Introduction to rivSurveyR

Jason L. Fischer

2016-5-16

Contents

Abstract

The `rivSurveyR` package accepts acoustic Doppler current profiler (ADCP) data exported from SonTek's RiverSurveyor Live program as MATLAB files and allows quick summarization and plotting of water velocities. It is capable of projecting replicate transects to a mean transect line and averaging measured values that overlap in geographic space. Planform and secondary velocities from multiple ADCP transects can be processed at once and multiple rotation methods can be used to determine secondary velocities. Planform flow data can be plotted in geographic space and cross-section plots of secondary velocities can easily be constructed. This vignette provides a walkthrough on the use of `rivSurveyR` functions and their underlying methods.

Introduction

Acoustic Doppler current profilers (ADCP) are capable of collecting large amounts of geographically and depth referenced flow data. Historically, the majority of ADCP measurements were conducted to obtain discharge, however, the large amount of spatially referenced measurements made by ADCPs make them useful for velocity mapping applications (Czuba et al. 2011; Parsons et al. 2013). Although the data collected by ADCPs may be ideal for velocity mapping, the large volume of data can make summarization a cumbersome process. Parsons et al. (2013) developed the Velocity Mapping Toolbox (VMT) a MATLAB based toolbox to streamline mapping ADCP velocity data. The VMT is a useful tool to quickly summarize and visualize ADCP data, but to date, has limited processing capabilities for data collected with SonTek ADCPs. This is in part due to the difficulty of calibrating compasses in the SonTek M9 and S5 units and the potential for poorly calibrated compasses to yield errors in velocity measurements when using GPS track reference to determine boat speed (Kevin Oberg, personal communication). However, this issue can be circumvented by using the Bottom-Track track reference mode to pre-process data in RiverSurveyor Live, before exporting data as a MATLAB file (this is only applicable if moving bed conditions are not present, since the Bottom-Track mode

uses the ADCP's movement relative to the river bed to determine unit speed and heading and must assume the river bed is stationary). Furthermore, recent compass upgrades address compass calibration difficulties ensuring the reliability of data pre-processed using the GPS-GGA or GPS-VTG track reference modes in RiverSurveyor Live.

The `rivSurveyR` package provides similar processing capabilities of the VMT for data collected using SonTek ADCPs using RiverSurveyor Live. It is capable of processing planform and three-dimensional flow data. First, replicate transects are projected to a mean transect line using an orthogonal projection. Next measurements that overlap in space are averaged. Water speeds, flow headings, and secondary velocities are then calculated. If substrate size is known shear velocities can be calculated using either Keulegan's resistance law from rough flow (Keulegan 1938; Garcia 2008) or the power law velocity profile used by RiverSurveyor Live (Simpson and Oltmann 1990; Chen 1991). Next, spatial averaging can then be conducted using a moving window of user defined size to reduce variation due to turbulence and better approximate mean velocity (Szupiany et al. 2007). Further, a moving window can be used to calculate standard deviation and determine the amount of spatial variability in velocities. Lastly, publication quality plots of planform velocities and depths and three-dimensional flow can easily be made. The functions and methods behind these steps are outlined in more detail in the following sections. While `rivSurveyR` eases the mapping process, it does not provide checks for data quality or compass errors and it is assumed that users are knowledgeable of potential errors (e.g. beam separation, poor compass calibration, GPS errors, etc.) and have performed the necessary QAQC prior to velocity mapping (see Mueller et al. 2013).

A number of the functions in `rivSurveyR` make use of the `data.table` structure, which inherits from and has many similarities to `data.frame`. The `data.table` structure offers faster processing than `data.frame`, which is advantageous when working with large datasets, such as those collected with ADCPs. However, `data.table` has a short learning curve and those who are not familiar with it are encouraged to visit the help pages by typing `?data.table` into the R Console. Nevertheless, the functions within `rivSurveyR` require little to no knowledge of the `data.table` structure and proficiency with `data.table` is not required to process and display data in `rivSurveyR`.

The first sections of this vignette provide a step-by-step walkthrough of `rivSurveyR`'s secondary functions to provide an overview of the data processing procedure. These sections discuss the functions called the primary processing functions discussed in the "Quickly process SonTek ADCP planform data" and "Quickly process SonTek ADCP cross-section data" sections and follow a similar procedure and order used by the functions discussed in those two sections. The final section provides a summary of other functions in `rivSurveyR`.

Processing SonTek ADCP planform data step-by-step

Compiling SonTek ADCP planform data

To get started load the `rivSurveyR` package and import ADCP files into R.

```
library(rivSurveyR)
```

`rivSurveyR` only accepts MATLAB files (.mat) which have been exported from RiverSurveyor Live (see RiverSurveyor Live help for details on converting RiverSurveyor Live files to MATLAB files). Additionally, all files must use an ENU (east, north, up) coordinate system, the coordinate system can be set in RiverSurveyor Live prior to exporting data as a MATLAB file. Once RiverSurveyor files have been converted to MATLAB format, they can be imported using `readMat` in the `R.matlab` package, which is included as part of `rivSurveyR`. Example MATLAB files are not provided, but the code below could be used to import the files corresponding to the replicated measurements of a transect.

```
transect1a <- readMat("transect1a.mat")
transect1b <- readMat("transect1b.mat")
```

Once all data has been imported into R, it can be compiled using the `xSec.planform` function. This function accepts a list of the transects imported into R and a list containing the names of those transects, as well as other optional arguments. Only the list of transects is required for the function to run. However, if the list of transect names is omitted, each file is assumed to represent a unique transect, which is only appropriate if there are no replicated transects. To get started an example set of MATLAB files which have already been imported into R has been provided as a list.

```
data(mNine)
names(mNine)
```

```
##  [1] "t1l"  "t1r"  "t2l"  "t2r"  "t3l"  "t3r"  "t4l"  "t4r"  "t5l"  "t5r"
## [11] "t6l"  "t6r"  "t7l"  "t7r"  "t8l"  "t8r"  "t9l"  "t9r"  "t10l" "t10r"
```

The data to be passed to the data argument of `xSec.planform` has already been coerced into mNine. The objects within the mNine list have been named to represent the transect and the starting bank (l or r) of each replicate, these can be used to name the transects, which can be done by creating a character list of the transect names.

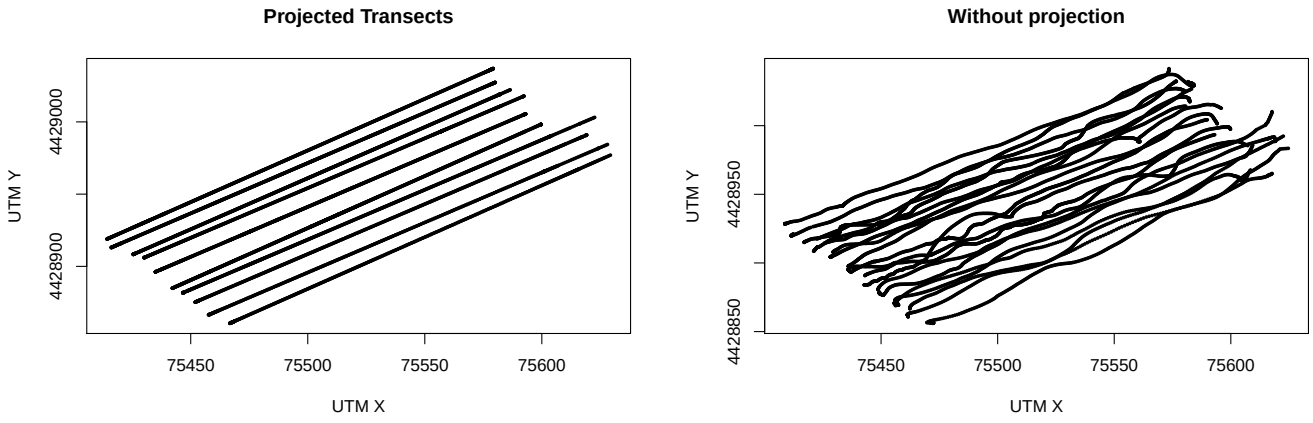
```
tNames <- list("t1l", "t1r", "t2l", "t2r", "t3l", "t3r", "t4l", "t4r")
```

Typing a list of names as above can be time consuming and is prone to typos, if objects in the data list have been named as in mNine, an alternative is to drop the last character identifying the replicate/starting bank from the names of the listed transect replicates.

```
tNames <- substr(names(mNine), 0, nchar(names(mNine))-1)
```

Once the dataset and transect names have been defined, the planform data can be compiled and projected to a mean transect line.

```
planform <- xSec.planform(mNine, tNames)
```



This compiles the transects using the default settings, where the depth measurements used are those defined in RiverSurveyor prior to exporting MATLAB files, layer averaged velocities correspond to the average velocity within the depth bounds of the ADCP measurements, and data is projected to a mean transect line. However, if there is interest in water velocities within a meter from the bottom, setting the layerReference to "bottom" defines the bounds of x1 and x2 as the distance from the bottom. Setting x1 = 1 and keeping x2 = "minimum" ensures that the layer averaged velocities is the average of the velocities measured between the bottom and 1 meter off the bottom.

```
planform <- xSec.planform(mNine, tNames, "VB", "bottom", 1, "minimum")
```

If x1 is greater than the maximum value of x, warnings will be produced, indicating that a NA was produced instead of a depth averaged velocity. If this is a concern, the value of x1 can be increased.

Averaging planform data from SonTek ADCP measurements

With all the flow data compiled it can be summarized using `average.planform`, this function takes ADCP data and applies a function summarizing overlapping sample points or ensembles. The default is to average overlapping ensembles, but other functions, such as standard deviation or count, can be used by calling the desired function with the FUN argument. To get started, supply the `planform` dataset to `average.planform`.

```
mean.planform <- average.planform(planform)
nrow(planform)
```

```
## [1] 7940
```

```
nrow(mean.planform)
```

```
## [1] 5982
```

In addition to providing the mean for each set of replicates (from replicate transects), the transect heading, distance each sample is from the start of the transect (transects are assumed to start river right), depth averaged and layer averaged water speeds, and the compass heading of depth averaged and layer averaged flow for each ensemble are calculated. Behind the scenes, the x and y UTM coordinates of projected mean transect lines are used to determine the compass heading of the transect when moving from river right to left and then the distance each ensemble is from the starting point of the transect is determined. The summarizing function (e.g. mean) is then applied to ensembles of equal distance along the same transect. Water speeds are calculated with the Pythagorean theorem (speed <- sqrt(velocity east^2 + velocity north^2)) and `heading` is used to determine compass heading.

Alternatively, a list of the transects imported into R and a list containing the names of those transects can be passed to `average.planform` and the first step can be skipped. Notice that many of the arguments are the same as those for `xSec.planform`, this is because if a list of transects is provided to the data argument, this information is passed to `xSec.planform` to be compiled before summarizing.

```
mean.planform.alt <- average.planform(mNine, tNames, depthReference="VB",
                                      layerReference="bottom", x1=1, x2="minimum")
all.equal(mean.planform.alt, mean.planform)
```

```
## [1] TRUE
```

If the standard deviation is desired:

```
sd.planform <- average.planform(mNine, tNames, FUN = sd, depthReference="VB",
                                layerReference="bottom", x1=1, x2="minimum")
```

Another option provided by `average.planform` is the `binWidth` or cell width used to group samples for summarization. The default is to use the mean distance between samples, which is essentially the mean width of ensembles measured in a transect. However, if there is need to summarize samples over larger distances, the distance used can be set with `binWidth`. The larger the `binWidth` the quicker `average.planform` completes summarization.

```
mean.planform.meter <- average.planform(mNine, tNames, binWidth=1, depthReference="VB",
                                        layerReference="bottom", x1=1, x2="minimum")
nrow(mean.planform.meter)
```

```
## [1] 2014
```

```
mean.planform.fiveMeter <- average.planform(mNine, tNames, binWidth=5, depthReference="VB",
                                            layerReference="bottom", x1=1, x2="minimum")
nrow(mean.planform.fiveMeter)
```

```
## [1] 411
```

Estimating shear velocity

If the sediment diameter of the 90th percentile and n for the study area are known, the shear velocity can be estimated. Here Dc is set to 0.0375m to represent a river bed of hard-pan clay covered with zebra mussels and velocity vectors from the last measured cell are used to calculate shear velocity.

```
shearVelocity <- shearVel(mean.planform.meter$BC.Vel.E, mean.planform.meter$BC.Vel.N,
                          mean.planform.meter$BC.Vel.Up, mean.planform.meter$depth,
                          mean.planform.meter$bottomCellDepth, Dc=0.0375, n=2, p=1000,
                          reference="bottomCell", units="metric")
shearVelocity
```

```
##            ustarE       ustarN      ustar ustarHeading
##    1:  0.04007887 -0.033856554 0.05251445    130.18939
##    2:  0.03375578 -0.035005989 0.04872162    136.04163
##    3:  0.02594672 -0.029411381 0.03930087    138.58126
##    4:  0.03281196 -0.038692766 0.05074006    139.70163
##    5:  0.02059405 -0.025753070 0.03307510    141.35160
##   ---
## 2010: -0.01777415 -0.020271340 0.02696089    221.24466
## 2011:  0.01816185 -0.010454139 0.02749996    119.92515
## 2012:  0.01951470  0.021355706 0.03303732     42.42085
## 2013:  0.03508477 -0.009796942 0.03881256    105.60166
## 2014:  0.02901785  0.016967382 0.03591764     59.68423
```

```
#Append to mean.planform
mean.planform.meter <- cbind(mean.planform.meter, shearVelocity)
```

Spatial averaging and standard deviation

Once the data has been summarized, it can be spatially averaged with `spatialAverage`, which uses a moving window of x and y (if twoD=TRUE) neighbors to average the each variable. To average the `mean.planform` dataset, define the number of units of the moving window in the horizontal direction (i.e., `xWindow`), define the coordinates used to project the data in space (i.e., `spatialCoords`), and the `groups` used to separate the transects. Here the `xWindow=20`, the `spatialCoords="tDist"` (the distance along a transect), and setting `groups="transectName"` ensures each transect is spatially averaged separately. Since averaging will be conducted along individual transects, `twoD=FALSE`, the default.

```
spatialMean.planform <- spatialAverage(mean.planform.meter, xWindow=20, spatialCoords="tDist",
                                       groups="transectName")
```

```
## Spatial Averaging in Process, Please Be Patient

## One Dimensional Spatial Averaging Complete

## Window Size: 20 cells
```

```
dim(spatialMean.planform)
```

```
## [1] 2014   33
```

```
dim(mean.planform.meter)
```

```
## [1] 2014   33
```

Similarly, `spatialSD` can be used to calculate the spatial standard deviation

```
spatialSD.planform <- spatialSD(mean.planform.meter, xWindow=20, spatialCoords=c("tDist"),
                                groups=c("transectName"))
```

```
## Spatial Standard Deviation Being Calculated, Please Be Patient

## One Dimensional Standard Deviation Calculated

## Window Size: 20 cells
```

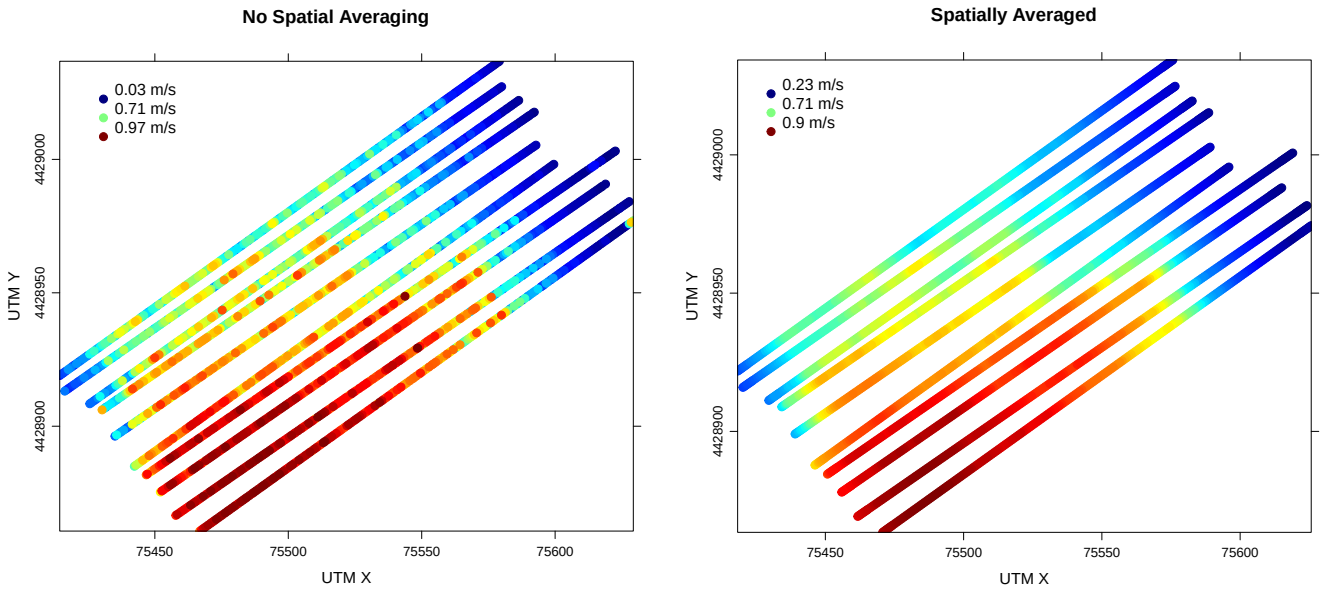
```
dim(spatialSD.planform)
```

```
## [1] 2014   33
```

```
dim(mean.planform.meter)
```

```
## [1] 2014   33
```

The plots below show depth averaged water speeds before and after spatial averaging.



Processing SonTek ADCP cross-section data step-by-step

Compiling SonTek ADCP cross-section data

Cross-sections of transects can also be processed following a procedure similar to the one used to process the planform data. The data that was imported into R in the "Compiling SonTek ADCP planform data" section can be used for these steps. Similar to `xSec.planform`, `xSec.secondary` takes a list of files exported from RiverSurveyor Live as MATLAB files and a character list containing the transect name each file represents and compiles the flow data for each cell within an ensemble. This can result in large datasets and the input files for this example are trimmed for the sake of speed.

```
mNineTrim <- mNine[1:6]
tNamesTrim <- tNames[1:6]
```

Next supply the trimmed lists to `xSec.secondary`.

```
secondary <- xSec.secondary(data=mNineTrim, transectNames=tNamesTrim)
```

Averaging cross-section data from SonTek ADCP measurements

The flow data can now be summarized using `average.secondary`, this function takes ADCP data and applies a function summarizing overlapping cells, similar to `average.planform`.

```
mean.secondary <- average.secondary(secondary)
nrow(secondary)
```

```
## [1] 44386
```

```
nrow(mean.secondary)
```

```
## [1] 26383
```

In addition to providing the mean for each set of replicates (from replicate transects), the transect heading, distance each sample is from the start of the transect (transects are assumed to start river right), cell width, water speed within a cell, the compass heading of flow within a cell, and altitude of a cell are calculated.

As with `average.planform` the list of transects imported into R and the list containing names of those transects can be passed directly to `average.secondary`.

```
mean.secondary.alt <- average.secondary(mNineTrim, tNamesTrim)
all.equal(mean.secondary, mean.secondary.alt)
```

```
## [1] TRUE
```

Lastly, the bin sizes used to group samples for summarization can be defined with `binWidth` (see "Averaging planform data from SonTek ADCP measurements" for details) and `binHeight`. `binHeight` controls the cell height used to group samples for summarization, which defaults to the mean cell height if this argument is not supplied.

```
mean.secondary.meterXHalfMeter <- average.secondary(mNineTrim, tNamesTrim, binWidth=1,
                                                    binHeight=0.5)
nrow(mean.secondary.meterXHalfMeter)
```

```
## [1] 9556
```

```
nrow(mean.secondary)
```

```
## [1] 26383
```

Rotations: Primary and secondary velocities

Water velocities measured by the ADCP are recorded as the magnitude to the east, north, and up (when using the ENU coordinate system). This references flow in geographic space, but provides little bearing on flow directions relative to the cross-section measured. Three rotation methods are available in `rivSurveyR` that calculate velocity parallel and perpendicular to the cross-section and primary and secondary velocities.

Calculate cross-section velocities parallel and perpendicular to the cross-section The first method (`xSec`) calculates the velocity vectors that are parallel and perpendicular to a cross-section. To begin supply `xSec` with velocities in the East, north, and up directions and the heading of a transect. `xSec`, `zeroSecQ`, and `rozovskii` assume all velocities correspond to the same transect and it is important that the data passed to `xSec`, `zeroSecQ`, and `rozovskii` is subset to contain only the transect of interest.

```
tThree.secondary <- mean.secondary.meterXHalfMeter[transectName=="t3",,]
xSec.tThree <- xSec(tThree.secondary$Vel.E, tThree.secondary$Vel.N, tThree.secondary$Vel.up,
                    mean(tThree.secondary$transectHeading, na.rm=TRUE))
#Add the cross-section velocities to the data set
tThree.secondary <- data.table(tThree.secondary, xSec.tThree)
```

This can be tedious if multiple transects need to be processed. Fortunately, other methods are available to process multiple transects at once, which are discussed in the "Process SonTek ADCP cross-section data" section.

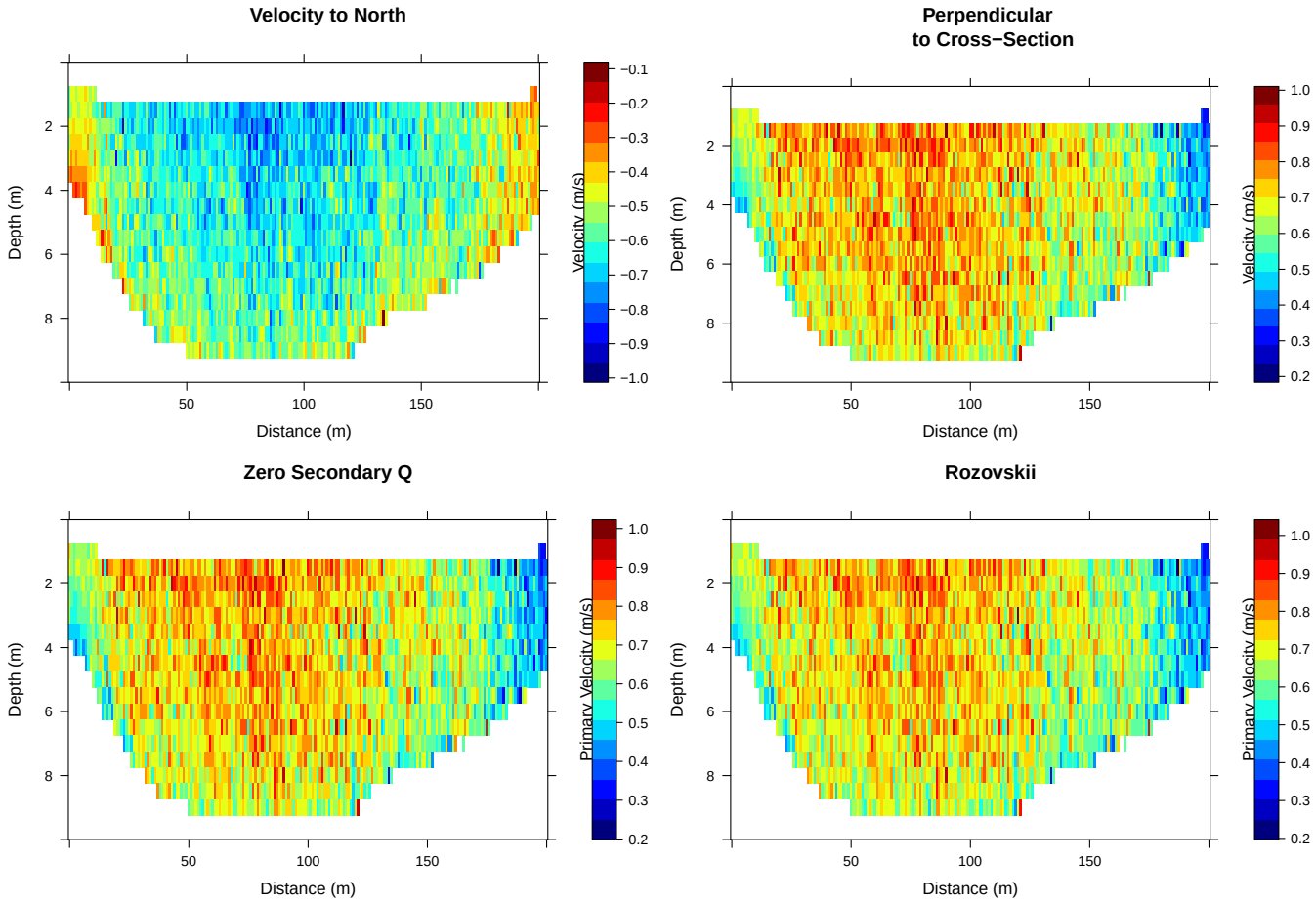
Calculate primary and secondary velocities using the zero secondary discharge method of rotation The second rotation method (`zeroSecQ`) uses the zero secondary discharge method of rotation to calculate primary and secondary velocities (see Lane et al. 2000 for details). `zeroSecQ` has the same data requirements as `xSec`, with the addition of the height and width of each cell sampled.

```
zsq.tThree <- zeroSecQ(tThree.secondary$Vel.E, tThree.secondary$Vel.N, tThree.secondary$Vel.up,
                       tThree.secondary$cellHeight, tThree.secondary$cellWidth,
                       mean(tThree.secondary$transectHeading, na.rm=TRUE))
tThree.secondary <- data.table(tThree.secondary, zsq.tThree)
```

Calculate primary and secondary velocities using the Rozovskii method of rotation Primary and secondary velocities are calculated using the Rozovskii method of rotation (Rhoads and Kenworthy 1998) in the third rotation method available (`rozovskii`). `rozovskii` requires velocities in the East, north, and up directions and East and North depth averaged velocities for each ensemble. As with the other methods of rotation, `rozovskii` assumes all data were measured along the same transect.

```
roz.tThree <- rozovskii(tThree.secondary$Vel.E, tThree.secondary$Vel.N, tThree.secondary$Vel.up,
                        tThree.secondary$Mean.Vel.E, tThree.secondary$Mean.Vel.N)
tThree.secondary <- data.table(tThree.secondary, roz.tThree)
```

The plots below show the water velocity with and without rotation.



Spatial averaging of cross-section velocities

The cross section data is still contains variation due to turbulence (Szupiany et al. 2007; Czuba et al. 2011), to better represent mean flows, spatial averaging can be conducted. The procedure follows as above, with a few differences. Because spatial averaging will be conducted by the distance along the transect, as well as by depth, `twoD=TRUE` and the number of units of the vertical moving window is supplied to `yWindow`. Additionally, "tDist" and "cellDepth" is passed to the `spatialCoords` argument, telling the moving window to move along each transect vertically by depth as well as horizontally by distance along the transect.

```
spatialMean.secondary <- spatialAverage(tThree.secondary, xWindow=20, yWindow=9,
                                        spatialCoords=c("tDist", "cellDepth"),
                                        groups=c("transectName"))
```

```
## Spatial Averaging in Process, Please Be Patient

## Two Dimensional Spatial Averaging Complete

## Window Size: 20x9 cells
```

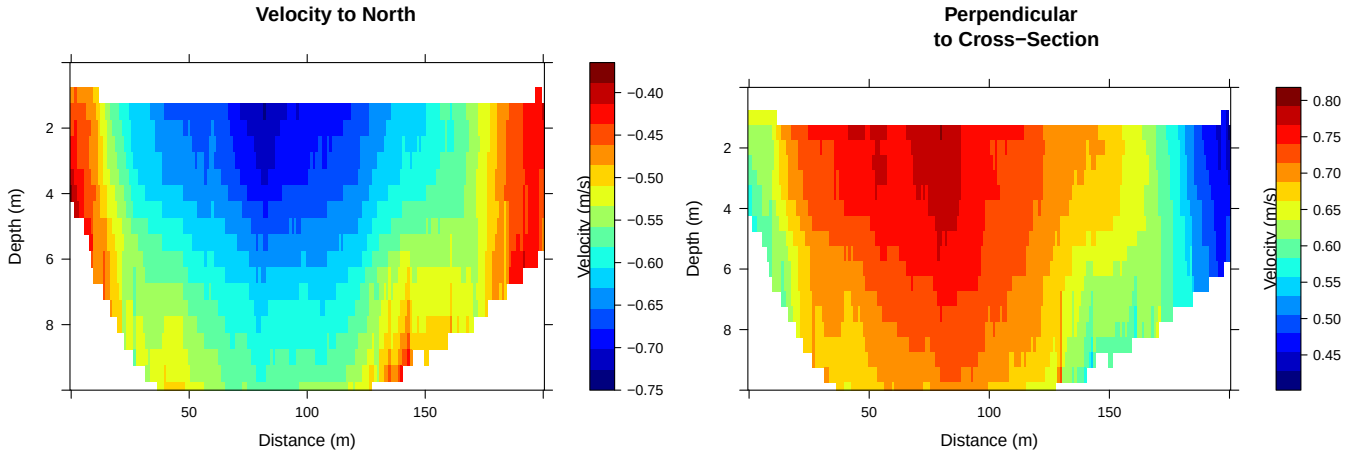
```
dim(spatialMean.secondary)
```

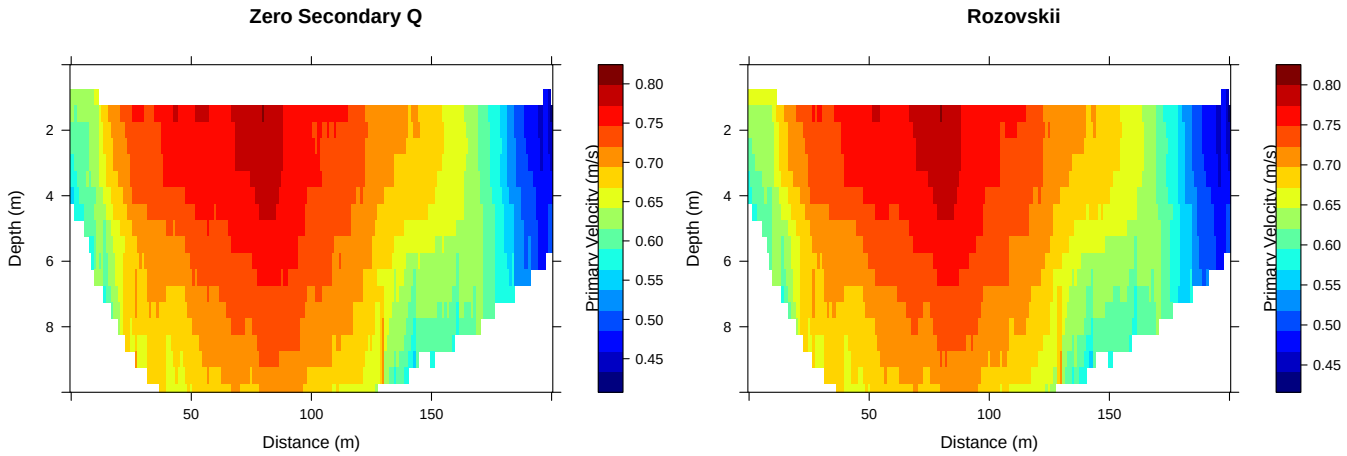
```
## [1] 3198   54
```

```
dim(tThree.secondary)
```

```
## [1] 3198   54
```

The figures below show the rotations after spatial averaging





Quickly process SonTek ADCP planform data

The previous sections provide a detailed example of the processing procedure and overview the functions used at each step. Although those functions can be used individually to process the ADCP data, `rivSurveyR` contains two functions which automate the steps used to process planform and cross-section data (`process.planform` and `process.secondary`). Planform data can easily be processed in one step with `process.planform`, which compiles the functions used in the "Processing SonTek ADCP planform data step-by-step" section. `process.planform` inherits all the arguments of those functions and provides some defaults to assist with processing. A list of transect files to process is the minimum that needs to be passed to `process.planform`. This runs the function with all the defaults and assumes transects are not replicated (i.e., all files are from individual transects). The default is to compute the layer averaged velocity using the entire measured water column, project the mean values of overlapping samples to the projected transect line, and spatial averaging is not conducted. Additionally, because the default omits Dc and n, shear velocities are not calculated.

```
planform.ADCP.basic <- process.planform(mNine)
```

If something other than the default is desired, data processing can be customized by specifying the arguments passed to `process.planform`. The code below produces the same output that was produced in the "Processing SonTek ADCP planform data step-by-step" section.

```
planform.ADCP <- process.planform(mNine, tNames, depthReference="VB",
                                  layerReference="bottom", x1=1, x2="minimum",
                                  binWidth=1, Dc=0.0375, n=2, xWindow=20)
```

```
## Compiling and Averaging Data

## Calculating Bottom Velocities

## Spatial Averaging in Process, Please Be Patient

## One Dimensional Spatial Averaging Complete

## Window Size: 20 cells
```

```
all.equal(spatialMean.planform, planform.ADCP)
```

```
## [1] TRUE
```

Additionally, to avoid confusing results shear velocity and spatial averaging are only conducted if `FUN=mean`.

Quickly process SonTek ADCP cross-section data

Similar to `process.planform`, `process.secondary` can be used to easily process cross-section velocities. At the minimum, only a list of transect files needs to be passed to `process.secondary`. If only a list of transect files are supplied, `process.secondary` is run with all the defaults, which are to project mean values of overlapping samples to a mean transect line, use mean cell height and horizontal distance between samples as binHeight and binWidth (respectively), calculate velocities orthogonal to the mean transect line, and spatial averaging is not conducted. Also, similar to `process.planform`, velocity rotations and spatial averaging are only conducted if `FUN=mean`. The example below yields the same output as the example in the "Processing SonTek ADCP cross-section data step-by-step" section.

```
secondary.ADCP <- process.secondary(mNineTrim, tNamesTrim, binWidth=1, binHeight=0.5,
                                    rotation = c("xSec", "zeroSecQ", "rozovskii"),
                                    xWindow=20, yWindow=9)
```

```
## Compiling and Averaging Data
```

```
## Spatial Averaging in Process, Please Be Patient
```

```
## Two Dimensional Spatial Averaging Complete
```

```
## Window Size: 20x9 cells
```

```
all.equal(spatialMean.secondary, secondary.ADCP[transectName=="t3",,],
          check.attributes=FALSE)
```

```
## [1] TRUE
```

In the "Processing SonTek ADCP cross-section data step-by-step" section, the dataset had to be subset by transect before being passed to a rotation function and only one transect could be processed at a time. This process is automated in by `process.secondary`, allowing individual rotations to occur for all transects, reducing the amount of code needed and preventing typos from being introduced.

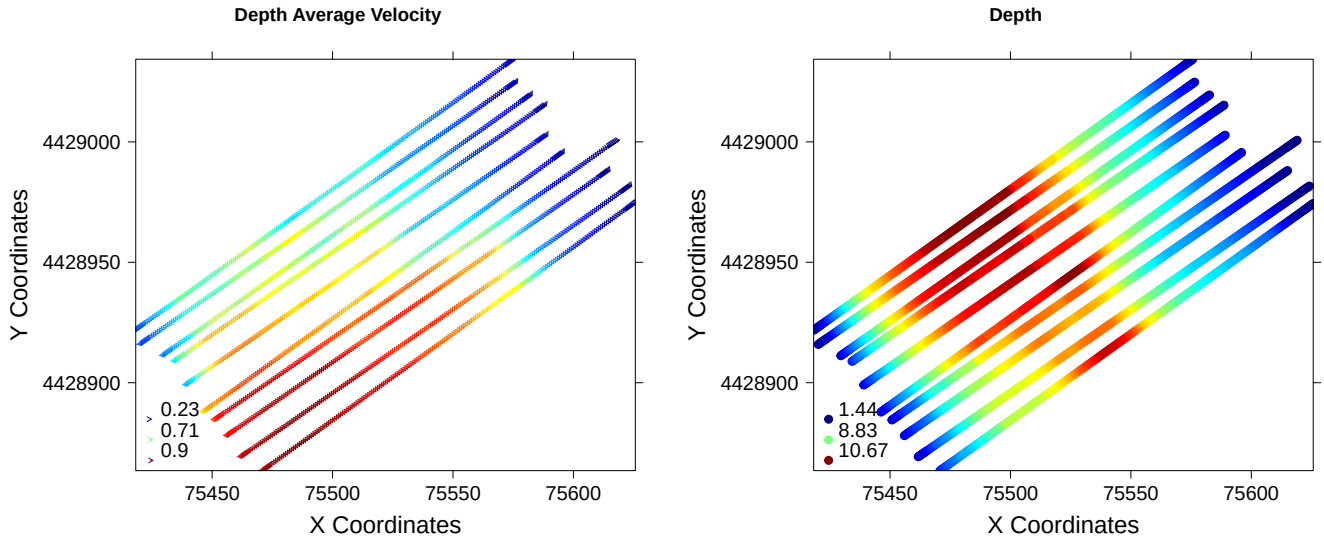
Plotting data

Once the ADCP data has been processed, plots of planform depths and velocities and cross-section velocities can be made. The `plot.xSec` and `plot.planform` plotting functions in `rivSurveyR` are based off the `xyplot` and `levelplot` functions in the `lattice` package and offer a number of options for flexible displays. Both functions grid the data (optional with `plot.planform`) and use inverse distance weighting (see `krige` in package `gstat` for details) to interpolate areas between points, to achieve a smooth display of mapped depths or water velocities.

Plot planform data

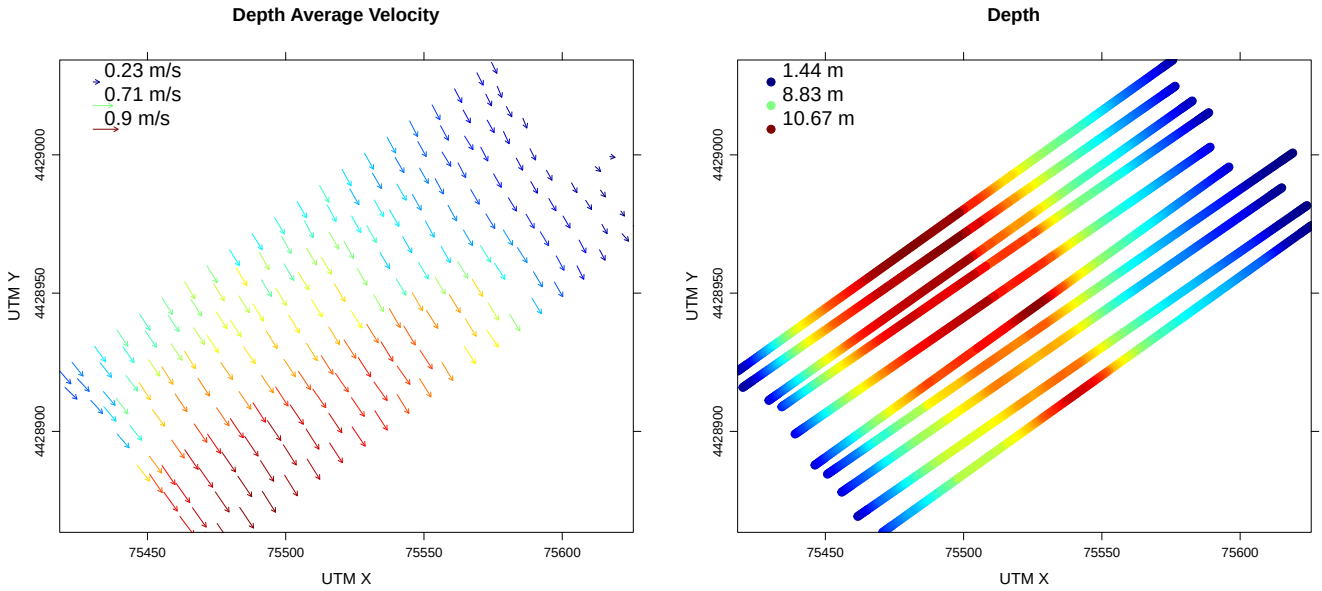
`plot.planform` offers a number of options for displaying planform data. If a dataset of processed ADCP planform data and a parameter from the dataset are passed to `plot.planform` the projected sample points along each transect are displayed with a blue to red color gradient to highlight low and high values of the parameter passed to `plot.planform` and arrows are used to display the magnitude and heading of the parameter (only if the parameter have x, y velocity vectors, otherwise, just the arrow heads are displayed). If a parameter without velocity vectors (e.g., depth) is to be displayed, sample points can be displayed as dots instead of arrow heads by setting `arrows=FALSE`. A basic plot can be produced with the code below.

```
plot.planform(planform.ADCP, DepthAveragedSpeed, main="Depth Average Velocity")
plot.planform(planform.ADCP, depth, arrows=FALSE, main="Depth")
```



The arrows in the plot above are small and displayed closely together. These arrows can be made larger by adjusting the `arrow_scale` and spacing can be increased by increasing the `point.spacing` value. Additionally, units can be added to the axis scale with `xUnits`. If the key overlaps data points, it can be positioned by defining the x and y coordinates of the key with `arrow.key`. The x and y-axis labels can be defined with xlab and ylab and the y-axis tick labels can be rotated with `scales=list(y=list(rot=90))` to make more efficient use of space.

```
plot.planform(planform.ADCP, DepthAveragedSpeed, point.spacing=10, arrow.key=c(75420,4429005),
              main="Depth Average Velocity", arrow_scale=10, xUnits="m/s", xlab="UTM X",
              ylab="UTM Y",  scales = list(y=list(rot=90)))
plot.planform(planform.ADCP, depth, arrows=FALSE, arrow.key=c(75420,4429005), main="Depth",
              xUnits="m", xlab="UTM X", ylab="UTM Y", scales=list(y=list(rot=90)))
```



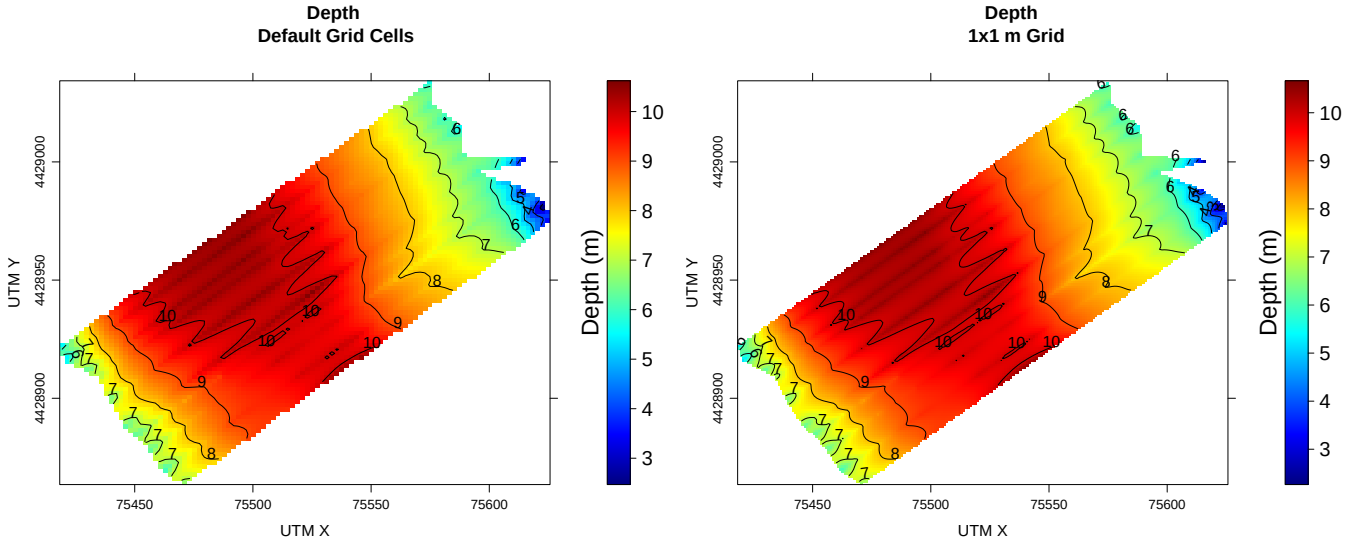
`plot.planform` also has an option for displaying a parameter as an interpolated raster that is confined within the bounds of the sampled area. The default cell size is the x-axis range/100 by the y-axis range/100, but can be specified with the `cellWidth` and `cellHeight` arguments.

```
plot.planform(planform.ADCP, interpolate=depth, main="Depth \nDefault Grid Cells", xlab="UTM X",
              ylab="UTM Y", ylab.right=list(label="Depth (m)", fontsize=18),
              scales=list(y=list(rot=90)))
```

```
## [inverse distance weighted interpolation]
```

```
plot.planform(planform.ADCP, interpolate=depth, cellWidth=1, cellHeight=1,
              main="Depth \n 1x1 m Grid", xlab="UTM X", ylab="UTM Y",
              ylab.right=list(label="Depth (m)", fontsize=18),
              scales=list(y=list(rot=90)))
```

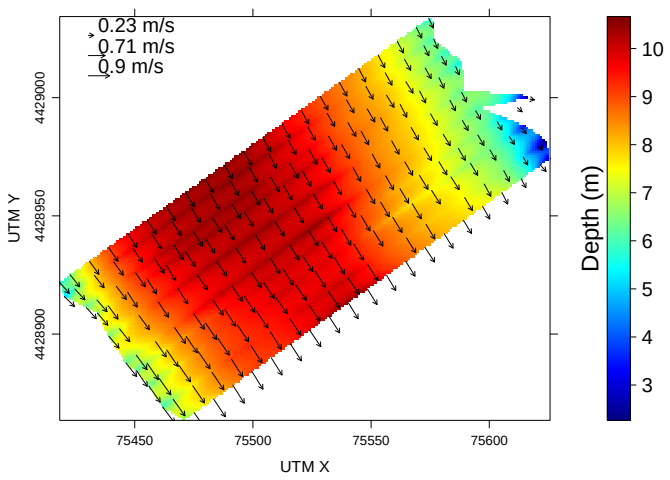
```
## [inverse distance weighted interpolation]
```



Additionally, an interpolated raster can be overlaid by a plot of points or arrows displaying another parameter. The plot below displays depth averaged velocity over a depth raster. The arrow colors (`point.color`) have been set to black to for a clearer display. To reduce the amount of noise when overlaying plots, contour lines can be disabled with `contour=FALSE`.

```
plot.planform(planform.ADCP, DepthAveragedSpeed, depth, point.spacing=10,
              arrow.key=c(75420,4429005), point.color="black", arrow_scale=10,
              cellWidth=1, cellHeight=1, xUnits="m/s", contour=FALSE,
              ylab.right=list(label="Depth (m)", fontsize=18), xlab="UTM X", ylab="UTM Y",
              scales=list(y=list(rot=90)))
```

```
## [inverse distance weighted interpolation]
```



Plot cross-section velocities

Plots of cross-section velocities are available through `plot.xSec`. `plot.xSec` accepts a dataset (`data`) and a parameter to plot (`x`), inverse distance weighting is then used to interpolate the parameter to a grid of a defined cell width (`cellWidth`) and height (`cellHeight`). If `cellWidth` and `cellHeight` are omitted, the default is to use the mean cell width and height of each transect plotted. If only specific transects are to be plotted, they can be defined with `transects`, which accepts a character vector of transect names. If all measured transects are to be plotted, transects can be omitted and `plot.xSec` defaults to plotting all transects in the dataset. The `...` of `plot.xSec` is for additional arguments that are to be passed to `plot.xSec.default`, which is the function that develops the plots. While `plot.xSec.default` can only plot one transect at a time, `plot.xSec` allows multiple transects to be processed, by sub-setting the dataset by transect and iteratively passing the data associated with each transect to `plot.xSec.default`.

Here the zero secondary discharge primary and secondary velocities are plotted. Note that only the primary velocity is supplied to `plot.xSec`, if a primary velocity ("vx", "vp.zsq", "vp.roz", "vpx.roz", or "vsx.roz") is supplied to the function and `arrows=TRUE` secondary velocities from the appropriate rotation method are plotted with arrows. If `x` is not "vx", "vp.zsq", "vp.roz", "vpx.roz", or "vsx.roz" and `arrows=TRUE`, an error will be produced, setting `arrows=FALSE` allows the figure to be plotted.

```
plot.xSec(secondary.ADCP, vp.zsq, "t3",
          header="Primary and Secondary Velocity \nZero Secondary Discharge")
```

```
## Cell Width: 1
```

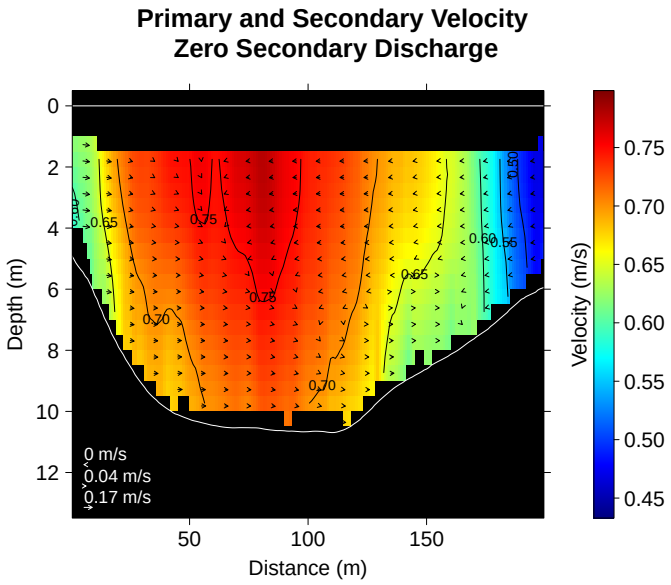
```
## Cell Height: 0.5
```

```
## Arrow X Spacing : 10
```

```
## Arrow Y Spacing : 0.53
```

```
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
```

```
## $t3
```




```
plot.xSec(secondary.ADCP, vp.zsq, "t3", arrows=FALSE,
          header="Primary Velocity \nZero Secondary Discharge")
```

```
## Cell Width: 1
```

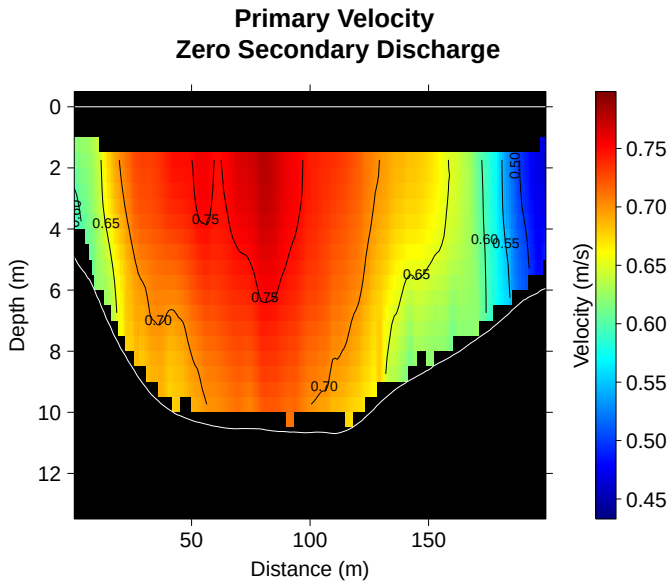
```
## Cell Height: 0.5
```

```
## Arrow X Spacing : 10
```

```
## Arrow Y Spacing : 0.53
```

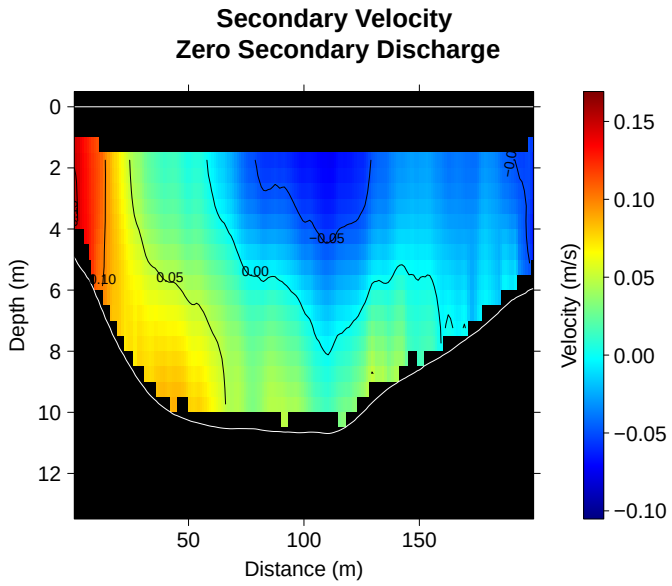
```
## [inverse distance weighted interpolation]
```

```
## $t3
```




```
plot.xSec(secondary.ADCP, vs.zsq,"t3", arrows=FALSE,
          header="Secondary Velocity \nZero Secondary Discharge")
```

```
## Cell Width: 1

## Cell Height: 0.5

## Arrow X Spacing : 10

## Arrow Y Spacing : 0.53

## [inverse distance weighted interpolation]

## $t3
```



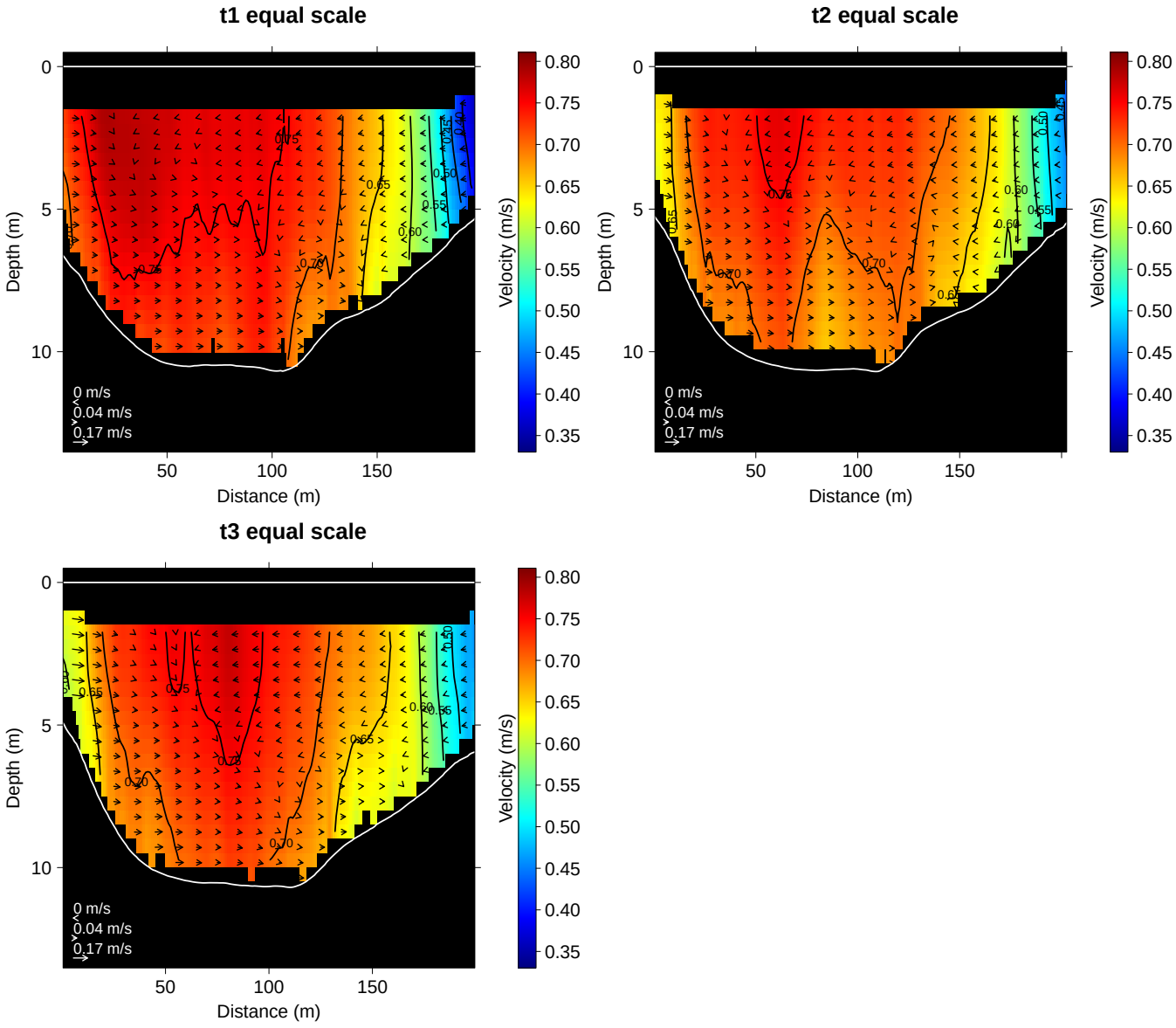
Some other settings worth discussion are the arrow spacing, plot colors, and scale. The x and y spacing to the arrows defaults to 1/20th the transect distance and depth, respectively, arrow spacing can be adjusted with `arrow.x.spacing` and `arrow.y.spacing`. The default colors used are `jet.colors`, however the color scheme can be changed with `col.regions`, by setting it to another color ramp (e.g., heat.colors or terrain.colors). `black.white` and `white.black` are two notable color schemes that allow the plots to be produced in black and white, which allow the color scale to be preserved in second generation copies. `scale.by` allows users to plot transects that are scaled to the range of the parameter within the transect (`scale.by="individual"`) or to the range of the parameter within the dataset (`scale.by="all"`). Setting `scale.by=individual` allows a greater contrast within individual plots, however, `scale.by="all"` allows for easier comparison when multiple transects are plotted. Similar to `planform.plot`, `arrow_scale` controls the length of secondary flow arrows, arrow width can be increased by increasing `arrowwd`, and `arrowCol` controls arrow color.

The plots below show the transects scaled equally.

```
plot.xSec(secondary.ADCP, vp.zsq, scale.by="all", arrow_scale=2, arrowwd=1.5, lwd=2,
          header=paste(transects[i], "equal scale"))
```

```
## Cell Width: 1

## Cell Height: 0.5

## Arrow X Spacing : 9.85

## Arrow Y Spacing : 0.53

## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]

## Cell Width: 1

## Cell Height: 0.49

## Arrow X Spacing : 10.15

## Arrow Y Spacing : 0.54

## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]

## Cell Width: 1

## Cell Height: 0.5

## Arrow X Spacing : 10

## Arrow Y Spacing : 0.53
```

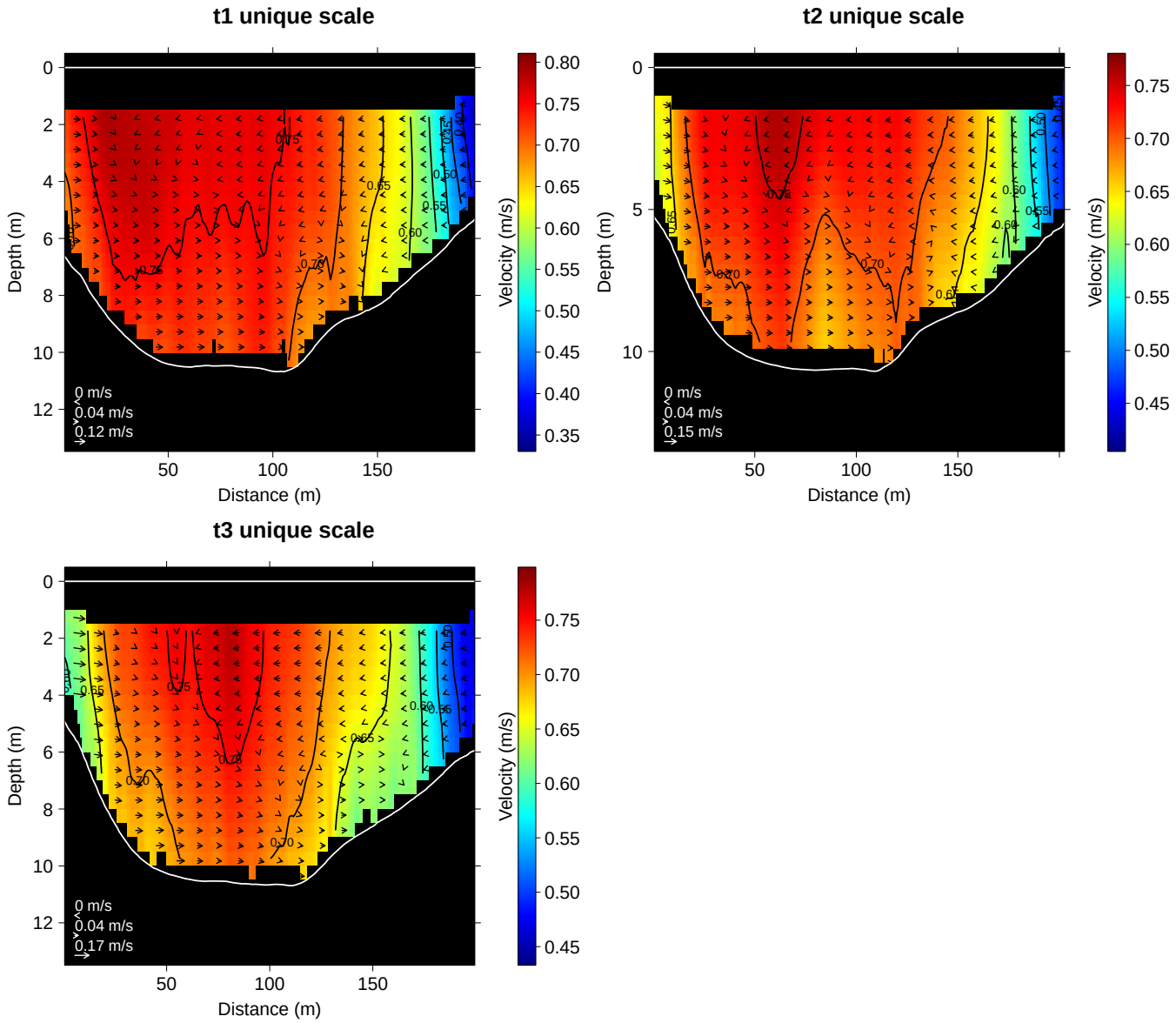
```
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]

## $t1

##
## $t2

##
## $t3
```



And the transects scaled individually.

```
plot.xSec(secondary.ADCP, vp.zsq, scale.by="individual", arrow_scale=2, arrowwd=1.5, lwd=2,
          header=paste(transects[i], "unique scale"))
```

```
## Cell Width: 1

## Cell Height: 0.5

## Arrow X Spacing : 9.85

## Arrow Y Spacing : 0.53

## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]

## Cell Width: 1

## Cell Height: 0.49

## Arrow X Spacing : 10.15

## Arrow Y Spacing : 0.54

## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]

## Cell Width: 1

## Cell Height: 0.5

## Arrow X Spacing : 10

## Arrow Y Spacing : 0.53

## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]

## $t1

##
## $t2

##
## $t3
```

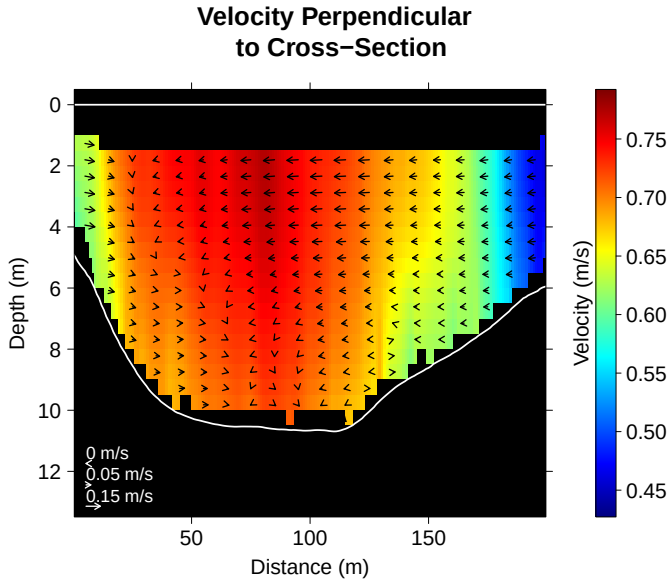


As with `planform.plot`, contours can be disabled by setting `contour=FALSE`. This can help reduce the amount of noise on the plot. Below are the three rotation methods without contours displayed.

```
plot.xSec(secondary.ADCP, vx, "t3", header="Velocity Perpendicular \n to Cross-Section",
          arrow_scale=2,arrowwd=1.5, lwd=2, contour=FALSE)
```

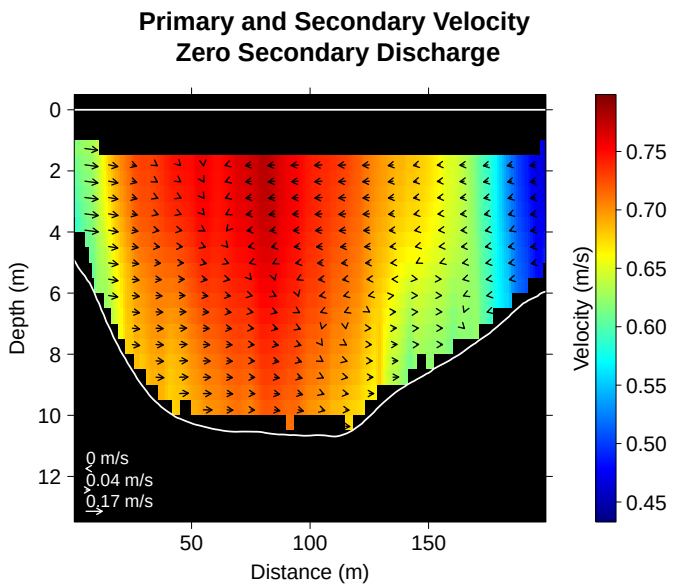
```
## Cell Width: 1

## Cell Height: 0.5

## Arrow X Spacing : 10

## Arrow Y Spacing : 0.53

## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
```

```
## $t3
```



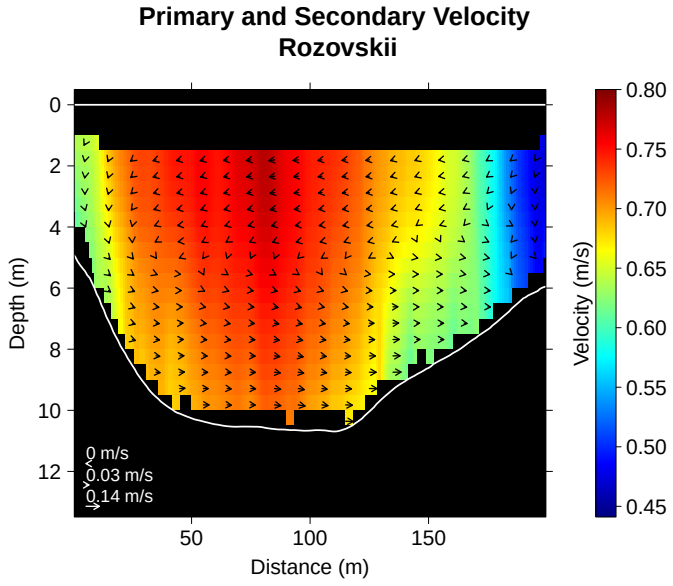

```
plot.xSec(secondary.ADCP, vp.zsq, "t3",
          header="Primary and Secondary Velocity \nZero Secondary Discharge",
          arrow_scale=2, arrowwd=1.5, lwd=2, contour=FALSE)
```

```
## Cell Width: 1

## Cell Height: 0.5

## Arrow X Spacing : 10

## Arrow Y Spacing : 0.53

## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]

## $t3
```




```
plot.xSec(secondary.ADCP, vp.roz,"t3", header="Primary and Secondary Velocity \nRozovskii",
          arrow_scale=2, arrowwd=1.5, lwd=2, contour=FALSE)
```

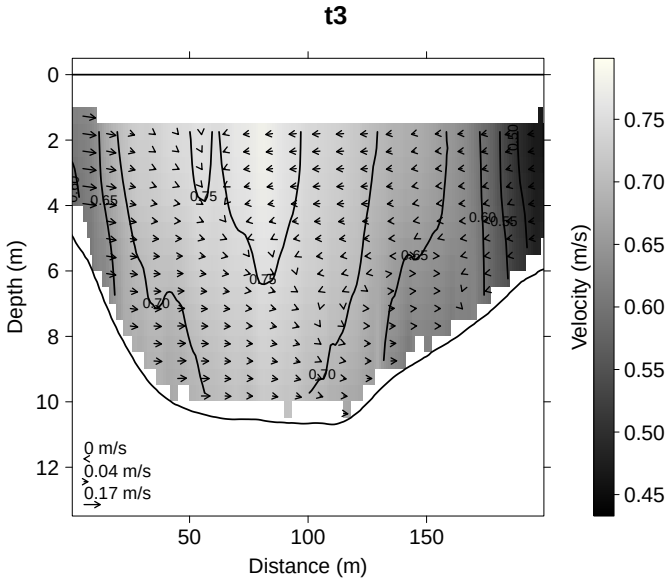
```
## Cell Width: 1

## Cell Height: 0.5

## Arrow X Spacing : 10

## Arrow Y Spacing : 0.53

## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]

## $t3
```



Below is an example plot produced in black and white for publication.

```
plot.xSec(secondary.ADCP, vp.zsq, transects="t3", scale.by="individual", arrow_scale=2,
          arrowwd=1.5, lwd=2, lineCol="black", col.regions=black.white,
          keyCol="black", par.settings=list(layout.widths=list(axis.key.padding=0,
                                                               ylab.right=2),
                                            panel.background=list(col="white")))
```

```
## Cell Width: 1

## Cell Height: 0.5

## Arrow X Spacing : 10

## Arrow Y Spacing : 0.53

## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]

## $t3
```



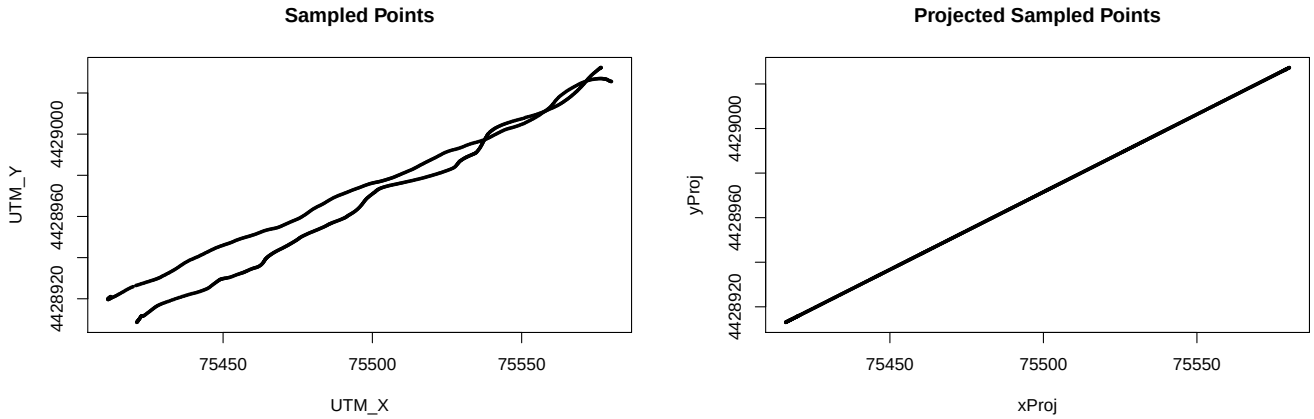
Other functions

This section discusses other functions used by `rivSurveyR`, these functions are called by those described in previous sections and users do not need to call these functions directly to process ADCP data. These functions are described to provided additional detail on how data is handled and processed in `rivSurveyR` and because they may be of use to other applications.

Orthogonal projections

Replicated measurements along a transect rarely overlap, especially when transects are over rough terrain or water bodies. To obtain mean values along a transect line, replicates can be orthogonally projected to the transect line. If the slope and intercept of the transect are known, the x, y coordinates of points sampled along the transect can be projected onto the transect with `ortho.proj`, which determines where the points intercept the line at right angles. If the transect slope and intercept are not know, they can be determined with a linear regression on the x, y coordinates of the sample points.

```
coords <- planform[transectName=="t3", list(UTM_X, UTM_Y),]
linmod <- lm(coords$UTM_Y~coords$UTM_X)
proj.coords <- ortho.proj(coords$UTM_X, coords$UTM_Y, linmod$coefficients[2],
                          linmod$coefficients[1])
plot(coords, main="Sampled Points", pch=20, cex=0.5)
plot(proj.coords, main="Projected Sampled Points", pch=20, cex=0.5)
```

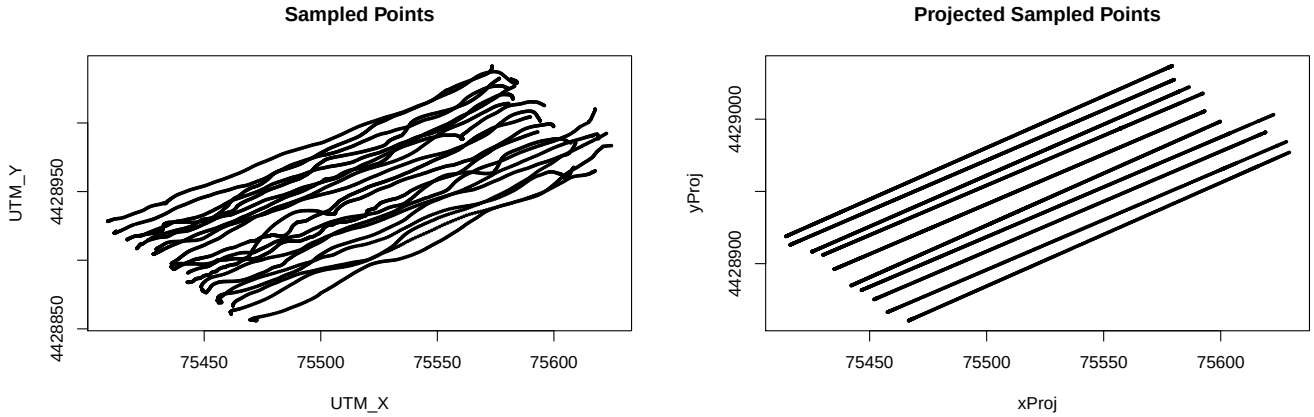


This process is simplified with `project.transect` discussed in the "Projecting transects" section.

Projecting transects

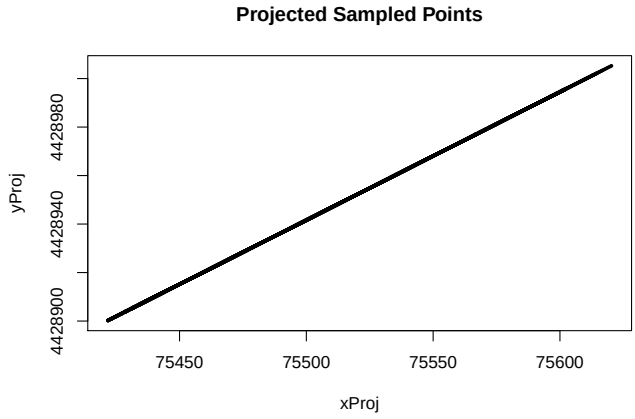
Because only one transect can be processed at a time with `ortho.proj`, projecting transects with `ortho.proj` can be tedious. Additionally, `ortho.proj` requires some pre-existing knowledge on the slope and intercept parameters of a transect, which are likely only know with additional processing. However, `project.transect` allows multiple transects to be projected at once and does not require any pre-existing knowledge of intercept or slope (it does this by automating some of the steps in the example above). `project.transect` does however, require the transect to be identified for every sample point supplied to the function, if the transects are not identified, all sample points are assumed to come from the same transect.

```
tran.coords <- planform[, list(UTM_X, UTM_Y, transectName),]
proj.tran <- project.transect(planform$UTM_X, planform$UTM_Y, planform$transectName)
plot(UTM_Y~UTM_X, tran.coords, main="Sampled Points", pch=20, cex=0.5)
plot(yProj~xProj, proj.tran, main="Projected Sampled Points", pch=20, cex=0.5)
```



If transect names are not supplied, only one transect is projected. This will only yield problems if multiple transects are to be processed at once. In the figure below, the ten transects have clearly been projected incorrectly as one transect.

```
tran.coords <- planform[, list(UTM_X, UTM_Y),]
proj.tran <- project.transect(planform$UTM_X, planform$UTM_Y)
plot(yProj~xProj, proj.tran, main="Projected Sampled Points", pch=20, cex=0.5)
```



Determine the heading of a transect

The compass heading of a transect relative to flow (from river right to river left) can be determined with `transectHeading`. This function takes x and y coordinates of points along a transect and determines the heading based on the distance travelled in the x and y directions. The flow heading is then used to determine the river right and left ends of the transect and the heading is adjusted to reflect a transect progressing from river right to left. Flow heading can either be provided (if known) or velocities and depth at each sample point can be supplied to `transectHeading` in lieu of flow heading. When depth and velocities are supplied to `transectHeading`, velocity vectors are weighted by depth (to account for area sampled) and the mean cross-section flow direction is calculated. This procedure assumes the depth- or layer-averaged velocities of an ensemble are supplied to `transectHeading`. If velocities of individual cells are supplied, supplying cell height instead of water depth is more appropriate.

```
#Determine transect heading from planform data
planform.t3 <- planform.ADCP[transectName=="t3",,]
transectHeading(planform.t3$UTM_X_Proj, planform.t3$UTM_Y_Proj, planform.t3$Mean.Vel.E,
                planform.t3$Mean.Vel.N, planform.t3$depth)
```

```
## [1] 55.1364
```

```
#Determine transect heading from cross-section data
secondary.t3 <- secondary.ADCP[transectName=="t3",,]
transectHeading(secondary.t3$UTM_X_Proj, secondary.t3$UTM_Y_Proj, secondary.t3$Mean.Vel.E,
                secondary.t3$Mean.Vel.N, secondary.t3$cellHeight)
```

```
## [1] 55.13583
```

Determining flow heading

The compass heading of two velocity vectors can be determined with `heading`. If the velocity vectors are of velocities to east and north, `heading` will return compass heading in degrees, of those vectors.

```
easting <- c(0,1,1,1,0,-1,-1,-1)
northing <- c(1,1,0,-1,-1,-1,0,1)
heading(easting, northing)
```

```
## [1]   0  45  90 135 180 225 270 315
```

Although `heading` is set-up to output compass heading, the arithmetic heading of two vectors can also be calculated by supplying the east and north or x and y vectors in reverse order.

```
heading(northing, easting)
```

```
## [1]  90  45   0 315 270 225 180 135
```

Calculating average velocity within a defined layer of the water column

The integrated mean of y within the defined bounds of x can be calculated with `layerAvg`. `layerAvg` requires a vector of y values, a sorted vector of x values, and the upper and lower bounds of x, which y is integrated over.

```
x <- c(1:10)
y <- x^2
#Determine the mean value of y between 2 and 8.
layerAvg(x,y,2,8)
```

```
## [1] 28.16667
```

```
#Application with ADCP data
#Determine the mean east velocity between the surface (0m) and 2 m depth
#Note that only one sample/ensemble can be processed at a time
ensembleOne <- secondary.t3[tDist==0,,]
ensembleOne <- ensembleOne[order(cellDepth),,]
layerAvg(ensembleOne$cellDepth, ensembleOne$Vel.E, 0, 2)
```

```
## [1] 0.4537948
```

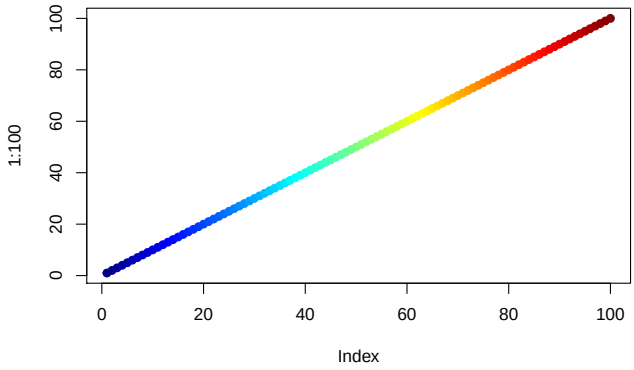
Make a color gradient

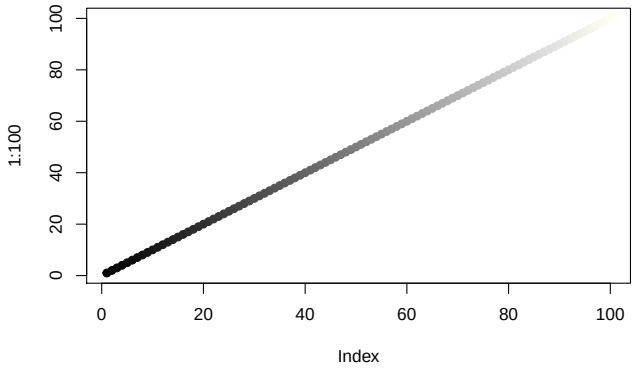
`jet.colors` is borrowed from the examples in `?colorRampPalette` and creates a blue to red color gradient. This is the default color gradient used by `plot.xSec` and `plot.planform`. See `colorRampPalette` for additional details. Black and white color gradients are also available with the `black.white` and `white.black` functions, which create a black to white and white to black gradient, respectively.

```
jet.colors(10)
```

```
##  [1] "#00007F" "#0000F0" "#0062FF" "#00D4FF" "#46FFB7" "#B7FF46" "#FFD400"
##  [8] "#FF6200" "#F00000" "#7F0000"
```

```
plot(1:100, pch=19, col=jet.colors(100))
plot(1:100, pch=19, col=black.white(100))
```





Package Info

U.S. Geological Survey (USGS) Computer Program rivSurveyR version 2.1. Written by Jason L. Fischer, USGS - Great Lakes Science Center, Ann Arbor, Michigan, USA. Written in programming language R (R Core Team, 2014, www.R-project.org), version 3.3.0 (2016-05-03). Run on a PC with Intel(R) Core(TM) i5 CPU, 2.40 GHz processor, 8.0 GB RAM, and Microsoft Windows 7 Enterprise operating system 2009 Service Pack 1. Source code is available from Jason L. Fischer on GitHub, jfischer@usgs.gov.

Disclaimer

Although this program has been used by the USGS, no warranty, expressed or implied, is made by the USGS or the United States Government as to the accuracy and functioning of the program and related program material nor shall the fact of distribution constitute any such warranty, and no responsibility is assumed by the USGS in connection therewith. Use of trade names is strictly for informative purposes and does not imply endorsement by USGS.

References

Cheng-Lung, 1991. "Unified Theory on Power Laws for Flow Resistance." Journal of Hydraulic Engineering, Vol. 117, No. 3, March 1991, 371-389.

Czuba, J.A., Best, J.L., Oberg, K.A., Parsons, D.R., Jackson, P.R., Garcia, M.H., Ashmore, P., 2011. Bed morphology, flow structure, and sediment transport at the outlet of Lake Huron and in the upper St. Clair River. Journal of Great Lakes Research 37, 480-493.

Garcia, M.H., 2008. Sediment transport and morphodynamics, chap. 2. In: Garcia, M.H. (Ed.), Sedimentation Engineering: Processes, Measurements, Modeling, and Practice No. 110. American Society of Civil Engineers, Reston, Virginia, pp. 21-163.

Keulegan, G.H., 1938. Laws of turbulent flow in open channels. Journal of Research of the National Bureau Standards 21, 707-741.

Lane, S.N., Bradbrook, K.F., Richards, K.S., Biron, P.M., Roy, A.G., 2000. Secondary circulation cells in river channel confluences: Measurement artefacts or coherent flow structures? Hydrological Processes 14, 2047-2071.

Mueller, D.S., Wagner, C.R., Rehmel, M.S., Oberg, K.A„ Rainville, F., 2013. Measuring discharge with acoustic Doppler current profilers from a moving boat (ver. 2.0, December 2013): U.S. Geological Survey Techniques and Methods, book 3, chap. A22, 95p.

Parsons, D.R., Jackson, P.R., Czuba, J.A., Engel, F.L., Rhoads, B.L., Oberg, K.A., Best, J.L., Mueller, D.S., Johnson, K.K., Riley, J.D., 2013. Velocity mapping toolbox (VMT): A processing and visualization suite for moving-vessel ADCP measurements. Earth Surface Processes and Landforms 38, 1244-1260.

Rhoads, B.L., Kenworthy, S.T., 1998. Time-averaged flow structure in the central region of a stream confluence. Earth Surface Processes and Landforms 23, 171-191.

Simpson, M.R., Oltmann, R.N., 1990. "An Acoustic Doppler Discharge Measurement System." Proceedings of the 1990 National Conference on Hydraulic Engineering, Vol. 2, 903-908.

Szupiany, R.N., Amsler, M.L., Best, J.L., Parsons, D.R., 2007. Comparison of fixed- and moving-vessel flow measurements with aDp in large a river. Journal of Hydraulic Engineering 133, 1299-1309.