2013 Saqardleq ADCP processing

Clark Richards

January 31, 2018

Introduction

This document describes the processing of the 2013 Saqardleq ADCP data – that is the shipboard ADCP data collected from the vessel-mounted ADCP on Ove's boat (not the JetYak). The data were originally processed by Dan Torres, I believe using the UHDAS ADCP processing toolbox, though there were some inconsistent results especially in the vicinity of the glacier face. I suspect that the issue with the UHDAS processing is that it uses a "water track" later for correcting the measured velocities for the vessel motion, and in potentially strong currents and uneven terrain this didn't work as expected (this is pure speculation on my part – I have no reason to distrust Dan's processing except that I get something different).

The processing for this data is done in the R language, using functions available in the `oce` package (https://cran.rstudio.com/web/packages/oce/index.html (https://cran.rstudio.com/web/packages/oce/index.html)) for reading and processing raw Teledyne RDI ADCP files.

Processing details

The raw data files were read using the `read.adp.rdi()` function, and the resulting objects (one for each file) saved to a structure in a binary R format for later reading and processing. The code to do this is as follows:

```
library(oce)
datadir <- '/data/archive/fjord/2013/sadcp/raw'
files <- dir(datadir, pattern='*.ENS', full.names = TRUE)[-1] #something wrong with the first file
adp <- lapply(as.list(files), read.oce)
section <- unlist(lapply(adp, function(x) substr(x[['filename']], 40, 42)))
names(adp) <- paste0('s', section)
save(file='sadcp_2013.rda', adp)
```

The raw files that are loaded for processing are the `.ENS` files, which contain the raw "single-ping" Doppler and backscatter data, but which has been merged with the navigation data (in NMEA form) from the external GPS.

Note that the ADCP was configured with a heading bias in the instrument setup of 45 degrees. The reason for this is to account for the fact that the ADCP was set up over the side of the boat with the y axis (beam 3) pointed 45 degrees to the starboard side of the bow of the boat. This is a common setup for ship mounted ADCPs, partly because it ensures that all 4 beams will see the ship motion equally (rather than mapping the ship motion mostly into the along-keel beams). However, the alignment of 45 degrees is needed only if the compass being used to determine orientation is actually aligned with the

ship's keel. In this case, the heading information was being provided by the internal ADCP compass, which means that the 45 degree alignment is irrelevant to the processing and is thus removed. Note: if the ADCP data are run through the RDI software, then there is a later step which corrects for the heading bias of 45 degrees by applying a "heading alignment" of -45, thus undoing the heading correction. However, the raw data as stored in the ADCP when there is a "heading bias" present is not the true heading, but the one modified by the setting).

Magnetic declination

The magnetic compass heading also needs to be corrected for magnetic declination for the area and the date.

```
library(oce)
```

```
## Loading required package: methods
```

```
## Loading required package: gsw
```

```
## Loading required package: testthat
```

```
load('sadcp_2013.rda')
datadir <- '/data/archive/fjord/2013/sadcp/raw'
files <- dir(datadir, pattern='*.ENS', full.names = TRUE)[-1] #something wrong
with the first file
lon <- mean(unlist(lapply(adp, function(x) x[['firstLongitude']])))
lat <- mean(unlist(lapply(adp, function(x) x[['firstLatitude']])))
dec <- magneticField(lon, lat, as.POSIXct('2013-07-25 00:00:00', tz='UTC'))$dec
lination
```

The average declination for the 2013 period is -32.5.

Removal of ship velocity

As the ADCP is attached to a moving vessel, in order to infer water velocities the motion due to the ship must be removed from the raw Doppler data. The ADCP (300 kHz Workhorse) was configured in "bottom-track mode", meaning that it uses a special series of pings in each sample to:

1. determine the range to the bottom for each beam, and
2. determine the apparent velocity of the bottom, using the same Doppler principles used for measuring the water velocity

Once the apparent "bottom velocity" is measured, it can be subtracted from the raw Doppler data to remove the velocity of the vessel and leave the water velocity as a residual. In R/oce, this is accomplished using the `subtractBottomVelocity()` function.

In single-ping mode, the bottom track data can often be noisy (like the raw water column data), and requires some processing to ensure spurious signals do not contaminate the final water velocity data. Typical processing often involves despiking both the bottom range and velocities, as well as interpolation of missing values.

Coordinate transformations

The raw ADCP data are collected in “beam” coordinates, meaning that the velocities for each of the 4 beams are measured as components along the axis of the beam. The geometry of the instrument allows a transformation to “instrument” coordinates (relative to the axes of the instrument/ship), and then to “earth” coordinates, often referred to as East-North-Up (ENU). Coordinate transformations are handled in R/oce using the `toEnu()` function (which handles all intermediate transformations in going from beam to ENU).

Processing of individual ADCP sections

In total during the 2013 field campaign, there were 10 “sections” completed (i.e. 10 individual datasets). The datasets are identified by number, and the files loaded are:

```
substr(files, 36, 53)
```

```
##  [1] "ADCP065_000000.ENS" "ADCP066_000000.ENS" "ADCP067_000000.ENS"
##  [4] "ADCP068_000000.ENS" "ADCP069_000000.ENS" "ADCP070_000000.ENS"
##  [7] "ADCP073_000000.ENS" "ADCP074_000000.ENS" "ADCP075_000000.ENS"
## [10] "ADCP076_000000.ENS"
```

The range of times spanned by each of the sections are:

```
startTimes <- numberAsPOSIXct(unlist(lapply(adp, function(x) head(x[['time']],
1))))
endTimes <- numberAsPOSIXct(unlist(lapply(adp, function(x) tail(x[['time']], 1)
)))
df <- data.frame(section=substr(files, 36, 53), start=startTimes, end=endTimes)
kable(df)
```

section	start	end
ADCP065_000000.ENS	2013-07-25 12:17:34	2013-07-25 12:22:33
ADCP066_000000.ENS	2013-07-25 12:26:05	2013-07-25 14:47:55
ADCP067_000000.ENS	2013-07-25 16:40:59	2013-07-25 16:55:00
ADCP068_000000.ENS	2013-07-26 14:05:23	2013-07-26 14:31:07
ADCP069_000000.ENS	2013-07-26 14:37:58	2013-07-26 16:15:26

section	start	end
ADCP070_000000.ENS	2013-07-26 16:47:27	2013-07-26 17:48:26
ADCP073_000000.ENS	2013-07-26 18:03:10	2013-07-26 18:16:15
ADCP074_000000.ENS	2013-07-27 13:28:04	2013-07-27 14:18:20
ADCP075_000000.ENS	2013-07-27 14:48:32	2013-07-27 15:14:47
ADCP076_000000.ENS	2013-07-27 15:26:40	2013-07-27 17:26:02

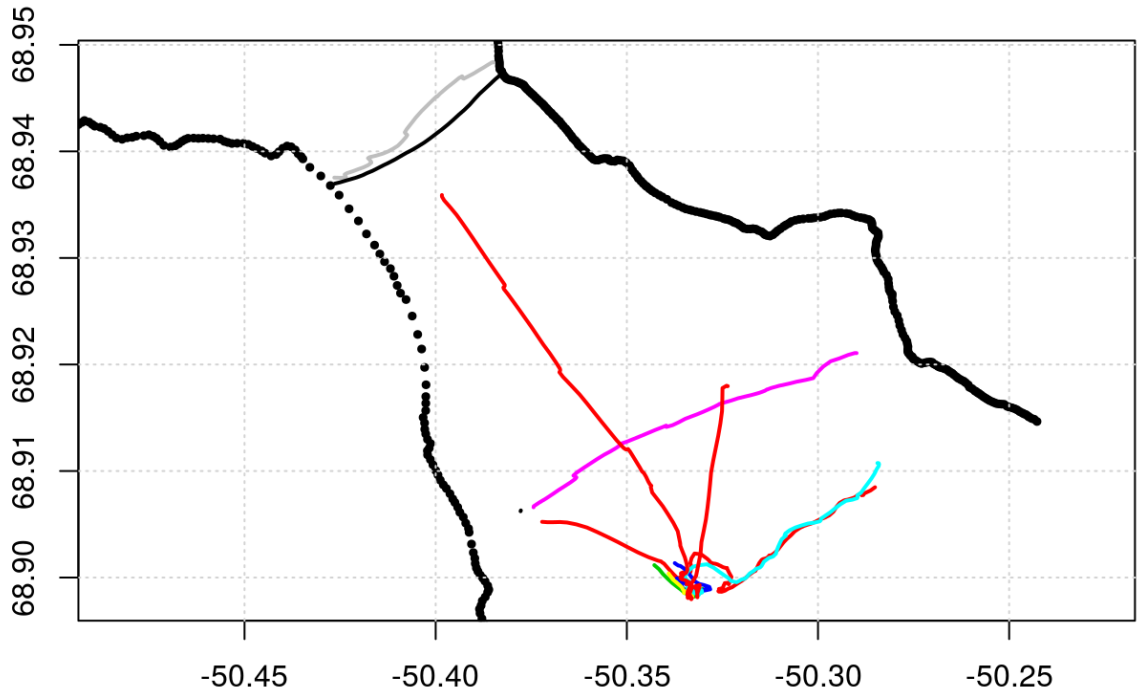
For each section, the first minute of data is discarded (often the data is of low quality before gps fixes are made, etc). Then the bottom track range and velocities are despiked, the bins located below the bottom (and within 15% of the total range) are discarded, and the ship velocity is removed. Then, the data is transformed to ENU coordinates, corrected for magnetic declination, and ensemble averaged to ensembles of 30 pings.

```
enu <- enua <- list()
for (ii in seq_along(adp)) {
    d <- subset(adp[[ii]], time > startTimes[ii] + 120)
    ## despike bottom velocities
    for (i in 1:4) {
        II <- d[['bv']][,i] < -10
        d[['bv']][II,i] <- NA
        d[['bv']][,i] <- despike(d[['bv']][,i], n=1, k=21)
        d[['bv']][,i] <- approx(d[['time']], d[['bv']][,i], d[['time']])$y
    }
    ## despike bottom range
    for (i in 1:4) {
        d[['br']][,i] <- despike(d[['br']][,i])
        d[['br']][,i] <- approx(d[['time']], d[['br']][,i], d[['time']])$y
    }
    ## trim sub-bottom bins
    mask <- array(1, dim=dim(d[['v']]))
    for (i in 1:4) {
        for (j in seq_along(d[['time']])) {
            II <- d[['distance']] > 0.85*(d[['br']][j,i])
            mask[j, II ,i] <- NA
        }
    }
    d[['v']] <- d[['v']]*mask
    d[['a']] <- d[['a', 'numeric']]*mask
    dd <- subtractBottomVelocity(d)
    enu[[ii]] <- toEnu(dd, declination=dec)
    enua[[ii]] <- adpEnsembleAverage(enu[[ii]], n=30)
}
names(enu) <- names(adp)
names(enua) <- names(adp)

save(file='sadcp_processed_2013.rda', enu, enua)
```

Processed data summary

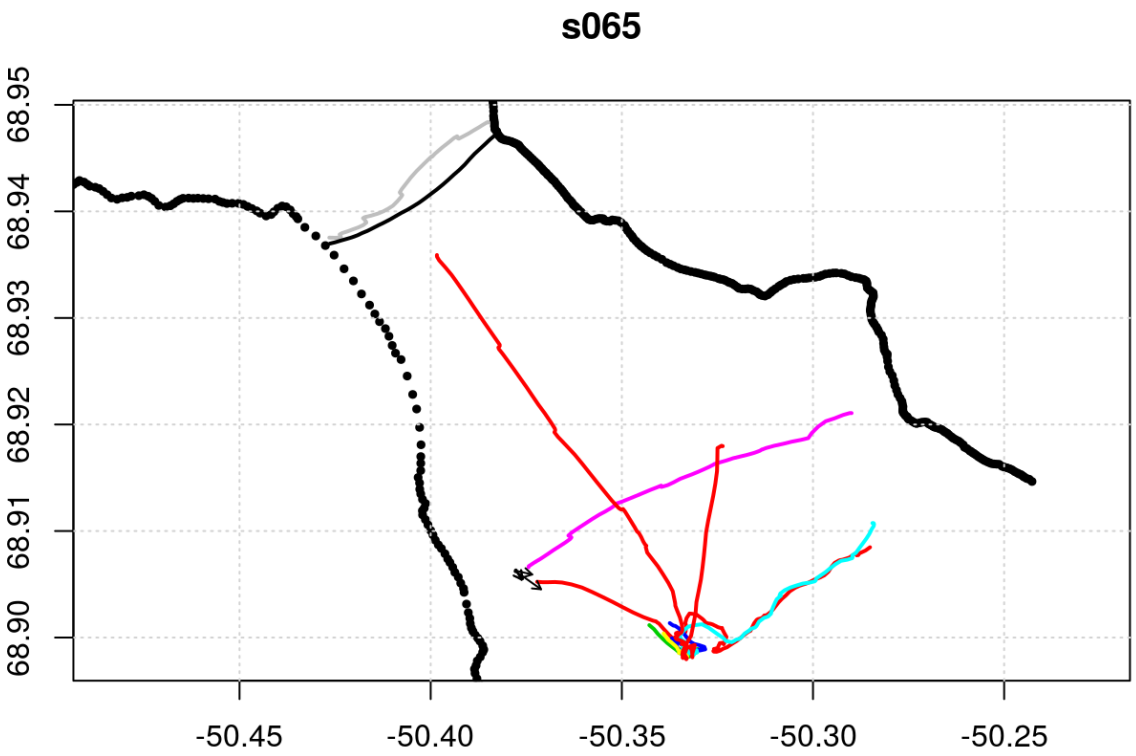
The various sections can be summarized on a map:

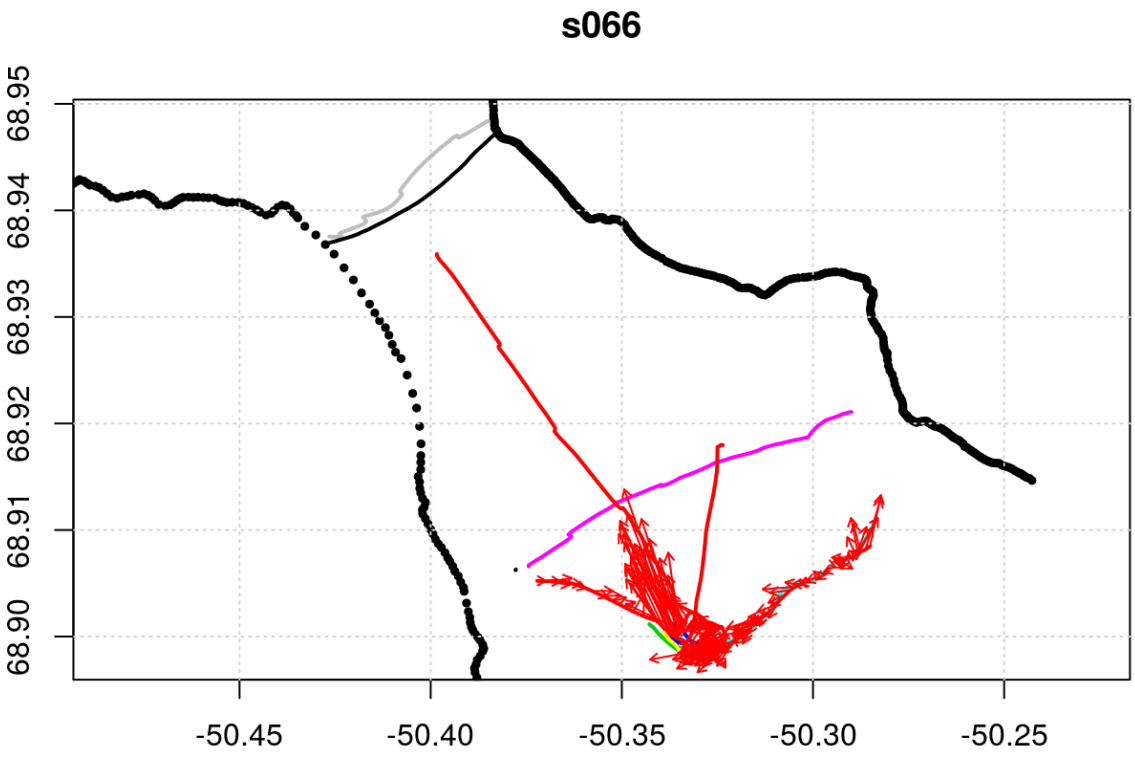
```
load('02_coast.rda')
lonlim <- range(lapply(enua, function(x) range(x[['firstLongitude']], na.rm=TRU
E)))
latlim <- range(lapply(enua, function(x) range(x[['firstLatitude']], na.rm=TRUE
)))
plot(clon, clat, asp=1/cos(68.95*pi/180), pch=19, cex=0.5,
     xlim=lonlim+c(-0.015, 0.015), ylim=latlim,
     xlab='', ylab='')
grid()

for (i in seq_along(enua)) {
    lines(enua[[i]][['firstLongitude']], enua[[i]][['firstLatitude']], col=i, l
wd=2)
}
```

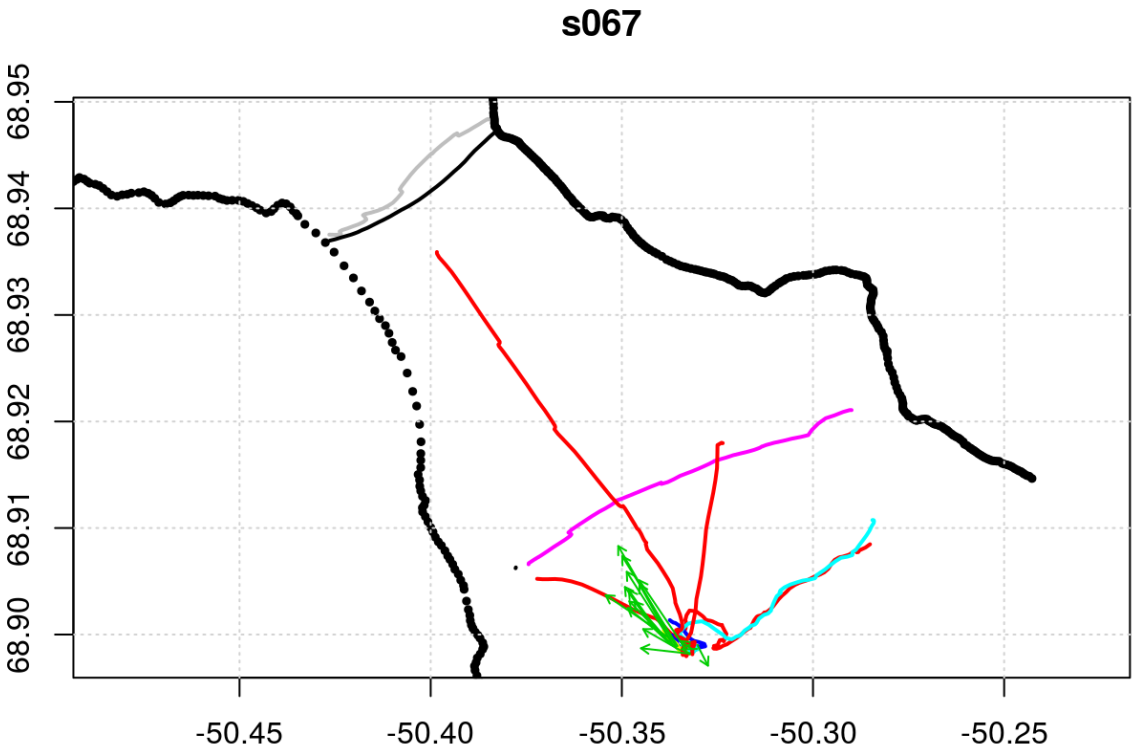


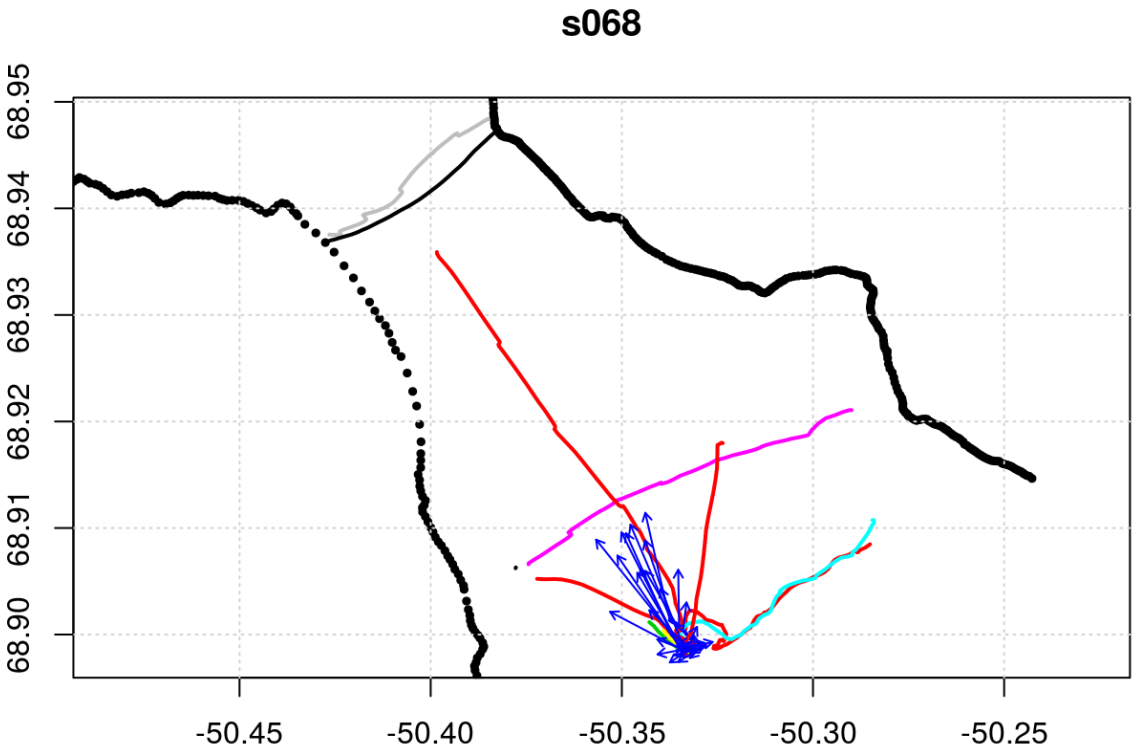
A summary of the near-surface velocity field can be plotted by averaging currents in the upper 50m, and plotting as arrows on a map.

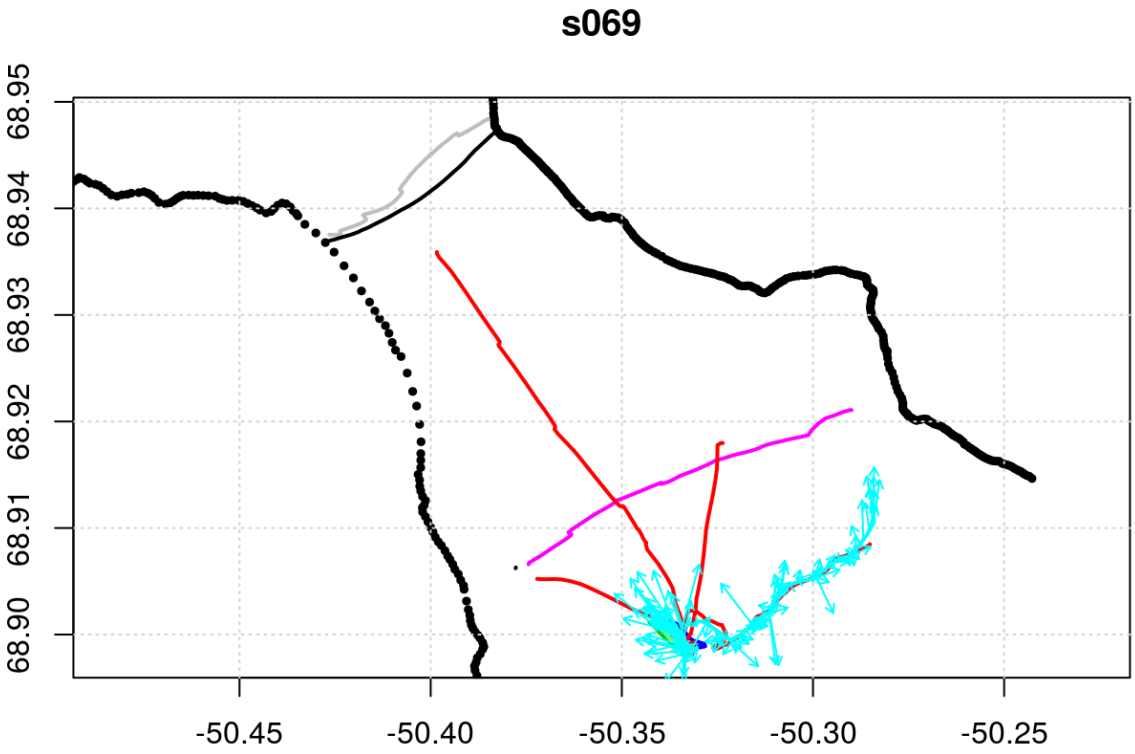
```
for (i in seq_along(enua)) {
    plot(clon, clat, asp=1/cos(68.95*pi/180), pch=19, cex=0.5,
         xlim=lonlim+c(-0.015, 0.015), ylim=latlim,
         xlab='', ylab='')
    title(names(enua)[i])
    grid()

    for (j in seq_along(enua)) {
        lines(enua[[j]][['firstLongitude']], enua[[j]][['firstLatitude']], col=
j, lwd=2)
    }

    II <- enua[[i]][['distance']] <= 50
    u <- apply(enua[[i]][['v']][,II,1], 1, mean, na.rm=TRUE)
    v <- apply(enua[[i]][['v']][,II,2], 1, mean, na.rm=TRUE)
    drawDirectionField(enua[[i]][['firstLongitude']], enua[[i]][['firstLatitude
']],
                       u, v, scalex=0.1, type=2, add=TRUE, col=i)
}
```

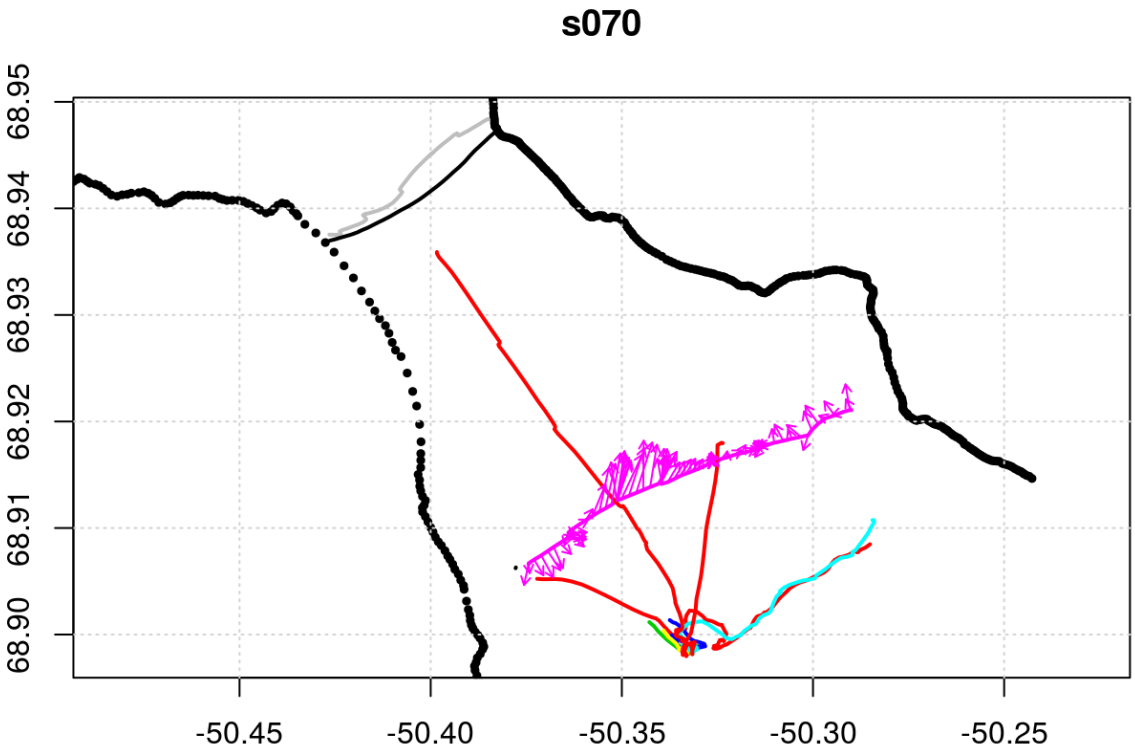


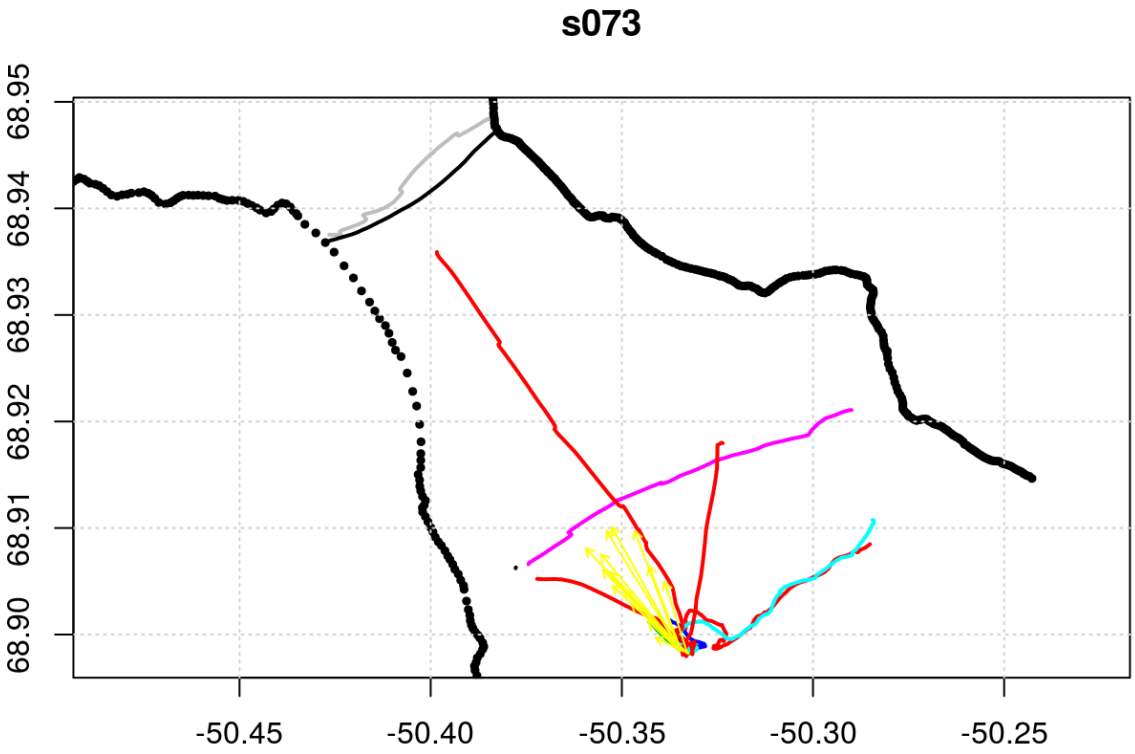

s066
68.95
68.94
68.93
68.92
68.91
68.90
-50.45
-50.40
-50.35
-50.30
-50.25

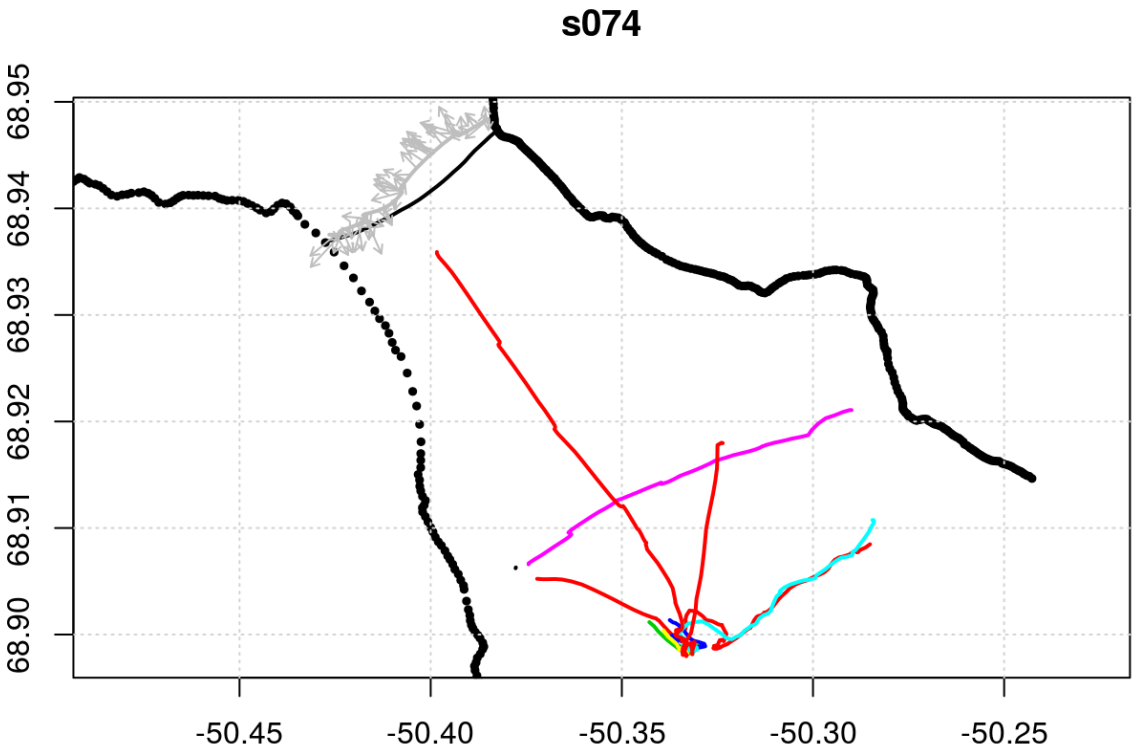

s067
68.95
68.94
68.93
68.92
68.91
68.90
-50.45
-50.40
-50.35
-50.30
-50.25

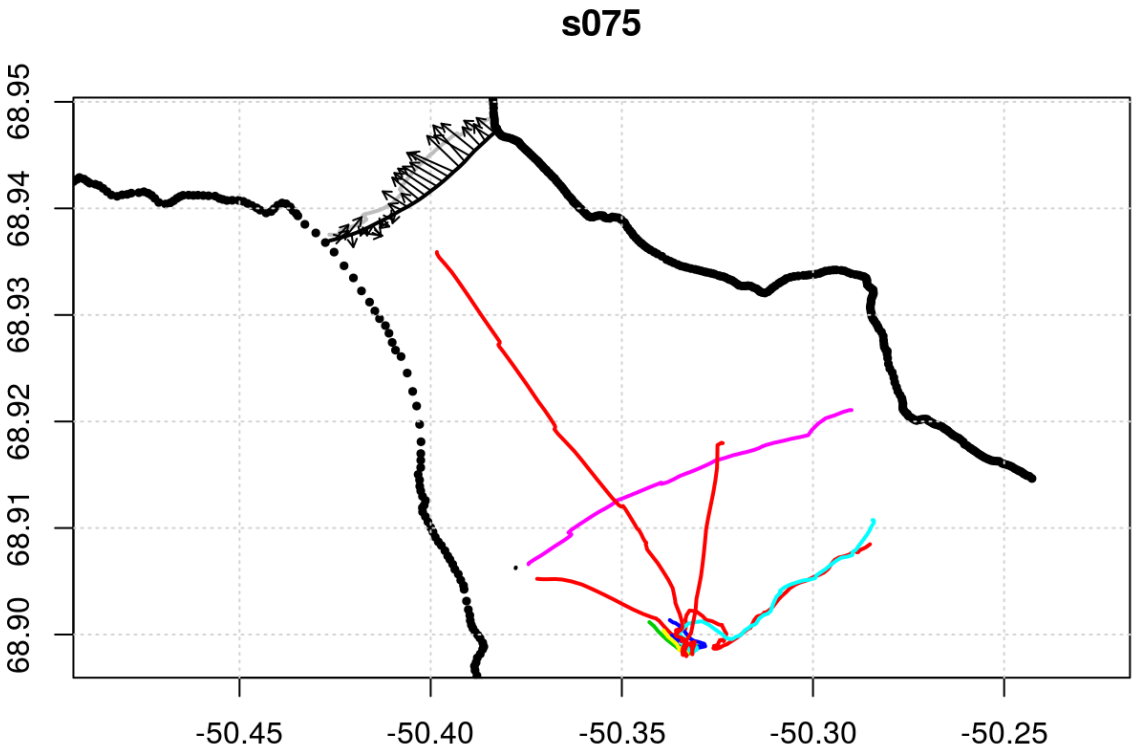

s068
68.95
68.94
68.93
68.92
68.91
68.90
-50.45
-50.40
-50.35
-50.30
-50.25

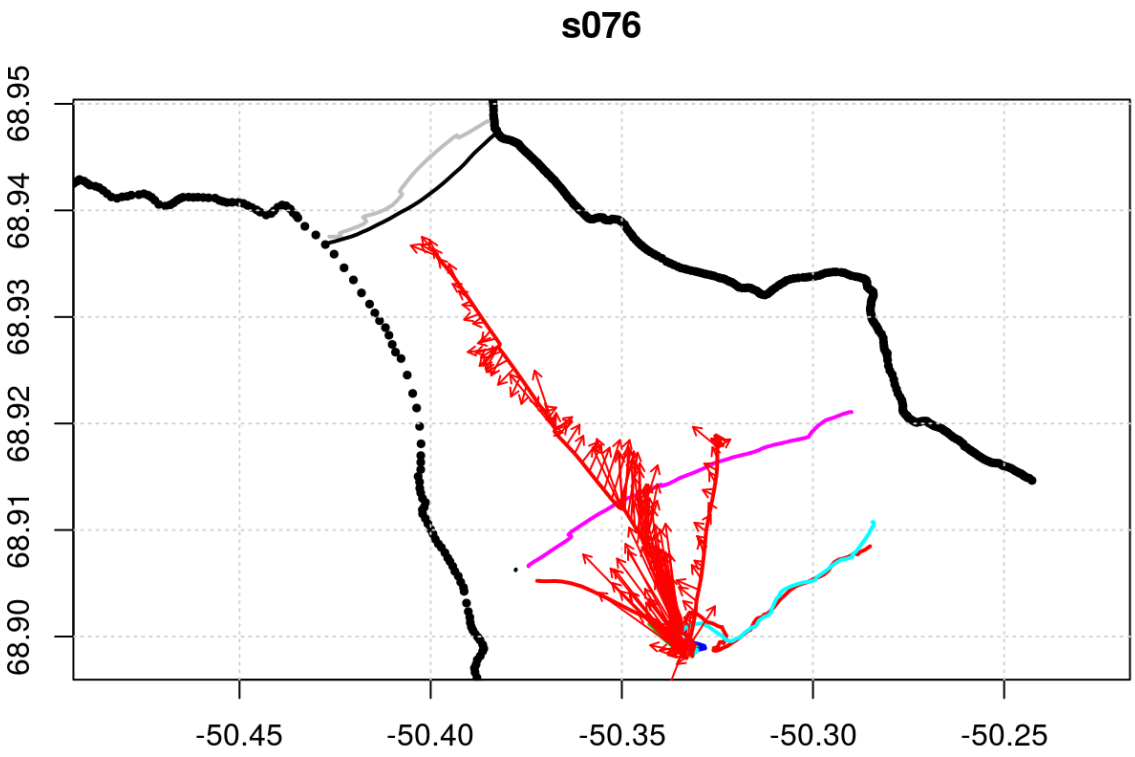

s069
68.95
68.94
68.93
68.92
68.91
68.90
-50.45
-50.40
-50.35
-50.30
-50.25


s070
68.95
68.94
68.93
68.92
68.91
68.90
-50.45
-50.40
-50.35
-50.30
-50.25


s073
68.95
68.94
68.93
68.92
68.91
68.90
-50.45
-50.40
-50.35
-50.30
-50.25

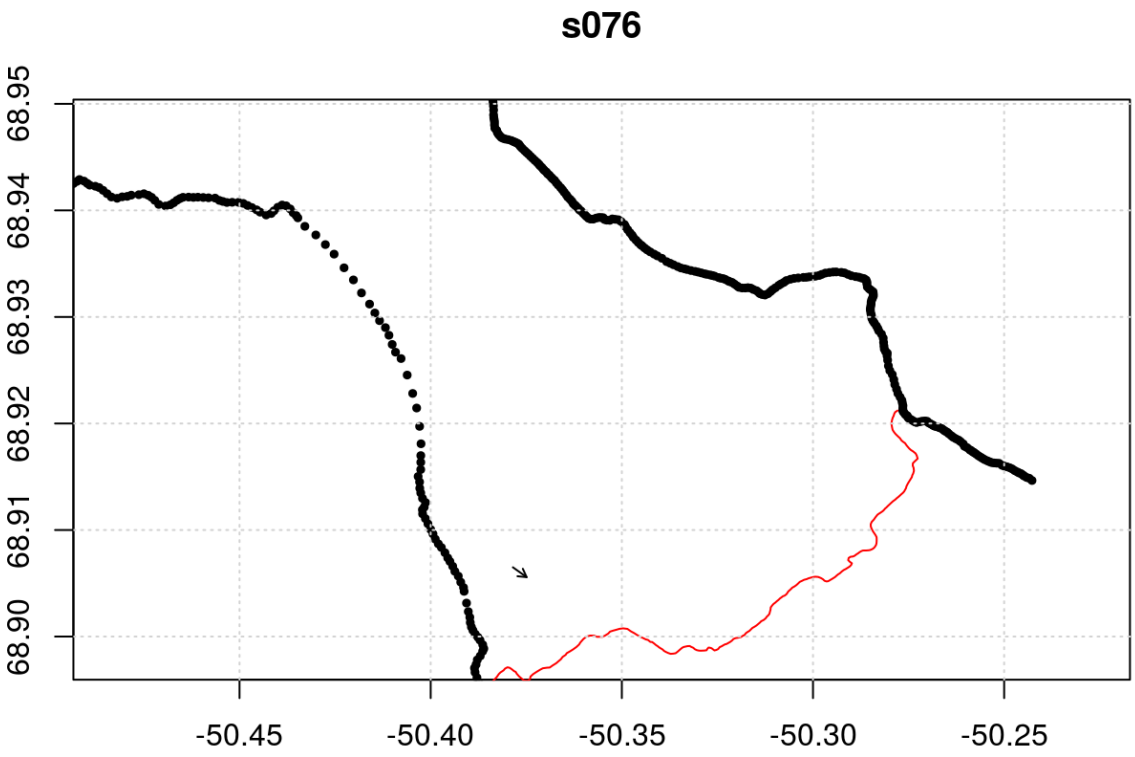

s074
68.95
68.94
68.93
68.92
68.91
68.90
-50.45
-50.40
-50.35
-50.30
-50.25

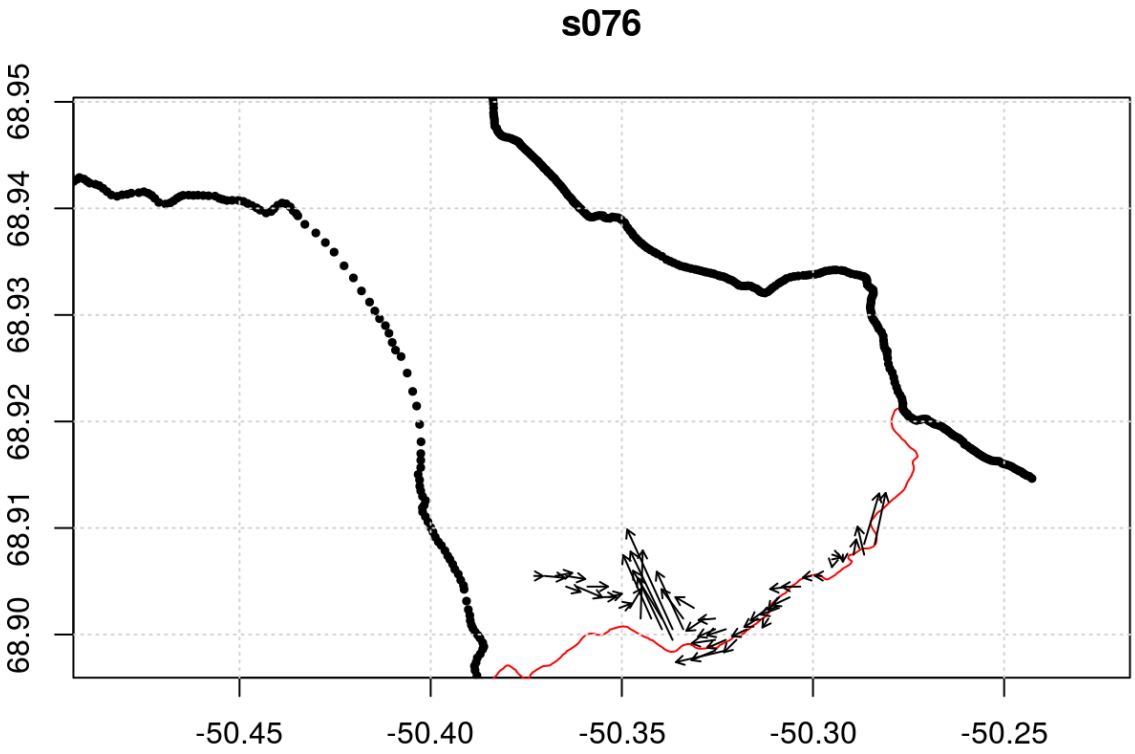

s075
68.95
68.94
68.93
68.92
68.91
68.90
-50.45
-50.40
-50.35
-50.30
-50.25

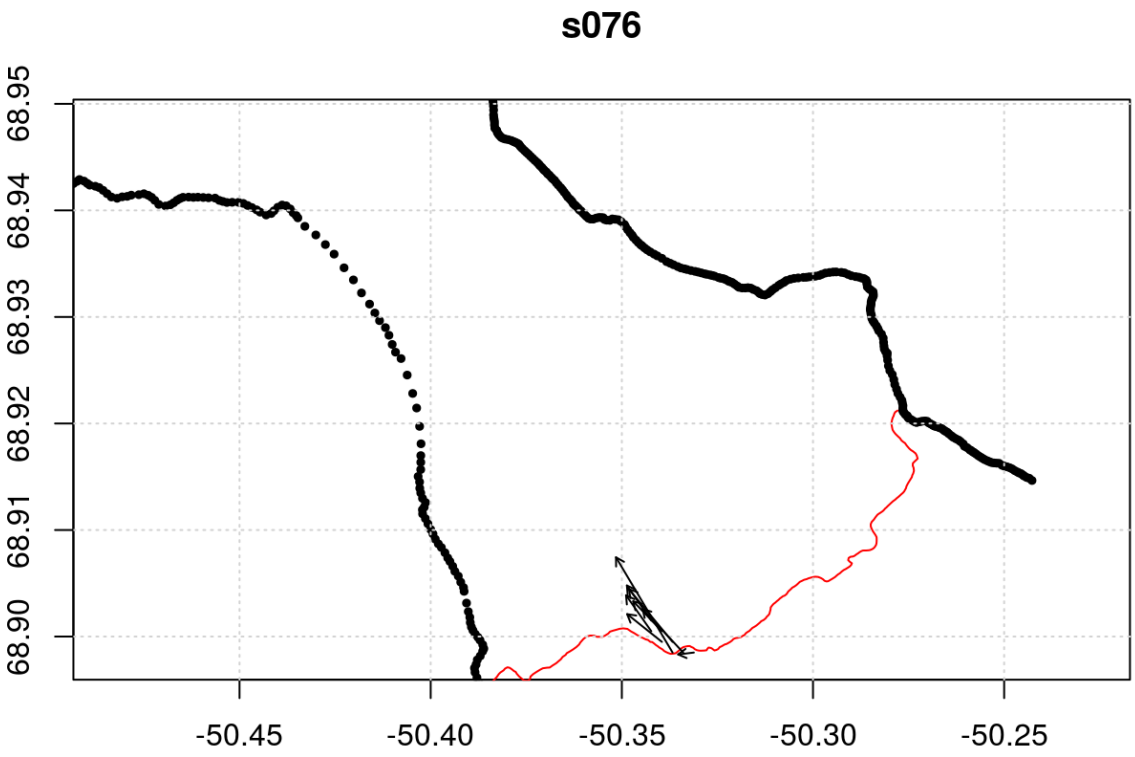


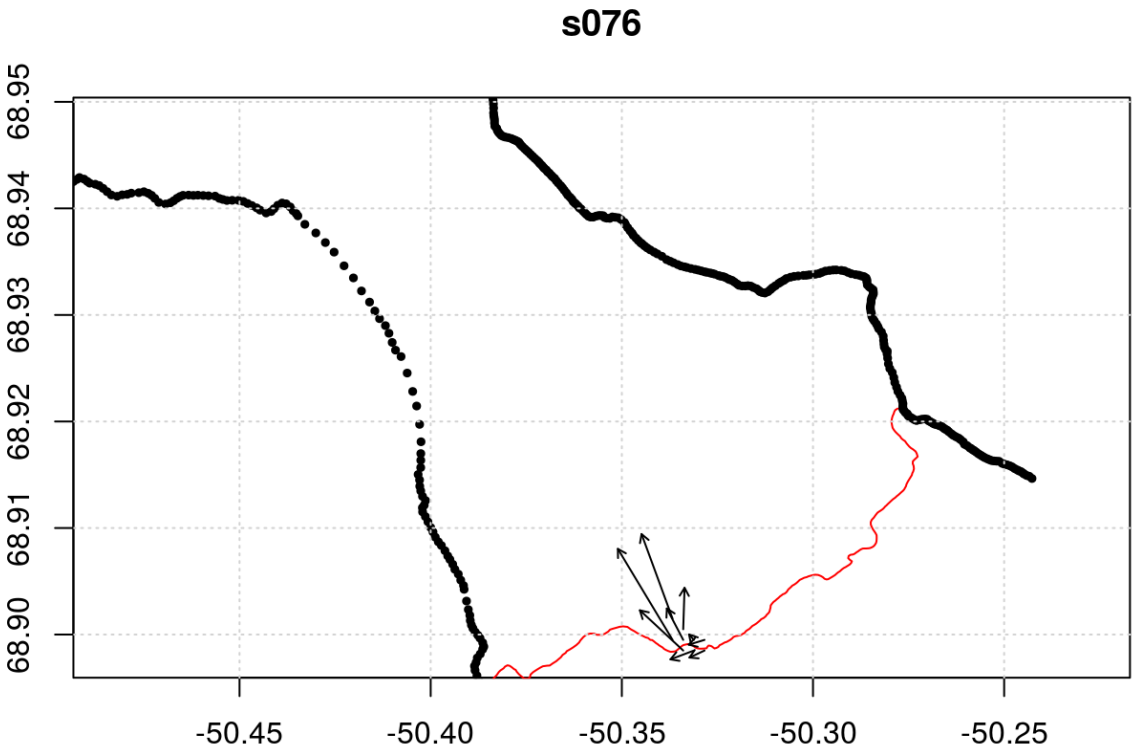
Just for fun, let's average the ensemble-averaged data onto a spatial grid to get rid of the high density of measurements in the places where the ship was moving slowly:

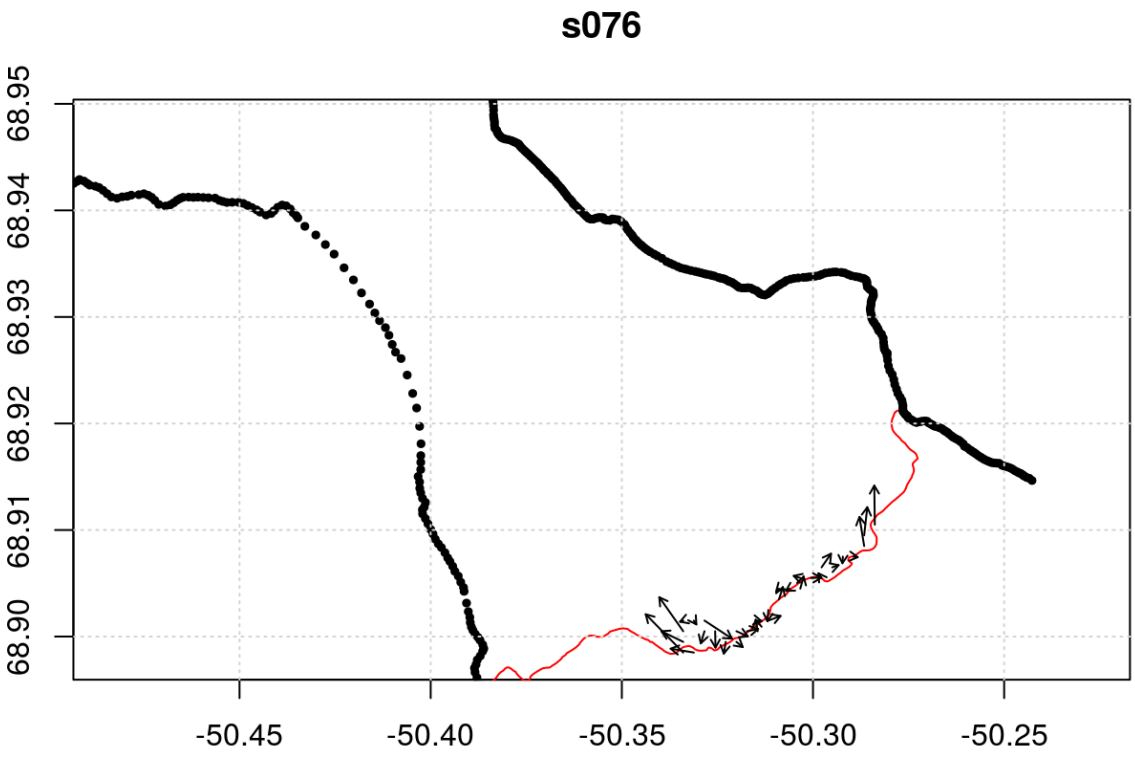
```
latg <- seq(68.89, 68.95, 0.001)
## scale the lon so that the boxes are approximately square
long <- seq(-50.43, -50.28, 0.001/cos(68.95*pi/180))

## vavg <- list()
ua <- va <- array(NA, dim=c(length(long)-1, length(latg)-1, length(enua)))
for (ii in seq_along(enua)) {
    d <- enua[[ii]]
    lon <- d[['firstLongitude']]
    lat <- d[['firstLatitude']]
    II <- d[['distance']] <= 50
    u <- apply(d[['v']][,II,1], 1, mean, na.rm=TRUE)
    v <- apply(d[['v']][,II,2], 1, mean, na.rm=TRUE)

    bu <- binMean2D(lon, lat, u, long, latg)
    bv <- binMean2D(lon, lat, v, long, latg)
    llon <- bu$xmids
    llat <- bu$ymids

    plot(clon, clat, asp=1/cos(68.95*pi/180), pch=19, cex=0.5,
         xlim=lonlim+c(-0.015, 0.015), ylim=latlim,
         xlab='', ylab='')
    lines(flon, flat, col=2)
    title(names(enua)[i])
    grid()
    drawDirectionField(llon, llat, bu$result, bv$result, scalex=0.1, type=2, ad
d=TRUE)

    ## vavg[[ii]] <- list(lon=llon, lat=llat, u=bu$result, v=bv$result)
    ua[,,ii] <- bu$result
    va[,,ii] <- bv$result
}
```

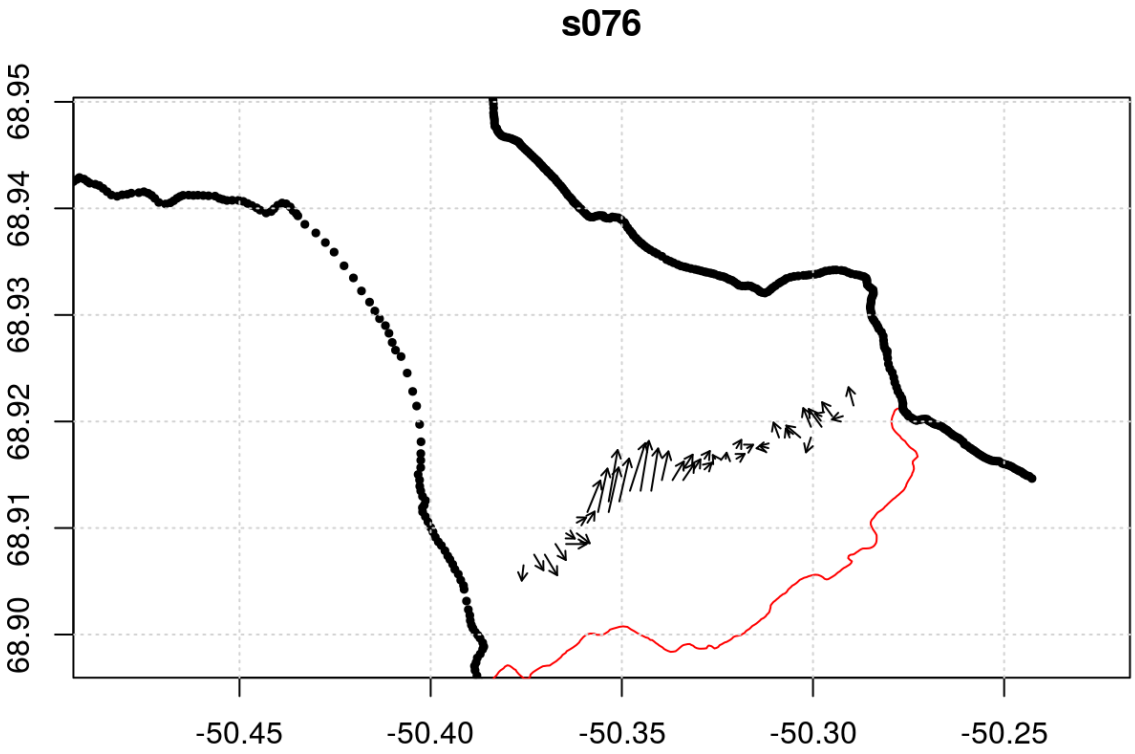

s076
68.95
68.94
68.93
68.92
68.91
68.90
-50.45
-50.40
-50.35
-50.30
-50.25

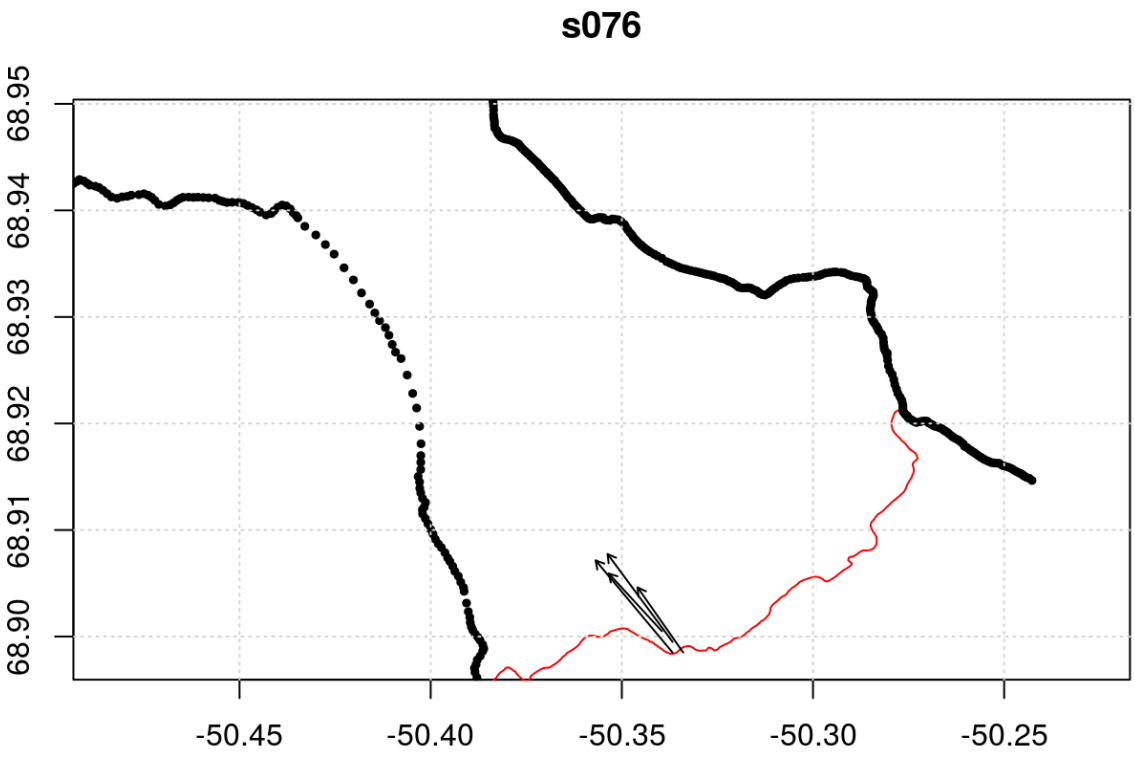

s076
68.95
68.94
68.93
68.92
68.91
68.90
-50.45
-50.40
-50.35
-50.30
-50.25

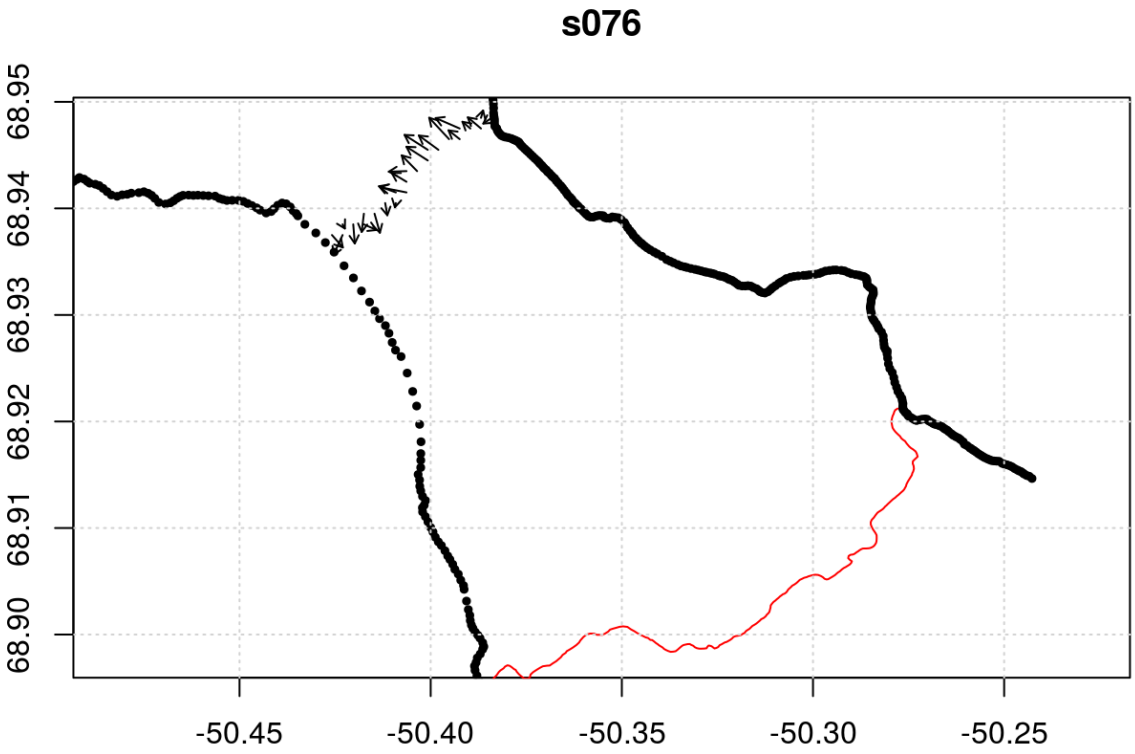

s076
68.95
68.94
68.93
68.92
68.91
68.90
-50.45
-50.40
-50.35
-50.30
-50.25

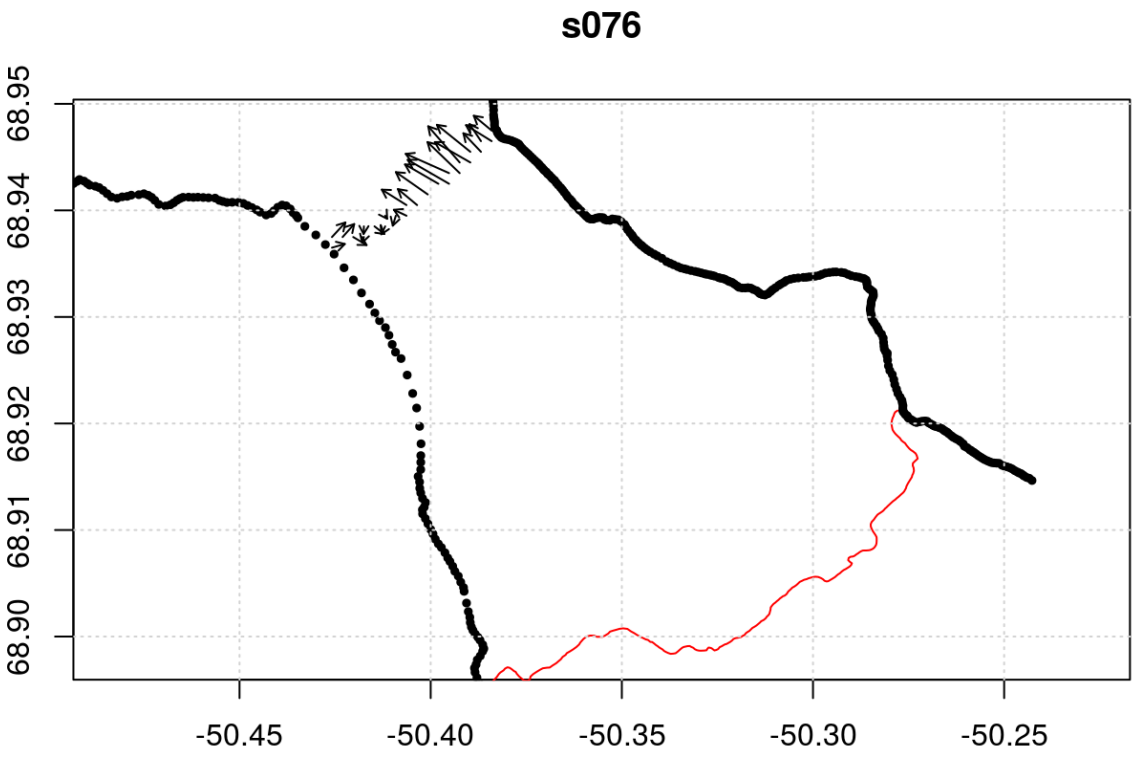

s076
68.95
68.94
68.93
68.92
68.91
68.90
-50.45
-50.40
-50.35
-50.30
-50.25

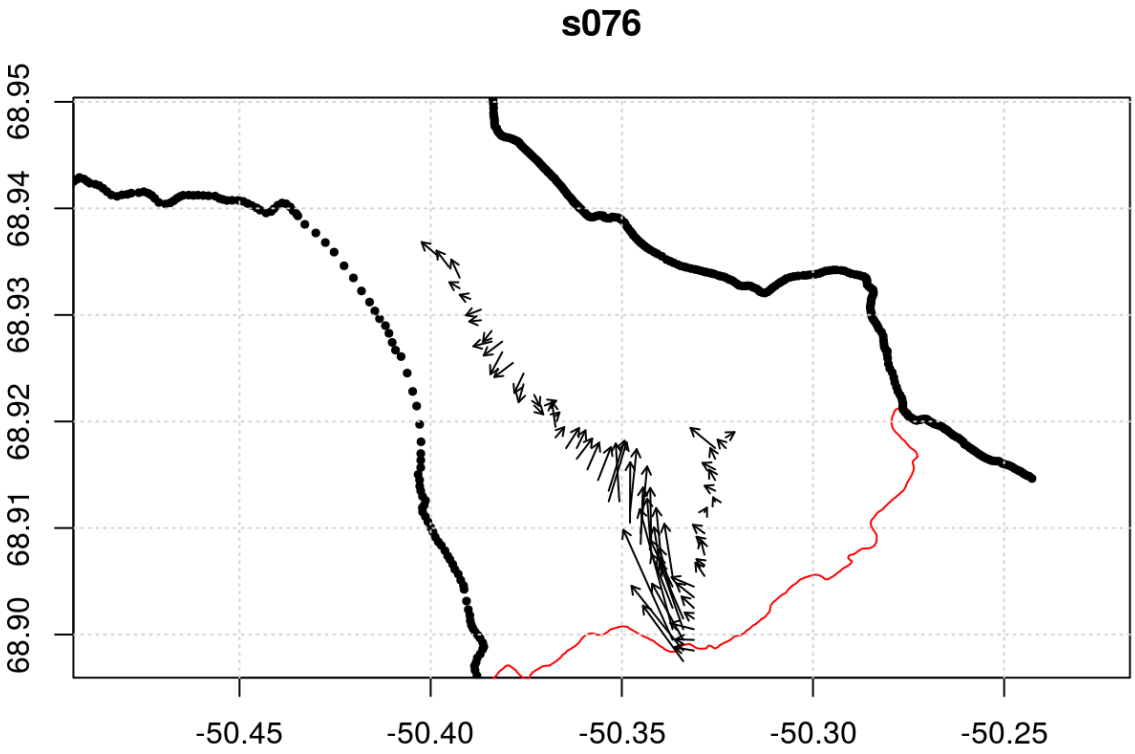

s076
68.95
68.94
68.93
68.92
68.91
68.90
-50.45
-50.40
-50.35
-50.30
-50.25


s076
68.95
68.94
68.93
68.92
68.91
68.90
-50.45
-50.40
-50.35
-50.30
-50.25


s076
68.95
68.94
68.93
68.92
68.91
68.90
-50.45
-50.40
-50.35
-50.30
-50.25


s076
68.95
68.94
68.93
68.92
68.91
68.90
-50.45
-50.40
-50.35
-50.30
-50.25

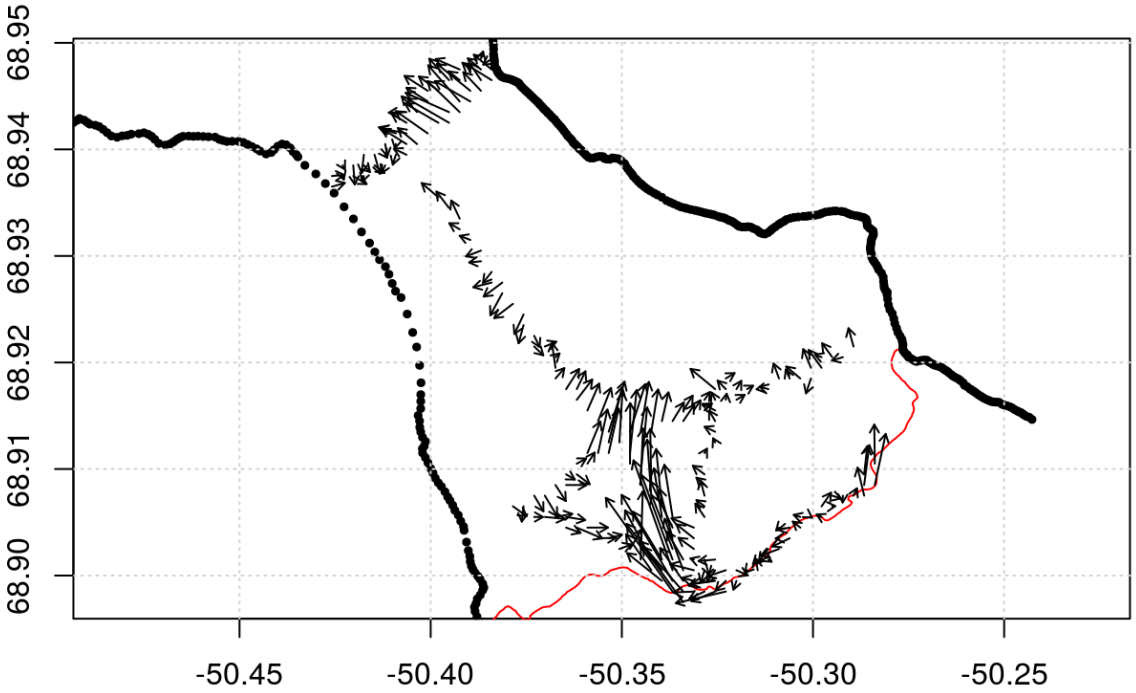

s076
68.95
68.94
68.93
68.92
68.91
68.90
-50.45
-50.40
-50.35
-50.30
-50.25


s076
68.95
68.94
68.93
68.92
68.91
68.90
-50.45
-50.40
-50.35
-50.30
-50.25

```
## names(vavg) <- names(adp)
```

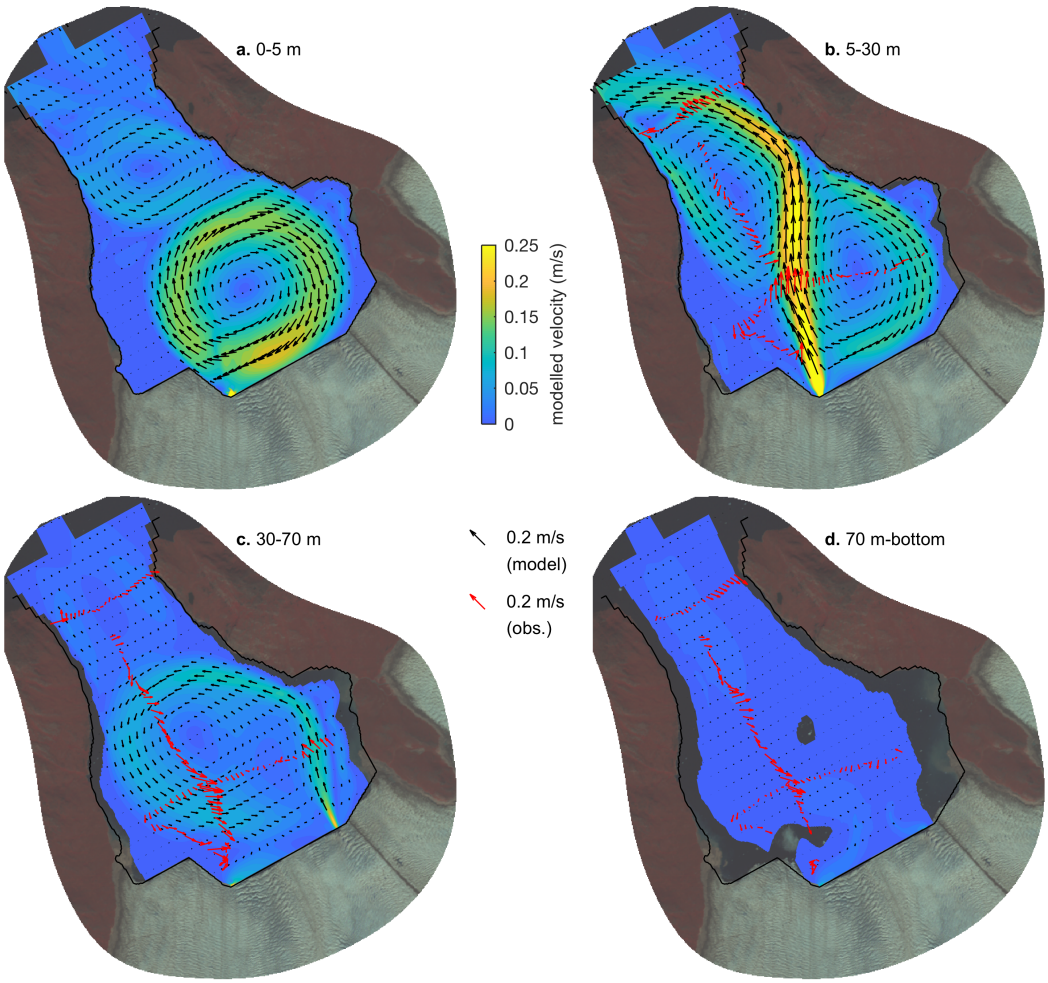
And, if we average the spatially-averaged fields all together:

```
u <- apply(ua, c(1, 2), mean, na.rm=TRUE)
v <- apply(va, c(1, 2), mean, na.rm=TRUE)
plot(clon, clat, asp=1/cos(68.95*pi/180), pch=19, cex=0.5,
     xlim=lonlim+c(-0.015, 0.015), ylim=latlim,
     xlab='', ylab='')
lines(flon, flat, col=2)
grid()
drawDirectionField(llon, llat, u, v, scalex=0.1, type=2, add=TRUE)
```

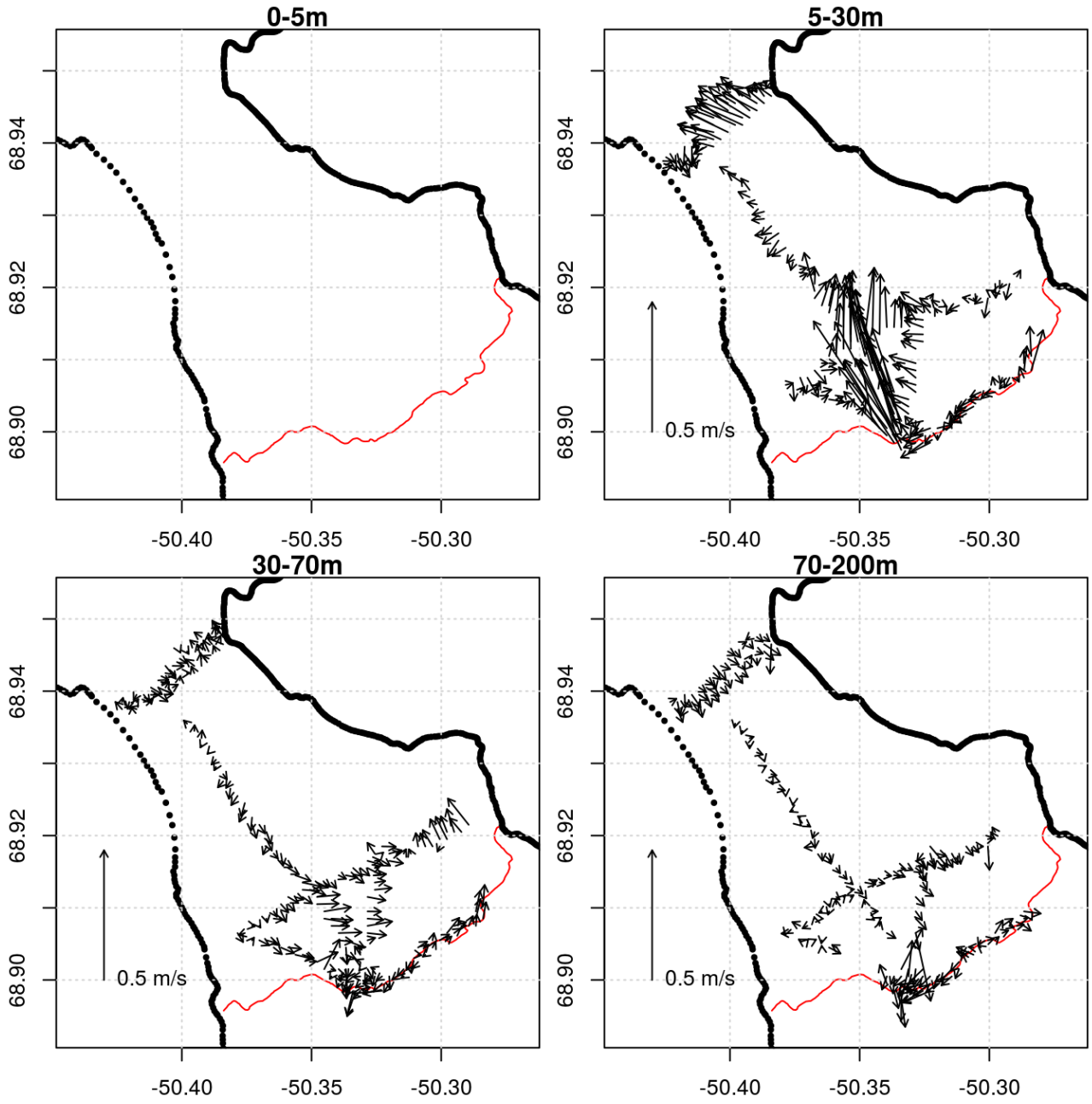


Depth dependence of velocity fields

Here I want to compare against Donald's ADCP/model comparison figure, which breaks the ADCP data up into 4 layers: 0-5 m (for which there is no ADCP data, since the first bin is centered at 7m), 5-30m, 30-70m, and 70-bottom.


a. 0-5 m
b. 5-30 m
c. 30-70 m
d. 70 m-bottom
0.25
0.2
0.15
0.1
0.05
0
modelled velocity (m/s)
0.2 m/s
(model)
0.2 m/s
(obs.)

```
depths <- c(5, 30, 70, 200)

U <- V <- array(NA, dim=c(length(long)-1, length(latg)-1, length(depths)-1))
for (jj in 1:(length(depths)-1)) {
    ua <- va <- array(NA, dim=c(length(long)-1, length(latg)-1, length(enua)))
    for (ii in seq_along(enua)) {
        d <- enua[[ii]]
        lon <- d[['firstLongitude']]
        lat <- d[['firstLatitude']]
        II <- d[['distance']] >= depths[jj] & d[['distance']] <= depths[jj+1]
        u <- apply(d[['v']][,II,1], 1, mean, na.rm=TRUE)
        v <- apply(d[['v']][,II,2], 1, mean, na.rm=TRUE)

        bu <- binMean2D(lon, lat, u, long, latg)
        bv <- binMean2D(lon, lat, v, long, latg)
        llon <- bu$xmids
        llat <- bu$ymids

        ua[,,ii] <- bu$result
        va[,,ii] <- bv$result
    }
    U[,,jj] <- apply(ua, c(1, 2), mean, na.rm=TRUE)
    V[,,jj] <- apply(va, c(1, 2), mean, na.rm=TRUE)
}

par(mfrow=c(2, 2), mar=c(2, 2, 1, 0.5))
for (i in 1:4) {
    plot(clon, clat, asp=1/cos(68.95*pi/180), pch=19, cex=0.5,
         xlim=lonlim+c(-0.015, 0.015), ylim=latlim,
         xlab='', ylab='')
    lines(flon, flat, col=2)
    grid()
    if (i != 1) {
        drawDirectionField(-50.43, 68.9, 0, 0.5, scalex=0.1, type=2, add=TRUE)
        text(-50.43, 68.9, '0.5 m/s', pos=4)
        drawDirectionField(llon, llat, U[,,i-1], V[,,i-1], scalex=0.1, type=2,
add=TRUE)
        title(paste0(depths[i-1], '-', depths[i], 'm'))
    } else {
        title(paste0('0-', depths[i], 'm'))
    }
}
```



Export to Matlab

For sharing the data are exported to Matlab format, using the `R.matlab` package. As I am not sure what format the previous ADCP data were provided in, I will export the raw single-ping ENU data for the individual sections, each with a matrix for U and V (east and north components), with dimensions of `time` x `depth` (rows x cols). The time vector is provided in the R "POSIX" format, which represents the number of seconds since 1970-01-01 00:00:00, in the UTC time zone. On testing using Octave, there was some weird behaviour with the time (perhaps because it is such a large number). As I don't have a working instance of Matlab on hand to test further, I will also provide a time vector that is the number of seconds since 2013-01-01 00:00:00 UTC, which seems to work as expected in Octave, and should be easy to convert to Matlab "datenum" format if required.

```
library(R.matlab)
```

```
## R.matlab v3.6.1 (2016-10-19) successfully loaded. See ?R.matlab for help.
```

```
##
## Attaching package: 'R.matlab'
```

```
## The following objects are masked from 'package:base':
##
##     getOption, isOpen
```

```
for (i in seq_along(enu)) {
    d <- enu[[i]]
    time_posix <- as.numeric(d[['time']])
    time_2013 <- as.numeric(d[['time']]) - as.numeric(as.POSIXct('2013-01-01 00
:00:00', tz='UTC'))
    distance <- d[['distance']]
    east <- d[['v']][,,1]
    north <- d[['v']][,,2]
    writeMat(paste0(names(enu)[i], '.mat'), time_posix=time_posix, time_2013=ti
me_2013, depth=distance, east=east, north=north)
}
```