values are considered when retrieving α and τ to see how much the results differ.

In this study, we consistently find 10 s, in agreement with Janzen and Creed (2011), and therefore determine as the free parameter that minimizes the RMSD while holding = 10 s constant.

This mission on the **Line P** occurred in a low energy environment of the shelf, so near-surface differences between a downcast and the subsequent upcast are unlikely to be caused by spatiotemporal variability. As such, we do not exclude segments of each profile in the upper water column from the minimization routine.

Considerations for using Janzen and Creed values: Prior processing used Janzen and Creed (2011) values α and τ for the thermal lag. For each mission, the thermal lag parameters were directly estimated. The steps are outlined below, but can be found in much greater detail here: **https://cproof.uvic.ca/gliderdata/deployments/reports/C-PROOF_SBE_CTDProcessingReport_v0.2.pdf**

- It was confirmed that the directly estimated value of τ was within $±$10s of the Janzen and Creed (2011) value of 10s
- The improvement with the directly estimated, as well as Janzen and Creed, parameters was quantified.
- The thermal lag correction was applied with the parameters that resulted in the greater improvement, and did not result in an over-correction.

- If a given sensor had a directly estimated value of τ that is significantly higher or lower than 10s, investigate further

Thermal lag parameters typically vary slightly among individual GPCTD sensors, more so than alignment correction constants. **Once an optimal is identified for a particular sensor, subsequent missions are tested with those constants**, and the correction is applied only if it improves the profile agreement. If not, the parameters are re-evaluated.

With this method, the recursive filter seeks to minimize the root-mean squared difference (RMSD), which is calculated as the square root of the sum of the squared areas between pairs of salinity profiles (binned by temperature), normalized by the number of pairs of profiles. The values of and that minimize the area between pairs of profiles (each dive and subsequent climb along the glider path) were determined using a brute force minimization scheme. This method also uses a subset of the remaining data, consisting of **100 pairs of profiles** equally spaced in time, to determine the correction.

Updated thermal lag correction procedure: Same to the above, this is based on Morison et al's (2011) second method which derives a modified temperature that is the "best guess" for what the temperature is in the conductivity cell based on the temperature observed by the thermistor. The temperature is corrected using `lfilter` which is just a recursive filter:

$T_T(n) = -bT_T(n-1) + aT(n) - aT(n-1)$

where

$$a = \frac{4f_n\alpha\tau}{1 + 4f_n\tau}$$

and

$$b = 1 - \frac{2a}{\alpha}$$

τ can be thought of as the time constant of the thermal lag (in seconds) and α as its strength. Following Gaurau et al, f_n is the sampling frequency. The cell temperature is then"

$$T_c(n) = T(n) - T_T(n)$$

and can be used to calculate salinity with the measured conductivity and pressure.

3.5.1 2.4.1 Pre-processing steps:

We **exlude** profiles from the correction for which the area between subsequent downcasts is more than one standard deviation from the mission mean. This ensures that no data crossing fronts or intrusions is included in the correction, in line with the key assumption that a downcast and the subsequent upcast be identical.

Furthermore, we impose a cutoff for the area between pairs of profiles that will be included in the subset used to estimate the parameters. Any pair of profiles whose area is more than 3 standard deviations away from the mean will be excluded from the determination of the RMSD. This ensures that a small number of anomalous profiles do not bias the results.

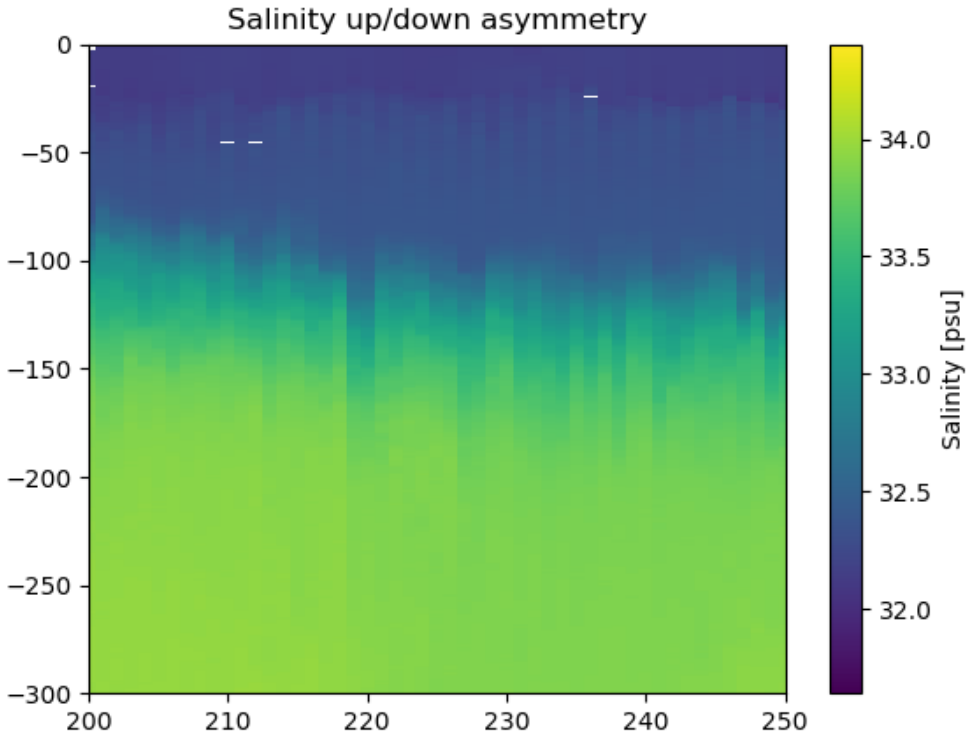
During this step, the **suspicious salinity profiles** identified earlier are excluded as well.

[68]:
```
# Set up our constants
fn = 0.5*fs #frequency for Sea-Bird GPCTD
density_cutoff = 1021 #this is not excluding the top of profiles (exclude␣
 ↪everything less dense than this from the minimization)
num_profs = 100 #number of profiles to include in the subset of data # This is␣
 ↪not used for the correction, but is used in the q/c steps
clean_profs_start = 50 #number of profiles to exclude from the start
clean_profs_end = 0 #number of profiles to exclude from the end
dn_stdev = 1 #how many standard deviations from the mean the area between␣
 ↪downcasts can be

fname = f'{filepath}/{deploy_name}_conductivityClean.nc'

# Load time series
print(f'Loading filename {fname}')
ts = xr.load_dataset(fname)#ds1.copy(deep=True)
ts = ts.assign_coords(pind=ts.profile_index) #add a profile index coordinate
tot_profs = int(np.nanmax(ts.profile_index.values))
print('Total number of profiles:', tot_profs)

# Overwrite conductivity in our working xarray with the clean, aligned␣
 ↪conductivity field
ts['conductivity'] = ts.conductivityClean
```

```
Loading filename deployments/dfo-walle652/dfo-walle652-20200616//dfo-
walle652-20200616_conductivityClean.nc
Total number of profiles: 330
```

[69]:
```
ds0 = xr.open_dataset(f'{filepath}/{deploy_name}_gridcondfilter.nc' )
fig, ax = plt.subplots()
pc=ax.pcolormesh(ds0.profile, -ds0.depth, ds0['salinity'],rasterized=True)
fig.colorbar(pc,label='Salinity [psu]')
ax.set_title('Salinity up/down asymmetry')
ax.set_xlim(200,250)
ax.set_ylim(-300,0)


print('There is a visible asymmetry in the up and down casts of succesive␣
 ↪profiles, causing stipes to appear in the raw data' )
```

```
There is a visible asymmetry in the up and down casts of succesive profiles,
causing stipes to appear in the raw data
```



We **exlude the first 50 profiles**, which were primarily collected in the highly energetic environment on the shelf near Tofino. We use a subet of the remianing data, consisting of **100 pairs of profiles** equally spaced in time to determine the corrected.

[70]:
```
print('Calculating profile pairs')
ts_sub, profile_bins, profile_bins_all, direction = pgs.profile_pairs(
    ts, clean_profs_start, clean_profs_end, num_profs, bad_profiles
) ##need to account for the NaN profiles added when correcting biofouling

# Identify boolean index for application of density cutoff
density_bool = ts_sub.density>=density_cutoff
```

```
Calculating profile pairs
```

Restricting profiles $>$ **1 standard deviation** greater than the mean space between profiles

[71]:
```
#Determine the area between subsequent downcasts to restrict profiles included␣
 ↪in correction
print(f'Restricting profiles')
dn_area, area_bad = pgs.TS_preprocess(
    density_bool, dn_stdev,
    profile_bins, profile_bins_all,
```

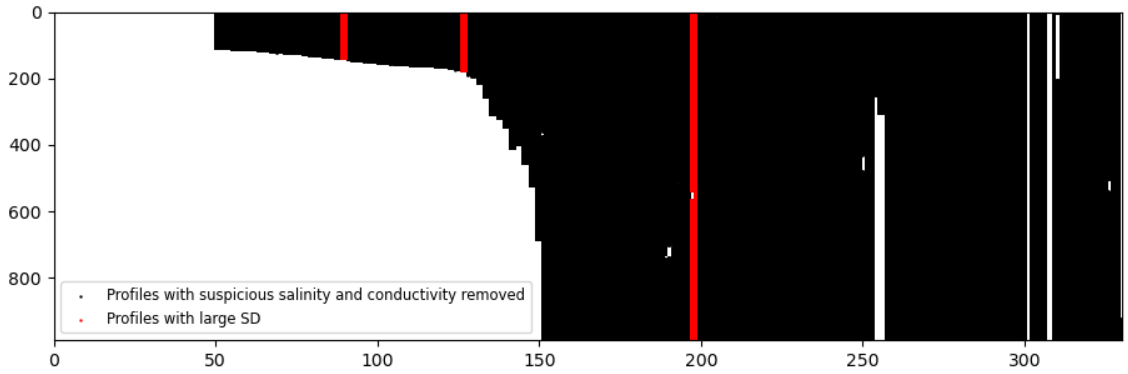
```
    direction, ts_sub)
print('Max and min area between downcasts = ', np.nanmax(dn_area), np.
 ↪nanmin(dn_area))

ts_bad = ts_sub.where(
    ts_sub.profile_index.isin(
        profile_bins_all[area_bad]),
        drop=True)
prof_list = ts_bad.profile_index
print('List of profiles to exclude:', np.unique(prof_list.values))
```

```
Restricting profiles
Max and min area between downcasts =  3.528591935584152 0.0011428285848211657
List of profiles to exclude: [ 89.  90. 126. 127. 197. 198.]
```

[72]:
```
ts_sub['profiles_to_exclude'] = ts_sub.profile_index.isin(prof_list.values)
```

[73]:
```
# Plot the profiles that were kept for the comparison!!
print('Red indicates profile pairs that were identified in this process, where␣
 ↪the area between \nprofile pairs was considered to be too large, and so are␣
 ↪not included in the thermal lag correction. \nWhite bands indicate salinity␣
 ↪profiles removed during step 2.2.')

subdata = ts_sub.where(ts_sub.profiles_to_exclude == True) #profile_index.
 ↪isin(prof_list.values)#(profile_bins)

fig, ax = plt.subplots(1,1, figsize=(9, 3), constrained_layout=True)

ax.scatter(ts_sub.profile_index, ts_sub.pressure, marker = '.', color='black',␣
 ↪s = 2,
           rasterized=True, label='Profiles with suspicious salinity and␣
 ↪conductivity removed')


ax.set_ylim([MAX_DEPTH, 0])
ax.set_xlim([0,np.nanmax(ts_sub.profile_index)])

ax.scatter(subdata.profile_index, subdata.pressure,
           color='red', marker = '.', s=2, rasterized=True, label = 'Profiles␣
 ↪with large SD')

ax.legend(fontsize='small', loc='lower left');
```

```
Red indicates profile pairs that were identified in this process, where the area
between
profile pairs was considered to be too large, and so are not included in the
thermal lag correction.
White bands indicate salinity profiles removed during step 2.2.
```



[74]:
```
# SAVING INTERMEDIATE FILE TO NETCDF
# print("Saving thermal lag preprocessing files to netcdf")
ts_sub.to_netcdf(f'{filepath}/{deploy_name}_goodprofiles.nc')
# Save a gridded version as well
outfile = make_gridfiles(f'{filepath}/{deploy_name}_goodprofiles.nc',␣
 ↪f'{filepath}', deployfile, fnamesuffix='goodprofiles')
```

3.5.2 2.4.2 Defining the range to calculate α and τ:

From examining the asymmetry in up-down profiles, we manually choose a range to apply the thermal lag correction to. It is ideal to pick areas with high temperature gradients in the water column, but generally low salinity gradients.

[75]:
```
fname = f'{filepath}/{deploy_name}_goodprofiles.nc'
gridfname= f'{filepath}/{deploy_name}_gridgoodprofiles.nc'
```

[76]:
```
# Select a subset of profiles to calculate tau and alpha
profile_lims = [200,230]
print(f'Using profile limits {profile_lims} for tau and alpha calculation')
```

```
Using profile limits [200, 230] for tau and alpha calculation
```

[77]:
```
ds=xr.open_dataset(f'{filepath}/{deploy_name}_gridgoodprofiles.nc')


tbins = ds.temperature.mean(dim='time', skipna=True)
tbins = np.sort(tbins[::6])
tbins = tbins[np.isfinite(tbins)]
# print(tbins)
depbins = ds.depth[::6]
```

[78]:
```
# Switches back to the time series....
# switching ds to ts
with xr.load_dataset(fname) as ds0:
```

```
    # USING PROFILE_LIMS DECIDED ABOVE!
    # print(f'Loading {fname}')
    inds=np.arange(profile_lims[0], profile_lims[1])

    indbins = np.arange(inds[0]-0.5, inds[-1]+0.5, 1.0)

    ts = ds0.where((ds0.profile_index >= inds[0]) & (ds0.profile_index <=␣
 ↪inds[-1]), drop=False)
    ts = ts.where(ts.depth >=200) ######### modified... just look deeper in the␣
 ↪water column

    # Once again, make sure to use density cutoff
    ts = ts.where((ts.density > density_cutoff), drop=True)
    # Also, profiles to exclude:
    ts = ts.where(ts.profiles_to_exclude == False, drop=True)

    sal = ts.salinity
```

3.5.3 Comparing the error measurements between estimated alpha & tau and Janzen and Creed values:

Below shows the subset of profiles, limited to 200 m depth, without any thermal lag correction, and using literature values (Janzen and Creed 2011). The Janzen and Creed α and τ values visibly reduce the error in the water column, but better results can be retrieved by calculating our own.

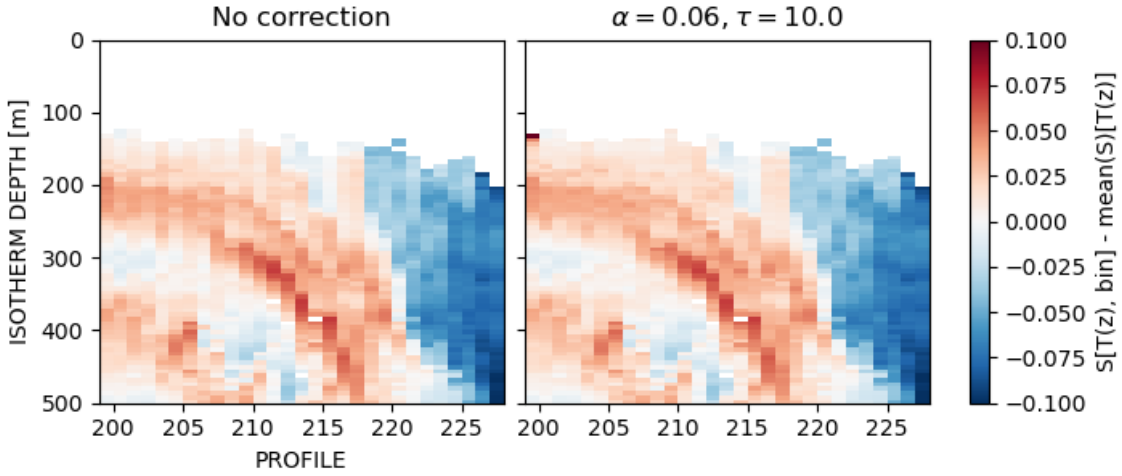
First we **bin our salinity data** into temperature bins of width 0.1 and profile index. We **sum the salinities in each bin and divide by the number of samples in each bin**. Error is the **difference in the mean salinity for successive profiles**, normlaized by salinity variance for each temperature bin.

[79]:
```
# Comparison - correcting the salinity with the janzen and creed values, and␣
 ↪comparing the error before and after
dt = ts.time.diff(dim='time').mean(dim='time').astype('float') / 1e9
fn = 1.0 / dt
alpha = 0.06
tau = 10.0

sal = pgs.correct_sal(ts, fn, alpha, tau) ###applied lag correction formula to␣
 ↪"good" salinity values

ss0, err0, totalerr = pgs.get_error(ts, ts.salinity, tbins, indbins) ### "good"␣
 ↪data with no lag correction
ss, err, totalerr = pgs.get_error(ts, sal, tbins, indbins)
```

[80]:
```
sp0 = np.nanmean(ss0, axis=1)

Y_LIMS = [500,0]
```

```
fig, ax = plt.subplots(1, 2, sharex=True, sharey=True,␣
 ↪layout='constrained',figsize = [7,3])
pc = ax[0].pcolormesh(indbins[:-1], depbins[:len(tbins)-1][::-1], ss0-sp0[:, np.
 ↪newaxis], cmap='RdBu_r', vmin=-0.1, vmax=0.1) #, vmin=3.34, vmax=3.52)
ax[0].set_ylim(Y_LIMS)
ax[0].set_ylabel('ISOTHERM DEPTH [m]')
ax[0].set_xlabel('PROFILE')
ax[0].set_title('No correction')

pc = ax[1].pcolormesh(indbins[:-1], depbins[:len(tbins)-1][::-1], ss-sp0[:, np.
 ↪newaxis], cmap='RdBu_r', vmin=-0.1, vmax=0.1) #, vmin=3.34, vmax=3.52)
ax[1].set_title(f'$\\alpha = {alpha}, \\tau = {tau}$')
fig.colorbar(pc, ax=ax, label='S[T(z), bin] - mean(S)[T(z)]')
```

[80]: <matplotlib.colorbar.Colorbar at 0x37fcbe0d0>



As seen above, the up/down asymmetry was slightly improved by applying the Janzen and Creed constants.

3.5.4 Finding alpha and tau values with lowerest error estimates:

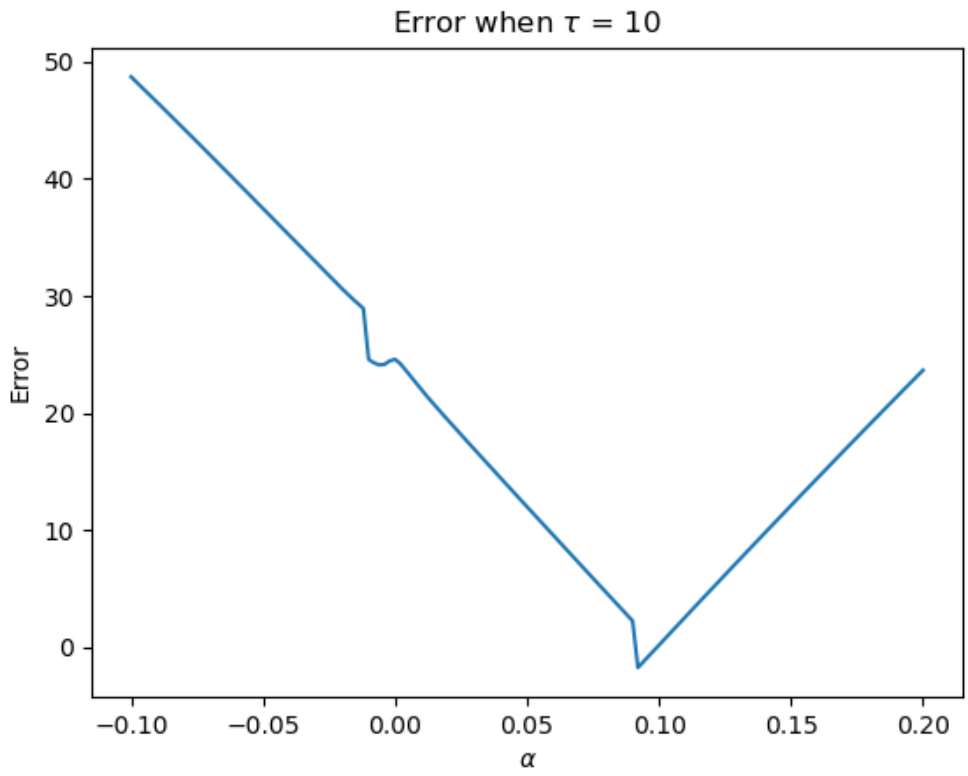
We will keep τ as 10, since the Janzen and Creed constants improved the data. We will try to find α that minimizes error.

[81]:
```
## scan:
alphas = np.arange(-0.1, 0.2, 0.002)
tau = 10


errors = np.zeros((len(alphas)))
for ny, alpha in enumerate(alphas):
```

```
    sal = pgs.correct_sal(ts, fn, alpha, tau)
    ss, err, totalerr = pgs.get_error(ts, sal, tbins, indbins)
    errors[ny] = np.nansum(np.nanmean(err, axis=1))
```

[82]:
```
fig,axs=plt.subplots()

axs.plot(alphas, errors)
axs.set_xlabel(r'$\alpha$')
axs.set_ylabel('Error')
axs.set_title(r'Error when $\tau$ = 10')

min_idx = np.unravel_index(np.argmin(errors), errors.shape)
print('The minimim alpha is ' + str(alphas[min_idx]))
```

```
The minimim alpha is 0.09200000000000016
```



There is a global minimum where the error is minimized. We will use that α to reduce the asymmetry.

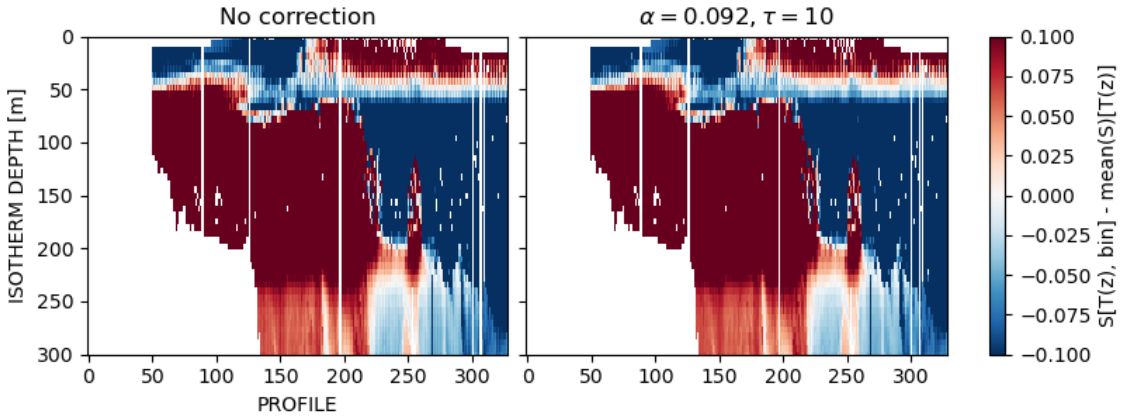
[83]:
```
dt = ts.time.diff(dim='time').mean(dim='time').astype('float') / 1e9
fn = 1.0 / dt
```

```
alpha = 0.092
tau = 10

print('******')
print(f'Applying alpha = {alpha} and tau = {tau}')
print('*****')

alpha2 = 0.06
tau2 = 10
##########

with xr.load_dataset(fname) as ts0:
    inds = np.arange(0, NUM_PROFILES)

    indbins = np.arange(inds[0]-0.5, inds[-1]+0.5, 1.0)

    ts = ts0.where((ts0.profile_index >= inds[0]) & (ts0.profile_index <=␣
 ↪inds[-1]), drop=False)
    # Once again, make sure to use density cutoff? maybe?
    ts = ts.where((ts.density > density_cutoff), drop=True)
    # Also, profiles to exclude:
    ts = ts.where(ts.profiles_to_exclude == False, drop=True)

    sal = pgs.correct_sal(ts, fn, alpha, tau)

    ss0, err0, totalerr = pgs.get_error(ts, ts.salinity, tbins, indbins)
    ss, err, totalerr = pgs.get_error(ts, sal, tbins, indbins)

    sal2 = pgs.correct_sal(ts, fn, alpha2, tau2)
    ss2, err2, totalerr2 = pgs.get_error(ts, sal2, tbins, indbins)


Y_LIMS = [300,0]

fig, ax = plt.subplots(1, 2, sharex=True, sharey=True, layout='constrained',␣
 ↪figsize = [8,3])
sp0 = np.nanmean(ss0, axis=1)
pc = ax[0].pcolormesh(indbins[:-1], depbins[:len(tbins)-1][::-1], ss0-sp0[:, np.
 ↪newaxis], cmap='RdBu_r', vmin=-0.1, vmax=0.1) #, vmin=3.34, vmax=3.52)
ax[1].set_ylim([200, 0])
ax[0].set_ylabel('ISOTHERM DEPTH [m]')
ax[0].set_xlabel('PROFILE')
ax[0].set_title('No correction')
ax[0].set_ylim(Y_LIMS)

pc = ax[1].pcolormesh(indbins[:-1], depbins[:len(tbins)-1][::-1], ss-sp0[:, np.
 ↪newaxis], cmap='RdBu_r', vmin=-0.1, vmax=0.1) #, vmin=3.34, vmax=3.52)
```

```
ax[1].set_title(f'$\\alpha = {alpha}, \\tau = {tau}$')
fig.colorbar(pc, ax=ax, label='S[T(z), bin] - mean(S)[T(z)]')
```

```
******
Applying alpha = 0.092 and tau = 10
*****
```

[83]: <matplotlib.colorbar.Colorbar at 0x37fb59d10>

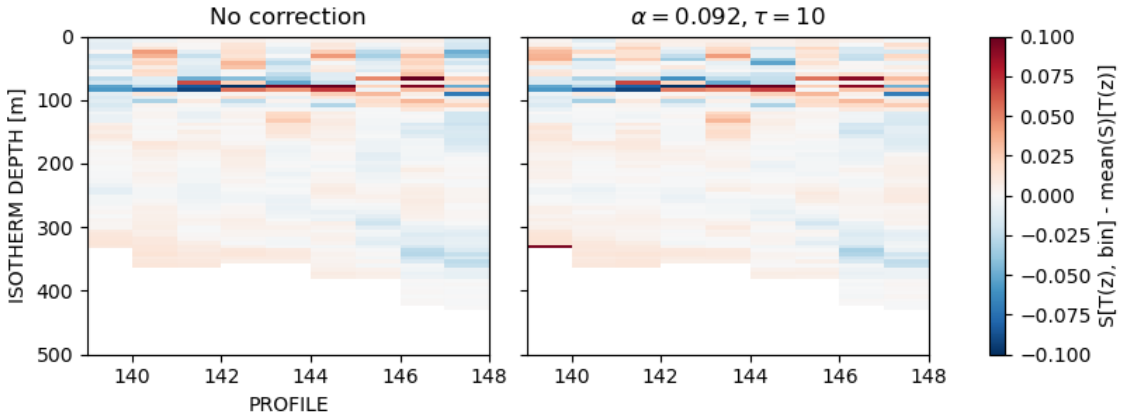


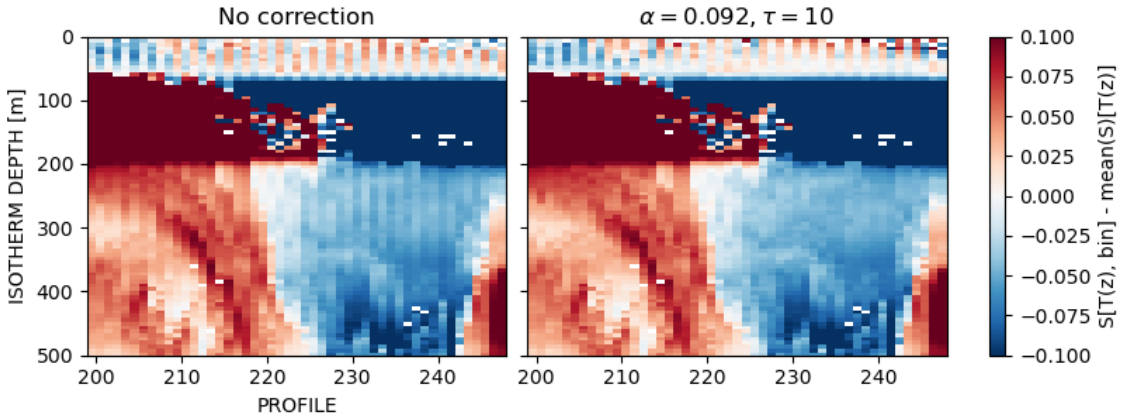
Zoom into three areas to observe change

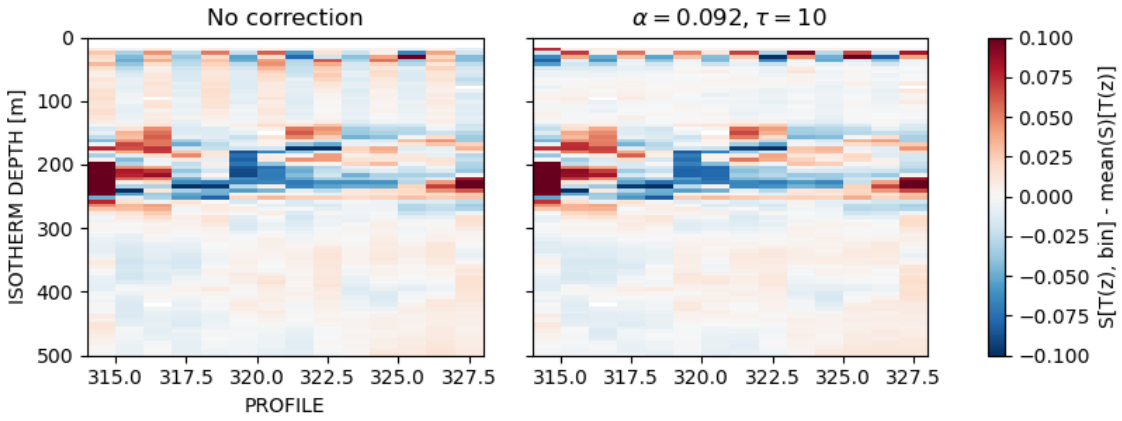
[84]:
```
def zoom_in_thermal_chg(min_ind,max_ind,fname,density_cutoff):
    with xr.load_dataset(fname) as ts0:
        inds = np.arange(min_ind, max_ind)

        indbins = np.arange(inds[0]-0.5, inds[-1]+0.5, 1.0)
        ts = ts0.where((ts0.profile_index >= inds[0]) & (ts0.profile_index <=␣
 ↪inds[-1]), drop=False)
        # Once again, make sure to use density cutoff? maybe?
        ts = ts.where((ts.density > density_cutoff), drop=True)
        # Also, profiles to exclude:
        ts = ts.where(ts.profiles_to_exclude == False, drop=True)

        sal = pgs.correct_sal(ts, fn, alpha, tau)

        ss0, err0, totalerr = pgs.get_error(ts, ts.salinity, tbins, indbins)
        ss, err, totalerr = pgs.get_error(ts, sal, tbins, indbins)

        sal2 = pgs.correct_sal(ts, fn, alpha2, tau2)
        ss2, err2, totalerr2 = pgs.get_error(ts, sal2, tbins, indbins)


    Y_LIMS = [500,0]
```

```
   fig, ax = plt.subplots(1, 2, sharex=True, sharey=True,␣
↪layout='constrained', figsize = [8,3])
   sp0 = np.nanmean(ss0, axis=1)
   pc = ax[0].pcolormesh(indbins[:-1], depbins[:len(tbins)-1][::-1], ss0-sp0[:
↪, np.newaxis], cmap='RdBu_r', vmin=-0.1, vmax=0.1) #, vmin=3.34, vmax=3.52)
   ax[1].set_ylim([200, 0])
   ax[0].set_ylabel('ISOTHERM DEPTH [m]')
   ax[0].set_xlabel('PROFILE')
   ax[0].set_title('No correction')
   ax[0].set_ylim(Y_LIMS)

   pc = ax[1].pcolormesh(indbins[:-1], depbins[:len(tbins)-1][::-1], ss-sp0[:,␣
↪np.newaxis], cmap='RdBu_r', vmin=-0.1, vmax=0.1) #, vmin=3.34, vmax=3.52)
   ax[1].set_title(f'$\\alpha = {alpha}, \\tau = {tau}$')
   fig.colorbar(pc, ax=ax, label='S[T(z), bin] - mean(S)[T(z)]')
```

[85]:
```
zoom_in_thermal_chg(140,150,fname,density_cutoff)
zoom_in_thermal_chg(200,250,fname,density_cutoff)
zoom_in_thermal_chg(315,330,fname,density_cutoff)
```







The corrections seem to reduce asymmetry everywhere in the time series. We are using this α as the constant for GPCTD 9404

Save corrected data

[86]:
```
print('*****')
print(f'Saving with alpha = {alpha} and tau = {tau} applied')
print('*****')

# We're going to apply the changes to the un-filtered data. That step happened␣
 ↪last here:

fname = f'{filepath}/{deploy_name}_conductivityClean.nc'
gridfname = f'{deploy_name}_gridcondfilter.nc'
```

```
*****
Saving with alpha = 0.092 and tau = 10 applied
*****
```

[87]:
```
ts.close()
ds.close()
```

[88]:
```
with xr.open_dataset(fname) as ts:
    display(ts)
    ts['conductivity'] = ts.conductivityClean
    s, t, d = pgs.correct_sal_temp_dens(ts, fn, alpha, tau)
    ts['salinity_corrected'] = ('time', s)
    ts['temperature_adjusted'] = ('time', t)
    ts['density_adjusted'] = ('time', d)
    # fix attributes
    ts.to_netcdf(f'{filepath}/{deploy_name}_thermal_lag.nc')
    display(ts)

outfile = make_gridfiles(f'{filepath}/{deploy_name}_thermal_lag.nc',
                         f'{filepath}',
                         deployfile, fnamesuffix='_thermal_lag')
```

```
<xarray.Dataset> Size: 127MB
Dimensions:                   (time: 567914)
Coordinates:
  * time                      (time) datetime64[ns] 5MB 2020-06-16T14:21:08 …
    latitude                  (time) float64 5MB …
    longitude                 (time) float64 5MB …
    depth                     (time) float64 5MB …
Data variables: (12/24)
    heading                   (time) float64 5MB …
    pitch                     (time) float64 5MB …
    roll                      (time) float64 5MB …
    waypoint_latitude         (time) float64 5MB …
    waypoint_longitude        (time) float64 5MB …
    conductivity              (time) float64 5MB …
    …                        …
    potential_density         (time) float64 5MB …
    density                   (time) float64 5MB …
    potential_temperature     (time) float64 5MB …
    profile_index             (time) float64 5MB …
    profile_direction         (time) float64 5MB …
    conductivityClean         (time) float64 5MB …
Attributes: (12/62)
    Conventions:              CF-1.6
    Metadata_Conventions:     CF-1.6, Unidata Dataset Discovery v1.0
    acknowledgement:          Funding from Fisheries and Oceans Canada, Cana…
    cdm_data_type:            Trajectory
    comment:                  Line P deployment
    contributor_name:         James Pegg, Tetjana Ross,  Jody Klymak, Hayley…
    …                        …
    summary:                  Glider deployed as part of the C-PROOF glider …
```

```
    time_coverage_end:          2020-07-01T16:00:39.000000000
    time_coverage_start:        2020-06-16T14:21:08.000000000
    title:                      dfo-walle652-20200616T1421
    transmission_system:        IRIDIUM
    wmo_id:                     4800996

<xarray.Dataset> Size: 141MB
Dimensions:                    (time: 567914)
Coordinates:
  * time                       (time) datetime64[ns] 5MB 2020-06-16T14:21:08 …
    latitude                   (time) float64 5MB …
    longitude                  (time) float64 5MB …
    depth                      (time) float64 5MB …
Data variables: (12/27)
    heading                    (time) float64 5MB …
    pitch                      (time) float64 5MB …
    roll                       (time) float64 5MB …
    waypoint_latitude          (time) float64 5MB …
    waypoint_longitude         (time) float64 5MB …
    conductivity               (time) float64 5MB nan nan nan nan … nan nan nan
    …                          …
    profile_index              (time) float64 5MB …
    profile_direction          (time) float64 5MB …
    conductivityClean          (time) float64 5MB …
    salinity_corrected         (time) float64 5MB nan nan nan nan … nan nan nan
    temperature_adjusted       (time) float64 5MB nan nan nan nan … nan nan nan
    density_adjusted           (time) float64 5MB nan nan nan nan … nan nan nan
Attributes: (12/62)
    Conventions:               CF-1.6
    Metadata_Conventions:      CF-1.6, Unidata Dataset Discovery v1.0
    acknowledgement:           Funding from Fisheries and Oceans Canada, Cana…
    cdm_data_type:             Trajectory
    comment:                   Line P deployment
    contributor_name:          James Pegg, Tetjana Ross,  Jody Klymak, Hayley…
    …                          …
    summary:                   Glider deployed as part of the C-PROOF glider …
    time_coverage_end:         2020-07-01T16:00:39.000000000
    time_coverage_start:       2020-06-16T14:21:08.000000000
    title:                     dfo-walle652-20200616T1421
    transmission_system:       IRIDIUM
    wmo_id:                    4800996
```

3.5.5 Before and after plots of the thermal lag correction

The **following plots** show the adjusted salinity, temperature, and density, respectively. The insets, right, correspond to the **red box**, delineating the profile range used to calculate the alpha and tau values. Further, unadjusted salinity, temperature, and density outliers, identified during the pre-processing steps, which were removed for the tau and alpha calculation to not skew results, are

shaded in a lighter colour.

These fields, adjusted and re-calculated using the new alpha and tau values, are saved in the output file as fields **salinity_adjusted**, **temperature_adjusted** and **density_adjusted**. The original delayed-mode, uncorrected fields are saved as **salinity**, **temperature** and **density**.

[89]:
```
# Open the ts file
ts=xr.load_dataset(f'{filepath}/{deploy_name}_thermal_lag.nc')
```

[90]:
```
# Open the grid file
ds=xr.open_dataset(f'{filepath}/{deploy_name}_grid_thermal_lag.nc')

dsbad = xr.open_dataset(f'{filepath}/{deploy_name}_gridgoodprofiles.nc')

# idx = dsbad.where(dsbad.profiles_to_exclude ==True)
# dsbad.salinity[idx] = np.nan

# Also testing dropping the bad profiles
dsbad = dsbad.where(dsbad.profiles_to_exclude==False, drop=False)

# Similarly, applying a density cutoff
dsbad = dsbad.where(dsbad.density>=density_cutoff, drop = False)
print(f'Density cutoff: {density_cutoff}')

# Trying to make a mask of the good and bad data used to derive correction␣
 ↪parameters from
ds['salinity_mask'] = ds.salinity.isin(dsbad.salinity)
```

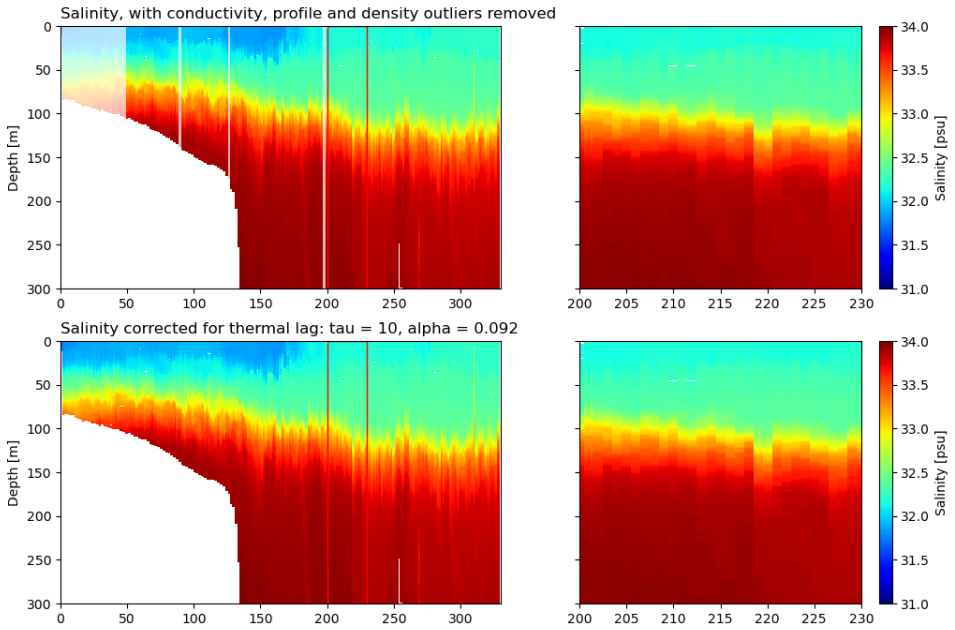
```
Density cutoff: 1021
```

[91]:
```
# RE-PLOTTING WITH THE COND FILTER!
%matplotlib ipympl

fig, axs = plt.subplots(2, 2, figsize=(12, 8), sharey=True, width_ratios=[2.
 ↪5,2], squeeze=True)

# Limits
xlims = [0, NUM_PROFILES]
ylims=[300,0] #[MAX_DEPTH,0]
sal_lims = [31,34]
cmap='jet'
profile_lims = [200,230]###defined earlier

# Rectangle showing area used to get alpha and tau
from matplotlib.patches import Rectangle
```

```
pc = axs[0,0].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['salinity'],cmap=cmap,rasterized=True,vmin=sal_lims[0], vmax=sal_lims[1],␣
 ↪alpha = 0.3 )
pc = axs[0,0].pcolormesh(dsbad.profile, dsbad.depth,␣
 ↪dsbad['salinity'],rasterized=True,vmin=sal_lims[0],␣
 ↪vmax=sal_lims[1],cmap=cmap)
axs[0,0].set_ylim(ylims)
axs[0,0].set_xlim(xlims)
# fig.colorbar(pc, ax=axs[0,0])
axs[0,0].set_ylabel('Depth [m]')
axs[0,0].set_title('Salinity, with conductivity, profile and density outliers␣
 ↪removed',loc='left')
axs[0,0].add_patch(Rectangle(xy=(profile_lims[0], ylims[1]), width =␣
 ↪profile_lims[1]-profile_lims[0], height = ylims[0]-ylims[1],
                        edgecolor = 'r', facecolor='none', linewidth=1));

pc = axs[1,0].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['salinity_corrected'],rasterized=True,vmin=sal_lims[0],␣
 ↪vmax=sal_lims[1],cmap=cmap)
axs[1,0].set_xlim(xlims)
# fig.colorbar(pc, ax=axs[1,0])
axs[1,0].set_ylabel('Depth [m]')
axs[1,0].set_title('Salinity corrected for thermal lag: tau = 10, alpha = 0.
 ↪092',loc='left')
axs[1,0].add_patch(Rectangle(xy=(profile_lims[0], ylims[1]), width =␣
 ↪profile_lims[1]-profile_lims[0], height = ylims[0]-ylims[1],
                        edgecolor = 'r', facecolor='none', linewidth=1));


# Subplots from correction area

xlims = profile_lims
# Subplot from correction area
pc = axs[0,1].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['salinity'],rasterized=True,vmin=sal_lims[0], vmax=sal_lims[1], alpha = 0.
 ↪3,cmap=cmap)
pc = axs[0,1].pcolormesh(dsbad.profile, dsbad.depth,␣
 ↪dsbad['salinity'],rasterized=True,vmin=sal_lims[0],␣
 ↪vmax=sal_lims[1],cmap=cmap)
axs[0,1].set_xlim(xlims)
fig.colorbar(pc, ax=axs[0,1], label = 'Salinity [psu]')

pc = axs[1,1].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['salinity_corrected'],rasterized=True,vmin=sal_lims[0],␣
 ↪vmax=sal_lims[1],cmap=cmap)
axs[1,1].set_xlim(xlims)
```

```
fig.colorbar(pc, ax=axs[1,1], label = 'Salinity [psu]')


print('Salinity before (top) and after adjustment (bottom)')
print('Profiles and data points that were ignored for thermal lag correction␣
 ↪are shaded in the top plot.')
print('The profile range used to determine the thermal lag correction is shown␣
 ↪on the right:')
```

```
Salinity before (top) and after adjustment (bottom)
Profiles and data points that were ignored for thermal lag correction are shaded
in the top plot.
The profile range used to determine the thermal lag correction is shown on the
right:
```

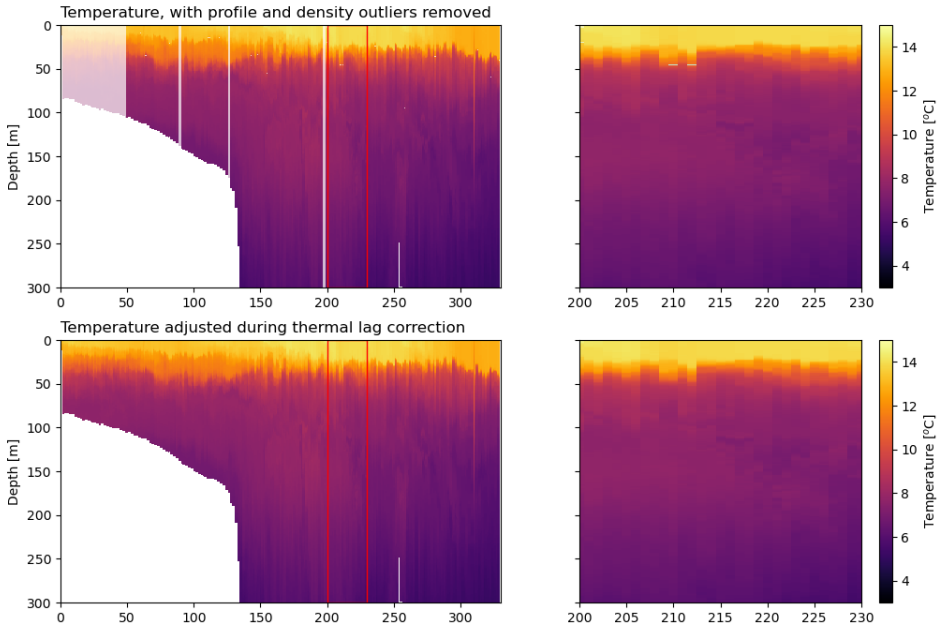


[92]:
```
# RE-PLOTTING WITH THE COND FILTER!
fig, axs = plt.subplots(2, 2, figsize=(12, 8), sharey=True, width_ratios=[2.
 ↪5,2], squeeze=True)

# Limits
xlims = [0, NUM_PROFILES]
ylims=[300,0]
t_lims = [3,15]
```

```
cmap = 'inferno'
# Rectangle showing area used to get alpha and tau
from matplotlib.patches import Rectangle
# print(f'Drawing rectangle centered at: {profile_lims[0], ylims[0]}')
# rect = #, linestyle='dotted'

pc = axs[0,0].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['temperature'],rasterized=True,vmin=t_lims[0], vmax=t_lims[1], alpha = 0.
 ↪3,cmap=cmap)
pc = axs[0,0].pcolormesh(dsbad.profile, dsbad.depth,␣
 ↪dsbad['temperature'],rasterized=True, cmap = cmap,vmin=t_lims[0],␣
 ↪vmax=t_lims[1])
axs[0,0].set_ylim(ylims)
axs[0,0].set_xlim(xlims)
# fig.colorbar(pc, ax=axs[0,0])
axs[0,0].set_ylabel('Depth [m]')
axs[0,0].set_title('Temperature, with profile and density outliers␣
 ↪removed',loc='left')
axs[0,0].add_patch(Rectangle(xy=(profile_lims[0], ylims[1]), width =␣
 ↪profile_lims[1]-profile_lims[0], height = ylims[0]-ylims[1],
                        edgecolor = 'r', facecolor='none', linewidth=1))

pc = axs[1,0].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['temperature_adjusted'],rasterized=True,vmin=t_lims[0], vmax=t_lims[1],␣
 ↪cmap=cmap)
axs[1,0].set_xlim(xlims)
# fig.colorbar(pc, ax=axs[1,0])
axs[1,0].set_ylabel('Depth [m]')
axs[1,0].set_title('Temperature adjusted during thermal lag␣
 ↪correction',loc='left')
axs[1,0].add_patch(Rectangle(xy=(profile_lims[0], ylims[1]), width =␣
 ↪profile_lims[1]-profile_lims[0], height = ylims[0]-ylims[1],
                        edgecolor = 'r', facecolor='none', linewidth=1))

# # Subplots from correction area

xlims = profile_lims
# Subplot from correction area
pc = axs[0,1].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['temperature'],rasterized=True,vmin=t_lims[0], vmax=t_lims[1], alpha = 0.
 ↪3)
pc = axs[0,1].pcolormesh(dsbad.profile, dsbad.depth,␣
 ↪dsbad['temperature'],rasterized=True,vmin=t_lims[0], vmax=t_lims[1],␣
 ↪cmap=cmap)
axs[0,1].set_xlim(xlims)
fig.colorbar(pc, ax=axs[0,1], label = 'Temperature [$^o$C]')
```

```
pc = axs[1,1].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['temperature_adjusted'],rasterized=True,vmin=t_lims[0], vmax=t_lims[1],␣
 ↪cmap=cmap)
axs[1,1].set_xlim(xlims)
fig.colorbar(pc, ax=axs[1,1], label = 'Temperature [$^o$C]')



print('Temperature before (top) and after adjustment (bottom).')
print('Profiles and data points that were ignored for thermal lag correction␣
 ↪are shaded in the top plot.')
print('The profile range used to determine the thermal lag correction is shown␣
 ↪on the right:')
```

```
Temperature before (top) and after adjustment (bottom).
Profiles and data points that were ignored for thermal lag correction are shaded
in the top plot.
The profile range used to determine the thermal lag correction is shown on the
right:
```

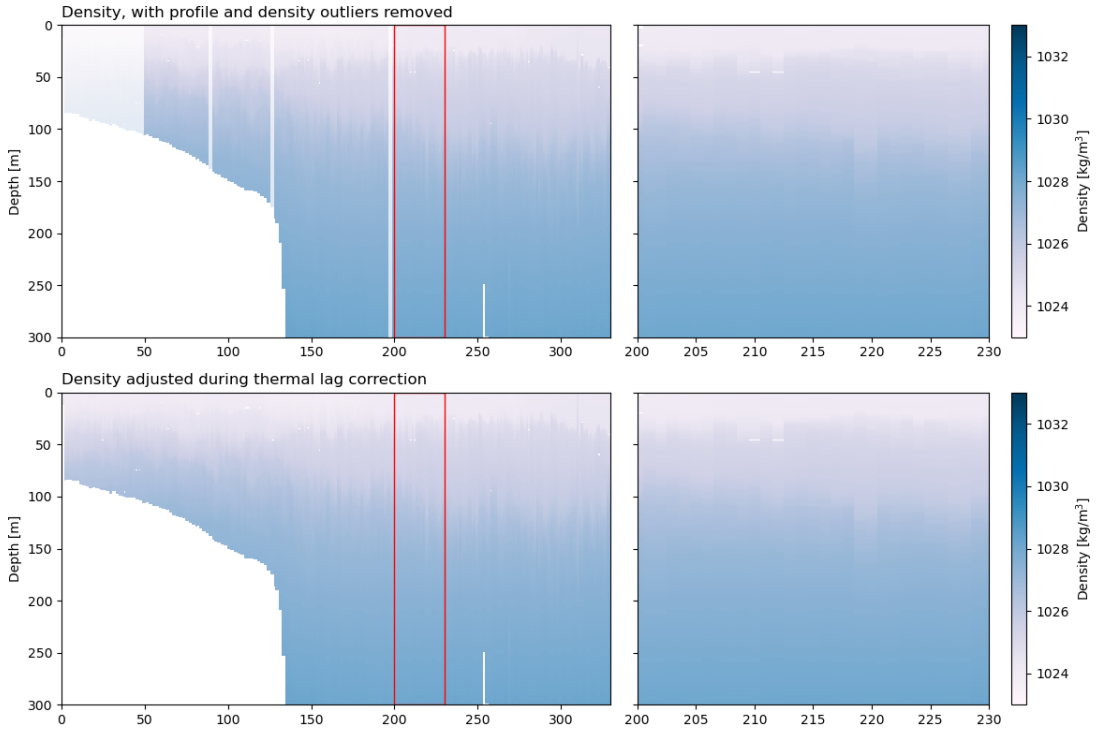


[93]:
```
# RE-PLOTTING WITH THE COND FILTER!
```

```
fig, axs = plt.subplots(2, 2, figsize=(12, 8), sharey=True, width_ratios=[2.
 ↪5,2], squeeze=True)

# Limits
xlims = [0, NUM_PROFILES]
ylims=[300,0]
t_lims = [1023,1033]

# Rectangle showing area used to get alpha and tau
from matplotlib.patches import Rectangle
# print(f'Drawing rectangle centered at: {profile_lims[0], ylims[0]}')
# rect = #, linestyle='dotted'

pc = axs[0,0].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['density'],rasterized=True,vmin=t_lims[0], vmax=t_lims[1], alpha = 0.
 ↪3,cmap='PuBu')
pc = axs[0,0].pcolormesh(dsbad.profile, dsbad.depth,␣
 ↪dsbad['density'],rasterized=True, cmap = 'PuBu',vmin=t_lims[0],␣
 ↪vmax=t_lims[1])
axs[0,0].set_ylim(ylims)
axs[0,0].set_xlim(xlims)
# fig.colorbar(pc, ax=axs[0,0])
axs[0,0].set_ylabel('Depth [m]')
axs[0,0].set_title('Density, with profile and density outliers␣
 ↪removed',loc='left')
axs[0,0].add_patch(Rectangle(xy=(profile_lims[0], ylims[1]), width =␣
 ↪profile_lims[1]-profile_lims[0], height = ylims[0]-ylims[1],
                        edgecolor = 'r', facecolor='none', linewidth=1))

pc = axs[1,0].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['density_adjusted'],rasterized=True,vmin=t_lims[0], vmax=t_lims[1],␣
 ↪cmap='PuBu')
axs[1,0].set_xlim(xlims)
# fig.colorbar(pc, ax=axs[1,0])
axs[1,0].set_ylabel('Depth [m]')
axs[1,0].set_title('Density adjusted during thermal lag correction',loc='left')
axs[1,0].add_patch(Rectangle(xy=(profile_lims[0], ylims[1]), width =␣
 ↪profile_lims[1]-profile_lims[0], height = ylims[0]-ylims[1],
                        edgecolor = 'r', facecolor='none', linewidth=1))


# # # Subplots from correction area

xlims = profile_lims
# Subplot from correction area
```

```
pc = axs[0,1].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['density'],rasterized=True,vmin=t_lims[0], vmax=t_lims[1], alpha = 0.3,␣
 ↪cmap='PuBu')
pc = axs[0,1].pcolormesh(dsbad.profile, dsbad.depth,␣
 ↪dsbad['density'],rasterized=True, cmap='PuBu',vmin=t_lims[0], vmax=t_lims[1])
axs[0,1].set_xlim(xlims)
fig.colorbar(pc, ax=axs[0,1], label = 'Density [kg/m$^3$]')

pc = axs[1,1].pcolormesh(ds.profile, ds.depth,␣
 ↪ds['density_adjusted'],rasterized=True,vmin=t_lims[0], vmax=t_lims[1],␣
 ↪cmap='PuBu')
axs[1,1].set_xlim(xlims)
fig.colorbar(pc, ax=axs[1,1], label = 'Density [kg/m$^3$]')


fig.tight_layout()
print('Density before (top) and after adjustment (middle), with density␣
 ↪difference (bottom).')
print('Profiles and data points that were ignored for thermal lag correction␣
 ↪are shaded in the top plot.')
print('The profile range used to determine the thermal lag correction is shown␣
 ↪on the right:')
```

```
Density before (top) and after adjustment (middle), with density difference
(bottom).
Profiles and data points that were ignored for thermal lag correction are shaded
in the top plot.
The profile range used to determine the thermal lag correction is shown on the
right:
```



3.5.6 2.4.3 Quantifying improvement

To quantify the improvement of the thermal lag correction further, the area of profile pairs was re-calculated and compared to the area between profiles prior to correction. This is calculated from the same quality-controlled data, with suspicious profiles and profile-pairs removed, as during the thermal lag correction, but using **evenly spaced profile pairs** across the deployment.

When examining the corrected data (shown in **orange**) relative to the uncorrected data (shown in **black**), we can see that up-down asymmetry was reduced, as well as a greater reduction compared to using Janzen and Creed constants (shown in **red**). Overall, we can see a large reduction in the area between profiles when using the calculated alpha and tau values. When looking at evenly spaced profiles across the deployment, some pairs have much larger area (e.g. in the middle of the deployment), whereas others remained quite low. When comparing upcasts and downcasts before and after correction in a T-S plot, we also see that they are much better aligned.

[94]:
```
fname=f'{filepath}/{deploy_name}_thermal_lag.nc'
```

[95]:
```
# Set up our constants
fn = 0.5*fs #frequency for Sea-Bird GPCTD
density_cutoff = 1021 #exclude everything less dense than this from the␣
 ↪minimization
num_profs = 100 #number of profiles to include in the subset of data
clean_profs_start = 0#110 #50 #number of profiles to exclude from the start
clean_profs_end = 0#50 #number of profiles to exclude from the end
```

```
dn_stdev = 1 #how many standard deviations from the mean the area between␣
 ↪downcasts can be

alpha = 0.092 ############################values determined above
tau = 10

alpha2 = 0.06 ####Janzen and Creed
tau2=10.1

# Load time series
ts = xr.load_dataset(fname)#ds1.copy(deep=True)
ts = ts.assign_coords(pind=ts.profile_index) #add a profile index coordinate
tot_profs = int(np.nanmax(ts.profile_index.values))
print('Total number of profiles:', tot_profs)

# Overwrite conductivity in our working xarray with the clean, aligned␣
 ↪conductivity field
ts['conductivity'] = ts.conductivityClean

spike_profiles = xr.DataArray([301,307,308], dims="bad_profiles") ##this is␣
 ↪brought from biofouling section
```

```
Total number of profiles: 330
```

[96]:
```
# Determine pairs of profiles for the selected subset of data
ts_sub, profile_bins, profile_bins_all, direction = pgs.profile_pairs(
    ts, clean_profs_start, clean_profs_end, num_profs, bad_profiles
)


# Identify boolean index for application of density cutoff
density_bool = ts_sub.density>=density_cutoff

#Determine the RMSD for the subset of profiles with no corrections applied
area_bad = np.full_like(profile_bins_all, False, dtype=bool)
```

[97]:
```
# Data for the plot below!

# Calculate the area between pairs of profiles for the corrected data
# Alpha and tau values from above

# print(f'Calculating area using alpha {alpha} and tau {tau}')
area_1, p_ind_1 = pgs.TS_diff((alpha*1000, tau),
                              ␣
 ↪fn,density_bool,area_bad,profile_bins,profile_bins_all,ts_sub,
                               ret_err=False)
```

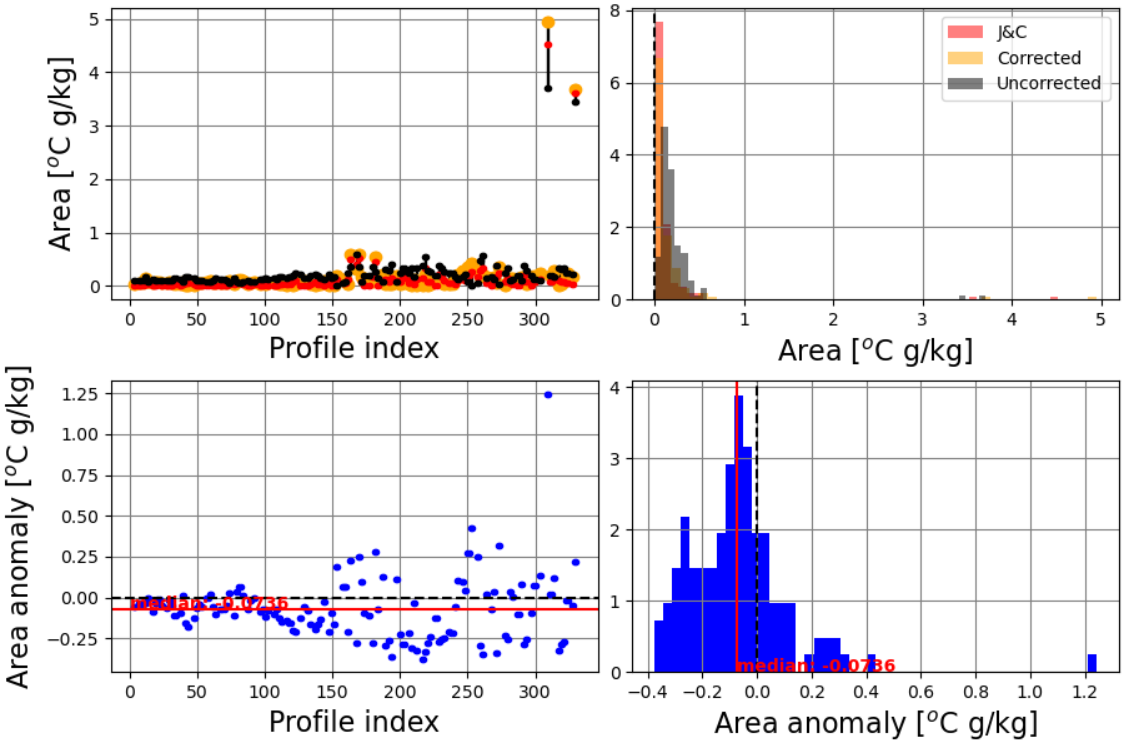
```
# # Janzen and creed values
# print(f'Calculating area using alpha {alpha2} and tau {tau2}')
area_2, p_ind_2 = pgs.TS_diff((alpha2*1000, tau2),
                      ␣
 ↪fn,density_bool,area_bad,profile_bins,profile_bins_all,ts_sub,
                       ret_err=False)

# Original, uncorrected values
# print(f'Calculating area with no corrections')
area_0, p_ind_0 = pgs.TS_diff((0, 0),
                      ␣
 ↪fn,density_bool,area_bad,profile_bins,profile_bins_all,ts_sub,
                       ret_err=False)
```

[98]:
```
fig, ax = plt.subplots(2, 2, figsize=(9, 6),
                           constrained_layout=True)
area_uncorrected = area_0
area_corrected=area_1
area_jandc = area_2

profile_index = p_ind_1

n_bins = 50

ax[0][0].plot([profile_index,profile_index],␣
 ↪[(area_corrected),(area_uncorrected)],
               'k', linestyle='-', linewidth = 1);

ax[0][0].plot(p_ind_1, area_corrected,
           color='orange', marker='o', linestyle='None');
ax[0][0].plot(p_ind_1, area_jandc,
           color='red', marker='.', linestyle='None');
ax[0][0].plot(p_ind_0, area_uncorrected,
           'k.', linestyle='None');
ax[0][0].set_ylabel('Area [$^o$C g/kg]', fontsize=16)
ax[0][0].set_xlabel('Profile index', fontsize=16)
ax[0][0].grid(color='0.5')

# Histogram
ax[0][1].hist(area_jandc, n_bins, density=True, histtype='bar',
              color='red', label='J&C', alpha = 0.5)
ax[0][1].hist(area_corrected, n_bins, density=True, histtype='bar',
              color='orange', label='Corrected', alpha = 0.5)
ax[0][1].hist(area_uncorrected, n_bins, density=True, histtype='bar',
              color='black', label='Uncorrected', alpha = 0.5)
ax[0][1].legend(prop={'size': 10})
ax[0][1].axvline(x = 0, color = 'k', linestyle = '--')
```

```
ax[0][1].set_xlabel('Area [$^o$C g/kg]', fontsize=16)
ax[0][1].grid(color='0.5')

ax[1][0].plot(profile_index,
           (area_corrected)-(area_uncorrected),
           color='blue', marker='.', linestyle='None');
ax[1][0].set_ylabel('Area anomaly [$^o$C g/kg]', fontsize=16)
ax[1][0].set_xlabel('Profile index', fontsize=16)
ax[1][0].grid(color='0.5')
ax[1][0].axhline(y = 0, color = 'k', linestyle = '--')
ax[1][0].axhline(y = np.nanmedian((area_corrected)-(area_uncorrected)), color =␣
 ↪'r', linestyle = '-')
ax[1][0].text(0, np.nanmedian((area_corrected)-(area_uncorrected)),
              f'median: {str(round(np.
 ↪nanmedian((area_corrected)-(area_uncorrected)), 4))}', color = 'r', weight =␣
 ↪'bold')

ax[1][1].hist((area_corrected)-(area_uncorrected), n_bins, density=True,␣
 ↪histtype='bar',
              color='blue')
ax[1][1].set_xlabel('Area anomaly [$^o$C g/kg]', fontsize=16)
ax[1][1].grid(color='0.5')
ax[1][1].axvline(x = 0, color = 'k', linestyle = '--')
ax[1][1].axvline(x = np.nanmedian((area_corrected)-(area_uncorrected)), color =␣
 ↪'r', linestyle = '-')
ax[1][1].text(np.nanmedian((area_corrected)-(area_uncorrected)), 0,
              f'median: {str(round(np.
 ↪nanmedian((area_corrected)-(area_uncorrected)), 4))}', color = 'r', weight =␣
 ↪'bold')

# caption = ('Area between pairs of salinity profiles, calculated in␣
 ↪temperature-salinity space, '
#           'plotted vs. profile index number (left) and as a histogram (right),␣
 ↪for the uncorrected '
#           'salinity field (orange) and the corrected salinity field (black),␣
 ↪and as an anomaly between '
#           'the uncorrected and corrected fields (bottom row, blue).')
# print(caption)

# # plot_correctedarea(p_ind_2, area_0, area_2, n_bins, caption)
print('Total area mean before correction =', np.nanmean(area_0))
print('Total area mean after correction =', np.nanmean(area_1))
print('Total area anomaly mean =', np.nanmean(area_1-area_0))
print('Total area anomaly median =', np.nanmedian(area_1-area_0))
print('*****')
```

```
print(f'Top: The area between {num_profs} profile pairs when uncorrected␣
  ↪(black), using Janzen and Creed alpha ')
print(f'and tau values (red) and when corrected with alpha = {alpha} and tau =␣
  ↪{tau}, shown by profile index and as a histogram.')
print(f'Bottom: the area change between profile pairs when corrected with␣
  ↪calculated alpha and tau. Shown by profile index (left) and histogram␣
  ↪(right):')
```

```
Total area mean before correction = 0.2425031051716113
Total area mean after correction = 0.17343636386298533
Total area anomaly mean = -0.06906674130862596
Total area anomaly median = -0.07357424370162487
*****
Top: The area between 100 profile pairs when uncorrected (black), using Janzen
and Creed alpha
and tau values (red) and when corrected with alpha = 0.092 and tau = 10, shown
by profile index and as a histogram.
Bottom: the area change between profile pairs when corrected with calculated
alpha and tau. Shown by profile index (left) and histogram (right):
```



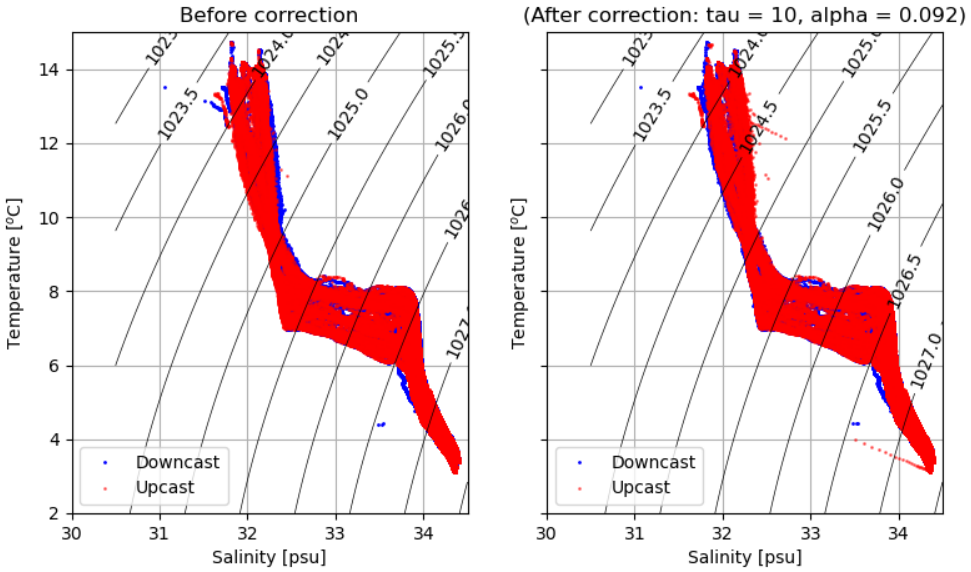
[99]:
```
#Compare the uncorrected and corrected data in T-S space

x_lim=[28, 34.5]
```

```
ts0 = xr.open_dataset(f'{filepath}/{deploy_name}_conductivityClean.nc')

#Create a density grid to contour plot isopycnals
S_range = np.linspace(int(np.min(ds.salinity))-0.5,
                      int(np.max(ds.salinity))+0.5, 1000)
T_range = np.linspace(int(np.min(ds.temperature))-1,
                      int(np.max(ds.temperature))+1, 1000)
S_grid, T_grid = np.meshgrid(S_range, T_range)
density_grid = seawater.eos80.dens0(S_grid, T_grid)
```

[100]:
```
#Compare the uncorrected and corrected data in T-S space

print('Temperature-salinity diagrams for all profiles, '
      'showing the difference between upcasts (red) and downcasts (blue), '
      'for the data without the thermal lag correction applied (left panel) and␣
 ↪'
      'the data with the thermal lag correction applied (right panel):')

x_lim=[30, 34.5]

#Plotting
fig, ax = plt.subplots(1, 2, sharex=True, sharey=True, figsize=(9,5))

ind = np.where(ts0.profile_direction.values== 1)[0]
ax[0].plot(ts0.salinity[ind], ts0.temperature[ind], 'b.', markersize=2,␣
 ↪rasterized=True, label = 'Downcast')

ind = np.where(ts0.profile_direction.values == -1)[0]
ax[0].plot(ts0.salinity[ind], ts0.temperature[ind], 'r.', markersize=2, alpha =␣
 ↪0.5, rasterized=True, label = 'Upcast')

CS = ax[0].contour(S_range, T_range, density_grid,
                   np.arange(1021,np.round(np.max(density_grid)),0.5),
                   colors='k', linewidths=0.5);
ax[0].clabel(CS, CS.levels, inline=True, fontsize=10)
ax[0].set_ylabel('Temperature [$^o$C]')
ax[0].set_xlabel('Salinity [psu]')
ax[0].set_title('Before correction')
ax[0].set_xlim(x_lim)
ax[0].grid()

########after correction is the thermal lag file
ind = np.where(ts_sub.profile_direction.values== 1)[0]
ax[1].plot(ts_sub.salinity_corrected[ind], ts_sub.temperature_adjusted[ind], 'b.
 ↪', markersize=2, rasterized=True, label = 'Downcast')
ind = np.where(ts_sub.profile_direction.values==-1)[0]
```

```
ax[1].plot(ts_sub.salinity_corrected[ind], ts_sub.temperature_adjusted[ind], 'r.
 ↪', markersize=2, alpha = 0.5, rasterized=True, label = 'Upcast')

CS = ax[1].contour(S_range, T_range, density_grid,
                   np.arange(1021,np.round(np.max(density_grid)),0.5),
                   colors='k', linewidths=0.5);
ax[1].clabel(CS, CS.levels, inline=True, fontsize=10)
ax[1].set_ylabel('Temperature [$^o$C]')
ax[1].set_xlabel('Salinity [psu]')
ax[1].set_title(f'(After correction: tau = 10, alpha = {alpha})' )
ax[1].grid()

ax[0].legend(prop={'size': 10});
ax[1].legend(prop={'size': 10});
```

Temperature-salinity diagrams for all profiles, showing the difference between upcasts (red) and downcasts (blue), for the data without the thermal lag correction applied (left panel) and the data with the thermal lag correction applied (right panel):



4 3.0 Summary of corrections applied to delayed mode data for this mission

Identification of anomalous conductivity values: * Profiles with spikes in the salinity data from biofouling were set to NaN. * Anomalous conductivity values at the surface caused by air bubbles in the cell were set to NaN.

Sensor alignment correction: * No sensor alignment correction was applied.

Identification of questionable salinity profiles: * Numerous salinity profiles were flagged as 'bad' and their values set to NaN for the thermal lag correction. The range of profiles examined was not limited, and these profiles were not removed from the final corrected salinity dataset.

Thermal lag correction: * The directly determined values for the thermal lag correction produced an improvement that was larger than the recommended values from Janzen and Creed (2011). * The correction overall significantly reduced the root-mean squared difference for the area between between pairs of profiles. * The final thermal lag correction was applied using the calculated values of:

[101]:
```
print(f'alpha = {alpha} and tau = {tau}')
```

```
alpha = 0.092 and tau = 10
```

[102]:
```
# Set up and our final datasets
ts_final = ts
ts0 = pgs.get_timeseries(filepath, deploy_name) ##we are changing the␣
 ↪spike_clean v
```

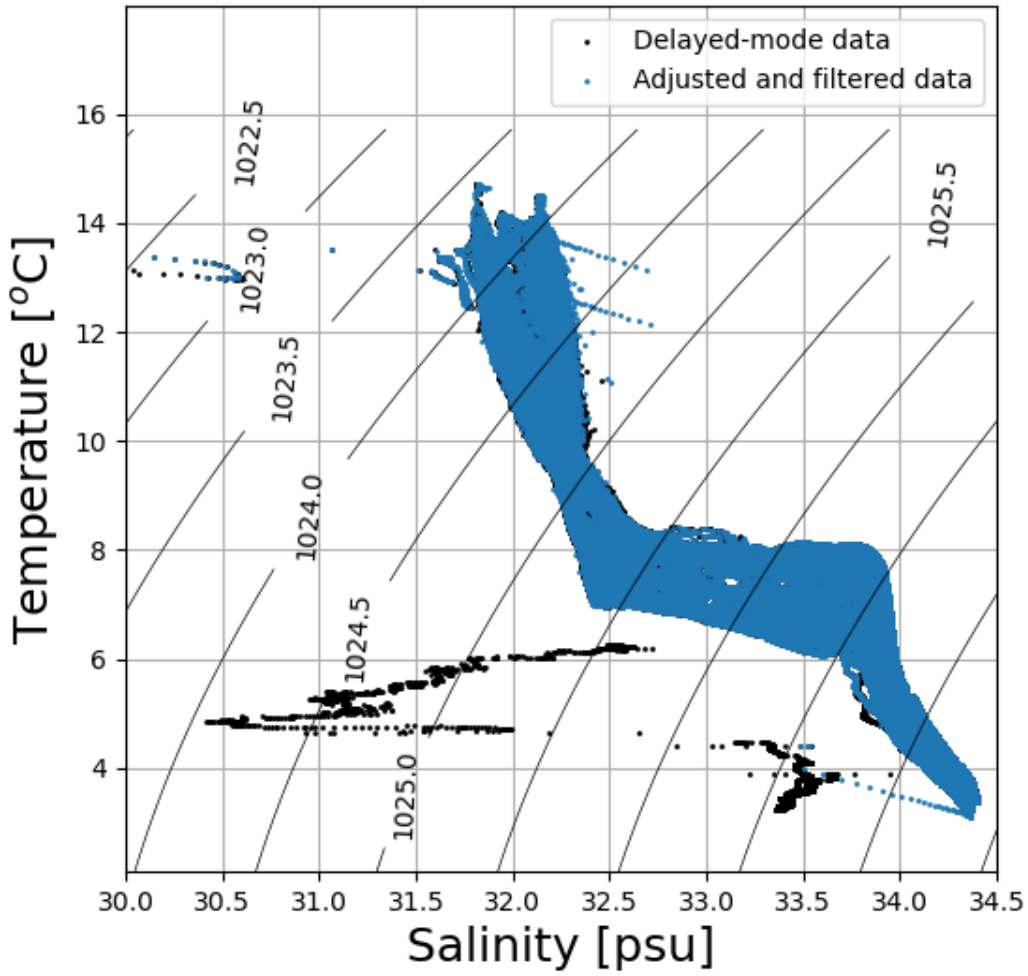
[103]:
```
#update processing attributes
ts0.attrs['processing_details'] = 'Processing details are located on the␣
 ↪C-PROOF website for this mission under the reports tab.'
ts0.attrs['processing_tech'] = 'Lauryn Talbot; ltalbot@uvic.ca'
ts0.attrs['citation'] = '"Klymak, J., & Ross, T. (2025). C-PROOF Underwater␣
 ↪Glider Deployment Datasets [Data set]. Canadian-Pacific Robotic Ocean␣
 ↪Observing Facility.doi:10.82534/44DS-K310"'
ts0.attrs['references'] = 'https://doi.org/10.82534/44DS-K310'

# Uncorrected conductivity
ts0['conductivity'].attrs['comment'] = 'uncorrected conductivity'

# Adjusted (aka cleaned) conductivity
ts0['conductivity_adjusted'] = ts_final.conductivity.copy()
ts0.conductivity.values = ts_final.conductivityClean.values
ts0['conductivity_adjusted'].attrs['comment'] = 'adjusted conductivity'
ts0['conductivity_adjusted'].attrs['processing_report'] = processing_report
ts0['conductivity_adjusted'].attrs['processing_date'] = processing_date
ts0['conductivity_adjusted'].attrs['processing_date'] = processing_protocol

# Uncorrected temperature
ts0['temperature'].attrs['comment'] = 'uncorrected temperature [degC]'

# Adjusted temperature
ts0['temperature_adjusted'] = ts_final.temperature_adjusted.copy()
ts0['temperature_adjusted'].attrs['comment'] = 'adjusted temperature [degC]'
ts0['temperature_adjusted'].attrs['processing_report'] = processing_report
ts0['temperature_adjusted'].attrs['processing_date'] = processing_date
```

```
ts0['temperature_adjusted'].attrs['processing_date'] = processing_protocol

# Uncorrected salinity
ts0['salinity'].attrs['comment'] = 'uncorrected salinity [psu]'

# Corrected salinity
ts0['salinity_adjusted'] = ts_final.salinity_corrected.copy()
ts0['salinity_adjusted'].attrs['comment'] = 'adjusted salinity [psu]'
ts0['salinity_adjusted'].attrs['method'] = ' '
ts0['salinity_adjusted'].attrs['processing_report'] = processing_report
ts0['salinity_adjusted'].attrs['processing_date'] = processing_date
ts0['salinity_adjusted'].attrs['processing_protocol'] = processing_protocol

# Unadjusted density
ts0['density'].attrs['comment'] = 'unadjusted density'

# Adjusted density
ts0['density_adjusted'] = ts_final.density_adjusted.copy()
ts0['density_adjusted'].attrs['comment'] = 'density from adjusted salinity␣
 ↪[psu] and temperature [degC]'
ts0['density_adjusted'].attrs['method'] = ' '
```

[104]:
```
# Visualize the final data
fig, ax = plt.subplots(1, 1, figsize=(6, 6))
X_LIM = [30,34.5]
# T-S diagram for fully corrected data
ax0 = ax
ax0.plot(ts0.salinity,ts0.temperature,'k.',markersize=2, label = "Delayed-mode␣
 ↪data")
ax0.plot(ts0.salinity_adjusted,ts0.temperature_adjusted,'.',markersize=2, label␣
 ↪= "Adjusted and filtered data")

#Create a density grid to contour plot isopycnals
S_range = np.linspace(np.nanmin(ts0.salinity_adjusted)-0.5,
                      np.nanmax(ts0.salinity_adjusted)+0.5, 1000)
T_range = np.linspace(np.nanmin(ts0.temperature_adjusted)-1,
                      np.nanmax(ts0.temperature_adjusted)+1, 1000)
S_grid, T_grid = np.meshgrid(S_range, T_range)
density_grid = seawater.eos80.dens0(S_grid, T_grid)

CS = ax0.contour(S_range, T_range, density_grid,
                 np.arange(1014,
                           np.round(np.max(density_grid)),0.5),
                 colors='k', linewidths=0.5);
ax0.clabel(CS, CS.levels, inline=True, fontsize=10)
ax0.set_xlabel('Salinity [psu]', fontsize=18)
ax0.set_ylabel('Temperature [$^o$C]', fontsize=18)
```

```
ax0.set_xlim(X_LIM)
ax0.grid()
ax0.legend()

print('The corrected temperature and salinity fields '
      'shown in a T-S diagram with density contours:')
```

```
The corrected temperature and salinity fields shown in a T-S diagram with
density contours:
```



[105]:
```
# Save our final datasets
ts0.to_netcdf(f'{filepath}{deploy_name}_CTDadjusted.nc')
```

```
# print(f'Corrected data saved to file: {filepath}/{glider_name}/{deploy_name}/
  ↪{deploy_name}_adjusted.nc')
```

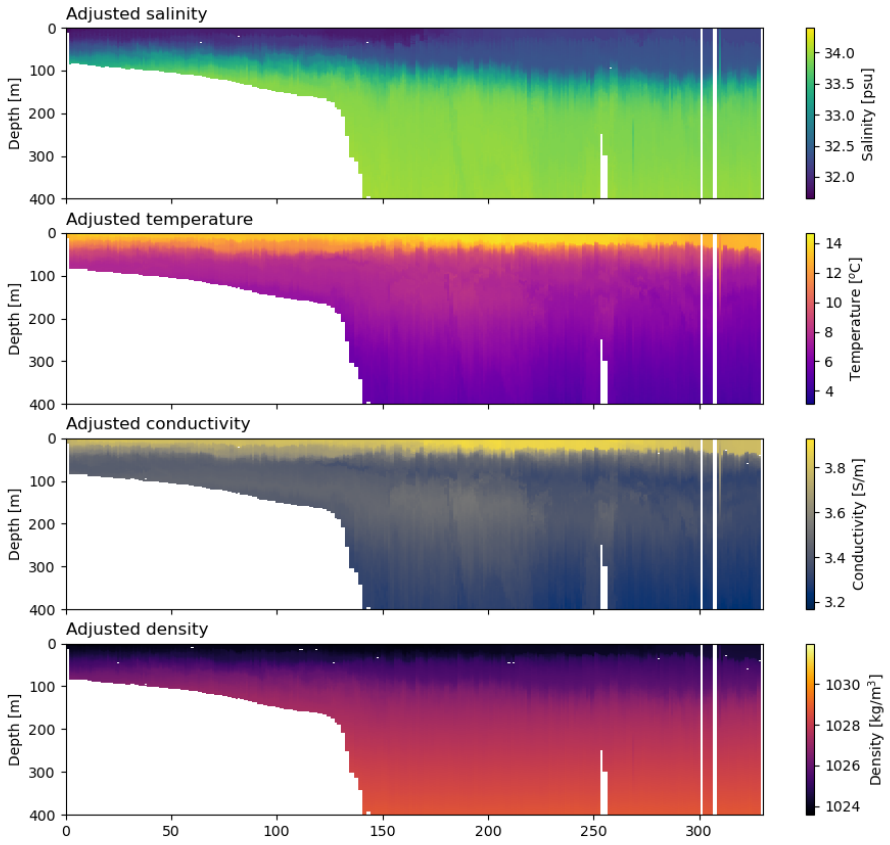
[106]:
```
# ds.close()
import pyglider.ncprocess as ncprocess
ncprocess.make_gridfiles(f'{filepath}{deploy_name}_CTDadjusted.nc',
                         f'{filepath}', deployfile, fnamesuffix='_CTDadjusted')
```

[106]:
```
'deployments/dfo-walle652/dfo-walle652-20200616//dfo-
walle652-20200616_grid_CTDadjusted.nc'
```

[107]:
```
# Open the grid file
ds=xr.open_dataset(f'{filepath}{deploy_name}_grid_CTDadjusted.nc')
# list(ds.keys())

# RE-PLOTTING WITH THE COND FILTER!
fig, axs = plt.subplots(4, 1, figsize=(11, 10), sharey=True, sharex=True)

xlims = [0, NUM_PROFILES]
ylims=[400,0]

pc = axs[0 ].pcolormesh(ds.profile, ds.depth,␣
  ↪ds['salinity_adjusted'],rasterized=True)
axs[0].set_ylim(ylims)
axs[0].set_xlim(xlims)
fig.colorbar(pc, ax=axs[0], label = 'Salinity [psu]')
axs[0].set_title('Adjusted salinity',loc='left')

pc = axs[1].pcolormesh(ds.profile, ds.depth,␣
  ↪ds['temperature_adjusted'],rasterized=True,cmap='plasma')
fig.colorbar(pc, ax=axs[1], label = 'Temperature [$^o$C]')
axs[1].set_title('Adjusted temperature',loc='left')

pc = axs[2].pcolormesh(ds.profile, ds.depth,␣
  ↪ds['conductivity_adjusted'],rasterized=True,cmap='cividis')
fig.colorbar(pc, ax=axs[2], label = 'Conductivity [S/m]')
axs[2].set_title('Adjusted conductivity',loc='left')

# pc = axs[3].pcolormesh(ds.profile, ds.depth,␣
  ↪ds['oxygen_concentration'],rasterized=True,cmap='inferno')
# fig.colorbar(pc, ax=axs[3])
# axs[3].set_title('Oxygen Concentration',loc='left')

pc = axs[3].pcolormesh(ds.profile, ds.depth,␣
  ↪ds['density_adjusted'],rasterized=True,cmap='inferno')
fig.colorbar(pc, ax=axs[3], label = 'Density [kg/m$^3$]')
axs[3].set_title('Adjusted density',loc='left')
```

```
axs[0].set_ylabel('Depth [m]')
axs[1].set_ylabel('Depth [m]')
axs[2].set_ylabel('Depth [m]')
axs[3].set_ylabel('Depth [m]')

print('The corrected salinity and temperature, shown with filtered conductivity␣
 ↪and adjusted density:')
```

```
The corrected salinity and temperature, shown with filtered conductivity and
adjusted density:
```



[108]:
```
display(Markdown('./docs/CTD_References.md'))
```

5 References

1. Ferrari, R., and Rudnick, D. L. Thermohaline variability in the upper ocean, J. Geophys. Res., 105(C7), 16857-16883, 2000.
2. Garau, B., Ruiz, S., Zhang, W. G., Pascual, A., Heslop, E., Kerfoot, J., & Tintoré, J. Thermal Lag Correction on Slocum CTD Glider Data, J. Atmos. Oceanic Technol., 28(9), 1065-1071, 2011.
3. Janzen, C. D., and Creed, E. L. Physical oceanographic data from Seaglider trials in stratified coastal waters using a new pumped payload CTD, OCEANS'11 MTS/IEEE KONA, Waikoloa, HI, USA, 1-7, 2011.
4. Morison, J., Andersen, R., Larson, N., D'Asaro, E., & Boyd, T. The correction for thermal-lag effects in Sea-Bird CTD data, J. Atmos. Oceanic Technol., 11, 1151-1164, 1994.
5. Sea-Bird Seasoft V2:SBE Data Processing - CTD Data Processing & Plotting Software for Windows, Sea-Bird Scientific, software manual revision 7.26.8, 2017.
6. Sea-Bird User Manual - GPCTD Glider Payload CTD (optional DO) - Conductivity, Temperature, and Pressure (optional DO) Sensor with RS-232 Interface, Sea-Bird Scientific, manual version 008, 2021.

[ ]: