Supercooling in Rivers: Field Measurements and Surface Energy Budget Analysis

by

Sean Ryan Boyd

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Water Resources Engineering

Department of Civil and Environmental Engineering
University of Alberta

Abstract

The process of supercooling, where turbulent water is cooled below freezing while remaining a liquid, is a necessary condition for the formation of frazil and anchor ice in northern rivers during the late fall and winter. While the phenomenon has been often investigated in laboratory, relatively few studies report field observations. This thesis reports on 696 supercooling events recorded on three regulated Alberta rivers from 2015 – 2020. These supercooling events were analyzed for properties including peak supercooling temperature and duration, and analyzed in order to increase our understanding of the behaviour of supercooling in rivers. The median peak supercooling temperature across all events was -0.013ºC and a median cooling rate of -2.5×10^{-4} ºC/minute. Though the median event duration was 4.8 hours, events with durations from 2 to 14 days were regularly observed, exceeding the longest events previously reported in literature of ~50 hours. Preliminary analysis of longitudinal gradient of supercooling events in rivers showed the potential impact of both dam regulation and urbanization on spatial distribution of supercooling. Comparison between the behaviour of supercooling on left and right bank in one river showed differences in frequency and duration of supercooling events cross channel, which may have additional implications for the spatial distribution of frazil ice production.

Since supercooling events are driven by climate factors of the late fall and winter, understanding the heat fluxes at the water surface and how they drive supercooling is crucial to improving understanding of supercooling. Supercooling data collected on two rivers for the 2016-2017 season was analyzed along side local weather data to provide insight into the relationship between supercooling events and the local surface energy budget. From the calculated shortwave, longwave, sensible, and evaporative heat fluxes, it was found that the shortwave and

longwave were the dominant heat fluxes during supercooling events, with 97.4 % of events predominantly warmed by shortwave radiation, and 80.0 % of events predominantly cooled by longwave radiation. Sensible heat flux was found to primarily be a secondary heat flux unless air temperature dropped significantly colder that the season average (average air temperature -14.8 ºC during supercooling compared to the season average of -8.99 ºC during supercooling). Evaporative heat flux tended to be negligible unless air temperature was positive along side high wind speeds and low magnitude radiative heat fluxes. While no linear correlation was found between supercooling parameters and event averaged heat flux components, it is clear from the analysis that the start and end of supercooling events are notably impacted by the diurnal cycling of shortwave radiation.

Preface

This thesis research is an original work by Sean Boyd under the supervision of Dr. Mark Loewen from the Department of Civil and Environmental Engineering at the University of Alberta and Dr. Tadros Ghobrial from the Department of Civil and Water Engineering at Laval University.

Chapter 2 of this thesis was published as:

Boyd, S., Ghobrial, T., Loewen, M., Jasek, M., Evans J., 2022. A study of supercooling in rivers. *Cold Regions Science and Technology*, 194(2022), 1034-1055, doi: https://doi.org/10.1016/j.coldregions.2021.103455

I was responsible for one season of data collection, as well as the development and implementation of the code used for analysis of the collected data. I was also responsible for the analysis of the final results as well as the manuscript composition. Loewen, M. was responsible for the supervision and administrative oversight, and along with Ghobrial, T. oversaw the concept development of the project, planning of data collection, and manuscript review and edits. Jasek M. and Evans, J. were instrumental in data collection on the Peace River, as well as providing additional manuscript edits. Since this study is the cumulation of multiple years of data collection by the University of Alberta River Ice Research Group, I would like to acknowledge the work in data collection by Kerry Paslawski, Vincent McFarlane, Hayden Kalke, Chris Schneck and Rhodri Howley, as well as Perry Fedun for his technical assistance for preparation for field deployment.

For Chapter 3, I was responsible for developing the methodology and code for analysis of the collected data and manuscript composition. Ghobrial, T. and Loewen, M. were both supervisory authors as well as contributing to the planning of data collection, concept development of the analysis process and manuscript edits. Since this data was collected in 2016-2017, I would like to again acknowledge the data collection done by Kerry Paslawski, Vincent McFarlane, Hayden Kalke, Chris Schneck, as well as Perry Fedun for his technical assistance for preparation for field deployment.

This thesis would not have been possible to pursue to completion without the support of family.
Thank you for all your support in a myriad of ways

Acknowledgements

The author would like to acknowledge and thank both Dr. Mark Loewen and Dr. Tadros Ghobrial for their assistance and support throughout all aspects of this thesis. Thank you for all your time and effort in providing insights and feedback.

Thank you as well to Dr. Yutong She for agreeing to be on the examining committee, and to Dr. Wenming Zhang for chairing the exam

This research was supported by the Natural Sciences and Engineering Research Council of Canada and is gratefully acknowledged.

Table of Contents

List of Table

List of Figures

Chapter 1 : Introduction

1.1 Overview

Rivers are a vital community resource, as they provide a water supply as well as serve as a critical component of sanitation, power generation, and transportation. For northern communities, the dynamics of river ice processes heavily impact all these sectors from late fall to early spring. Daly (1994) described the evolution of a river ice regime, and how supercooling (i.e., water temperatures cooled below 0ºC) is a critical component of the process. Under freezing conditions, water begins supercooling and entrained seed crystals in the water column act as sites for secondary nucleation (Daly 1994). The forming of frazil ice increases the available nucleation sites, causing a 'bloom' in frazil ice production. Frazil ice crystals in supercooled water are strongly adhesive, freezing to other ice crystals, as well any submerged surfaces such as the river bed and banks and any infrastructure (Arden and Wigle 1973). Flocculation of frazil in the water column results in growing frazil slush balls that eventually become buoyant enough to float to the surface to form frazil pans or adhere to the bed, resulting in anchor ice (Daly 1994). The anchor ice grows in size through both accumulation of frazil and in-situ growth until thermal warming or mechanical release from sufficient buoyancy frees the anchor ice from the bed. The surface ice masses are then transported downstream, freezing together into larger frazil rafts. Eventually, the leading edge is halted by grounding on the bed or a constriction from the banks or border ice (Daly 1994). This forms the starting point for a consolidating ice cover to propagate upstream, insulating the water from the cooling conditions. Breaks in this cover may still develop from the introduction of warm water or due to high flows (rapids). Fluctuations in flow such as those from tidal surges and dam hydropeaking are also known to break up ice covers and maintain open water conditions throughout the winter (Maheu et al. 2014; Richard and Morse 2008). The break-up of the ice cover in the spring re-opens the water surface to the air, and can result in a second period of supercooling under sufficiently cool weather conditions. The formation of suspended frazil during freeze-up impacts water intakes (Richard and Morse 2008) while anchor ice development can impact power generation (Arden and Wigle 1973).

Since the formation of ice is exothermic, the water temperature is a balance between ice production and the net cooling flux. Laboratory experiments (e.g., Carstens 1966) show that under a constant

heat flux, the water temperature will drop to a minimum temperature before rising to an equilibrium temperature. At this stage, the latent heat of frazil ice production matches the cooling surface heat flux, resulting in a constant water temperature referred to as the residual temperature. In more dynamic conditions in the field, the net heat flux typically varies and supercooling events will tend to end when the net heat flux becomes positive (Ashton 1986). The lowest temperatures observed during field supercooling were reported by Matousek (1992) at -0.18°C at the water surface shortly before skim ice was observed. McFarlane et al. (2019) described an event that reached a peak supercooling temperature of -0.145°C. Literature reported duration of supercooling events to be on average less than 8 hours, though individual events were reported to last around two days (Richard and Morse 2008; Nafziger et al. 2013). The cooling rates at the start of supercooling events measured in laboratory studies (-1.29×10^{-3} to -5.02×10^{-2} ◦C/minute according to Daly (1994)) were estimated to be approximately an order of a magnitude larger than the cooling rates measured in natural streams (Osterkamp (1978)).

Ashton (1986) reported up to eleven modes of heat transfer between the water column and its boundaries, though most studies only quantify the most significant of these heat fluxes for the specific analysis. Previous literature have found that surface heat fluxes account for the majority of the energy budget in most rivers (Evans et al. 1998; Hannah et al. 2004). However, groundwater sourced streams can have reduced sensitivity to weather variation due to underground water being isolated from the weather conditions (Brown et al. 2006). At the surface, the shortwave and longwave heat fluxes were found to be the most important components for a river's winter energy budget (Evans et al. 1998; Webb and Zhang 2004; McFarlane and Clark 2021). The balance of these radiative heat fluxes results in a net loss in the winter due to reduced solar radiation but continually outgoing longwave radiation. The sensible heat flux is the next most prominent heat flux, and can be a significant positive or negative heat flux depending on the local weather conditions (Hannah et al. 2004; Richard et al. 2015). Even through reduced daylight hours, there is still a significant variability between day and night heat fluxes with Richard et al. (2015) noting differences of ~200 W/m^2.

1.2 Research objectives and methodology

The first objective of this research is to investigate the behaviour of supercooling events in rivers, through analysis of multiple seasons of events across different rivers. Previous studies have taken field measurements for duration and minimum temperatures of supercooling, but did not have sufficiently large data sets to determine trends in behaviour of supercooling in rivers. Chapter 2 presents an analysis of water temperature time series over multiple winters between 2015 and 2020 on three Alberta rivers: the North Saskatchewan River, Peace River and Kananaskis River. The analysis detailed in Chapter 2 is one of the first to look at large multiple winter data set and attempt to develop means to determine trends in how supercooling events occur and behave. This includes the start and end time of day, duration, peak supercooling temperature, principal supercooling duration, principal supercooling average cooling rate and cumulative degree minutes of supercooling. The analysis would highlight the differences between the supercooling behavior in different rivers as well as the spatial variation of supercooling behavior in the lateral and longitudinal directions in the rivers.

The second objective of this research is to improve our understanding of the relationship between water temperature and the surface energy budget during supercooling events. Chapter 3 looks at the 2016-2017 sub-set of supercooling events observed on the North Saskatchewan and Peace Rivers alongside local weather data to estimate the local energy budget during supercooling events. A better understanding of how changes in surface heat fluxes impacts supercooling events will inform modeling and forecasting of the development of a river's ice regime, and inform planning and forecasting around supercooling and ice production.

Chapter 2 : A study of supercooling in rivers[1]

2.1 Introduction

Supercooling can occur in rivers when the turbulent water surface is exposed to weather conditions that exert a strong, persistent negative heat flux. These conditions typically begin in late fall during freeze-up prior to the formation of a continuous ice cover and early spring as the cover starts to break-up. In supercooled turbulent water, frazil ice crystals first form near the water surface and then propagate deeper into the water column through vertical mixing (Arden and Wigle 1973). Frazil crystals surrounded by supercooled water are called 'active' and readily adhere to each other forming frazil flocs or adhere to the river bed to form anchor ice (Daly 1994). As the frazil ice concentration grows, frazil flocs increase in size and float to the surface once they are large enough for buoyant forces to overcome turbulence. As the slush formed by the gathering flocs moves downstream, it freezes together into frazil pans. If the downstream flow of frazil pans is halted due to grounding or a constriction, a continuous ice cover is formed and propagates upstream as more pans arrive. The resulting ice cover insulates the water from any cooling heat fluxes, preventing supercooling from occurring unless there is open water upstream of the ice cover (e.g., rapids or open leads). Break-up of the ice cover in the early spring may cause a second period of supercooling to occur depending on the local weather conditions. Understanding the behavior of supercooling in rivers is of great importance due to the impact frazil ice and anchor ice formation can have on winter hydraulics (e.g., Jasek et al. 2015), water intakes (e.g., Richard and Morse 2008) and hydropower generation (e.g., Arden and Wigle 1973). In addition, ice formation processes have a significant impact on river ecosystems (Prowse 2001). For example, it has been observed that long periods of supercooling may lead to large mortality rates in benthic organisms buried in the river bed, even without exposure to frazil ice (Prowse 2001).

[1] This chapter has been published in Cold Region and Technologies as:
Boyd, S., Ghobrial, T., Loewen, M., Jasek, M., Evans J., 2022. A study of supercooling in rivers. Cold Regions Science and Technology, 194(2022), 1034-1055, doi: https://doi.org/10.1016/j.coldregions.2021.103455

Several previous laboratory studies have reported measurements of cooling rates and peak supercooling temperatures. In these experiments supercooling was typically generated by exposing the water surface in a stirred tank or open channel flume to a constant sub-zero air temperature (i.e., a constant heat flux from the water to the air). In a tank or channel exposed to a constant upward heat flux, the water temperature (T_w) will vary with time as shown in Figure 2-1. Initially, T_w decreases at a constant rate until frazil ice crystals start to form. The production and growth of frazil ice releases latent heat into the water, gradually reducing the cooling rate. At the time when the heat generated by frazil ice production balances with the surface heat flux, the lowest supercooling temperature, referred to as "peak supercooling", is reached. After this point, the heat generated by frazil ice production exceeds the surface heat flux, so T_w increases, then ice production begins to decrease and when the latent heat release again equals the surface heat flux a constant residual temperature is reached. The period of time between the start of supercooling until when the residual temperature is reached is called the principal supercooling stage and this is followed by the residual supercooling stage (Ye et al. 2004). The shape of the time series for this type of event shown in Figure 2-1 (i.e., constant heat flux) was defined as a 'classical' supercooling event by Kalke et al. (2019).

Carstens (1966) studied the behaviour of supercooling in a laboratory racetrack flume and found that higher cooling rates lead to lower peak supercooling and residual temperatures. They observed peak supercooling temperatures that varied from -0.03 to -0.18°C. Michel (1967) used an experimental flume to establish a threshold for frazil ice nucleation temperature around -0.05ºC. Michel (1971) reported that a cooling rate larger than 0.02 ºC/minute is required for a supercooling event to establish a residual temperature. Matousek (1992) used both laboratory and field experiments to show that skim ice begins to form at water surface temperatures of -0.18ºC, even when the bulk water temperatures are above freezing. Using a counter rotating laboratory flume to study frazil ice formation, Ye et al. (2004) reported average cooling rates of -0.002 to -0.0083 ºC/minute and peak supercooling temperatures from -0.028 to -0.052ºC. Ghobrial et al. (2012) and Schneck et al. (2019) measured similar ranges of cooling rates of -0.006 to -0.012 ºC/minute and -0.008 to -0.013 ºC/minute, respectively. McFarlane et al. (2015) measured frazil ice particle sizes in a stirred tank and observed peak supercooling ranging from -0.072 to -0.093°C.

The earliest reported measurements of supercooling in the field go back over a century when Barnes (1908), wrote that the supercooling water only reached "a few thousandths of a degree" below freezing. Based on thousands of measurements, Altberg (1936) reported that supercooling temperatures in rivers could reach -0.05°C. Arden and Wigle (1973) conducted studies on the upper Niagara River to better understand the conditions that lead to the development of surface ice and reported supercooling temperatures as low as -0.07°C. This study also observed active frazil in the upper layers of the flow at water temperatures of approximately -0.02 to -0.03°C. Osterkamp (1978) reports nucleation temperatures $\geq$ -0.01°C on small streams, hypothesizing that the heat and mass exchange at the water surface may be key in the observed nucleation temperatures. Osterkamp (1978) also observed that the cooling rates observed in laboratory settings tend to be an order of magnitude greater than those measured in rivers. Daly (1994) compiled heat loss rates of supercooling events recorded in the literature and found values from $9x10^{-5}$ to $3.5x10^{-3}$ J/s·cm^3, which translates to cooling rates of $-1.29x10^{-3}$ to $-5.02x10^{-2}$ °C/minute.

A field study on the tidal St. Lawrence River near Quebec City, Canada measured peak supercooling between -0.01°C and -0.06°C (Richard and Morse 2008). While not quantifying the duration of events, Richard and Morse (2008) state that the majority of active frazil periods lasted less than 8 hours, with 5 events lasting longer than two tide cycles (>~50 hours). Water temperature measurements at six sites in five rivers in Newfoundland and New Brunswick were used by Nafziger et al. (2013) to investigate the behaviour of supercooling during freeze-up and break-up. They found that freeze-up events tended to reach a residual temperature indicating that the latent heat released by frazil production was balanced by heat loss due to surface cooling, while break-up events never reached a residual temperature and ended abruptly. Peak supercooling did not drop below -0.09°C and the average duration of supercooling events at each site ranged from 2.2 to 8.5 hours, with a maximum of 42.7 hours. While studying the freeze-up energy budget on the Dauphin River in Manitoba, McFarlane and Clark (2021) observed six overnight supercooling events. The maximum peak supercooling temperature observed was -0.064°C and the durations of events ranged from 1 to 18 hours.

McFarlane et al. (2017) collected water temperature data on the Kananaskis and North Saskatchewan Rivers and reported peak supercooling temperatures from -0.026 to -0.061°C, and principal supercooling periods of 1 to 2.36 hours. They compared their field observations to laboratory measurements and found that laboratory supercooling reached lower temperatures (-0.072 °C to -0.093°C) and established an equilibrium much faster (12 to 15 minutes from the event start) than in the field. In a subsequent study on the North Saskatchewan River, McFarlane et al. (2019) observed a supercooling event with a peak supercooling temperature of -0.145°C. This event was also notable, since no detectable suspended frazil ice was generated but rapid anchor and skim ice formation was observed. Howley et al. (2019) used River1D to model river ice processes on the North Saskatchewan River and related a novel parameter, the degree minutes of supercooling (*DMS*), to frazil ice concentration. *DMS* was defined as the integrated area of a supercooling event temperature time series (i.e., the area between 0°C and the supercooling curve). Similar to how cumulative degree days of freezing (CDDF) is used to quantify the severity of periods of freezing air temperatures, Howley et al. (2019) suggested *DMS* as a means to quantify the intensity of a supercooling event and relate it to the peak frazil production. Preliminary tests of this method with a data set with a maximum *DMS* of ~250 °C·minutes suggests that it could be viable, but additional field measurements would be required to confirm its applicability (Howley et al. 2019).

It is evident that previous field studies of supercooling have provided some valuable data regarding peak supercooling and some measurements of the duration of events. However, the data is sparse and additional field measurements are needed to increase our knowledge of supercooling behaviour in rivers and thus improve our understanding of frazil ice formation. To address this need, high resolution winter water temperature time series measurements were collected in three Canadian rivers from 2015 to 2020. Preliminary analysis and results based on these field measurements were presented in Kalke et al. (2019) and Boyd et al. (2020).

2.2 Study area

Supercooling measurements were collected in three regulated rivers in Alberta, Canada, namely: the Kananaskis, North Saskatchewan and Peace Rivers as shown in Figure 2-2. The characteristics of each study reach are listed in Table 2.1. The Kananaskis River (KR) is the smallest river in the study, with an annual average flow rate and depth of approximately 10 m^3/s and 0.60 m, respectively (Table 2.1). Three measurement sites were used on the study reach and were located 10, 15, and 20 km downstream of the Pocaterra Dam (see Figure 2-2b). Daily hydropeaking causes the discharge to vary between 2 – 20 m^3/s which typically prevents the formation of a permanent ice cover at the measurement sites. Previous field studies in this reach have shown that these conditions allow supercooling events to occur for most of the winter season (McFarlane et al. 2017).

The North Saskatchewan River (NSR) is the intermediate sized river in this study with an annual average flow rate of 220 m^3/s and an average depth of 1.40 m within the study reach (Table 2-1). Seven measurement sites were used on the NSR over a ~104 km long reach. The most upstream site at Genesee, Alberta is 48 km upstream of the City of Edmonton limits, and approximately 360 km and 195 km downstream of the Bighorn and Brazeau Dams, respectively. The remainder of the sites were located within the City of Edmonton limits as shown in Figure 2-2c. Dam regulation causes daily water level fluctuations between 0.3 and 0.4 m during freeze-up in the city. Freeze-up in this reach typically starts in early November and ends late November to early December.

The Peace River (PR) is the largest river in this study with an annual average flow rate of ~1,600 m^3/s and an average depth of flow of 2.6 m (Table 2.1). As shown in Figure 2-2a the dams regulating the flow are approximately 300 km upstream of the study reach. Nine measurement sites were used on this reach and they are labeled in Figure 2-2d based on their distance downstream of the W.A.C. Bennett Dam. The baseflow regulation from the upstream dams introduces significant volumes of warm water to the PR, altering the river ice regime downstream. The ice front is initially formed in early December ~800 km downstream and then slowly advances upstream and by early March is typically ~100 km downstream of the dams before beginning its retreat (Jasek et al. 2011). Supercooling events can occur at any given measurement site up until shortly after the ice front passes that location and a continuous stable ice cover is formed.

2.3 Methodology

Water temperature data was sampled at 1-minute intervals (or 5-minute intervals in the case of the PR sensors deployed from 2018-2020) using RBR Solo T temperature loggers (±0.002ºC) housed in protective metal casings and anchored to the riverbed using metal pins as shown in Figure 2-3. These casings were used on the KR and NSR, where the sensors were deployed by wading to a depth of ~0.75 m. In the PR, the loggers were installed in circular casings, bolted to flat steel bars, that were deployed from a boat and cabled to an anchoring point (e.g., a large tree) on the riverbank. Table 2.2 presents a summary of the number of measurements sites during every year of measurements on each river. The loggers were deployed in the fall prior to freeze-up and were retrieved the following year after the end of the break-up. One season of measurements was collected on the KR (2016-2017) and four seasons of measurements (between 2015 and 2020) on the other two rivers (see Table 2.2). The number of measurement sites varied from 1 to 7 sites due to logistical constraints and field conditions.

In Figure 2-4, a typical supercooling event is plotted illustrating graphically the parameters that were used to characterize supercooling events. These parameters include the following: event start t_s and end t_e times defined as the interpolated times when the temperature reached 0°C; event duration D, the time between t_s and t_e; peak supercooling temperature T_P, the minimum temperature reached during an event; the principal supercooling duration D_P, the time between t_s and T_P; the cumulative degree minutes of supercooling $CDMS$, the intensity of the supercooling event as introduced by Howley et al. (2019); and the principal supercooling average cooling rate CR_P calculated as the ratio between T_P and D_P. Also, it should be noted that the use of the term 'principal supercooling' for both T_P and CR_P differs from its definition in previous studies (e.g., Ye et al., 2004; McFarlane et al., 2019). In those studies, the principal supercooling period was defined for a classical supercooling event as the time between the start of an event and the point when the residual temperature was reached. The change in definition proposed in this study was motivated by the fact that supercooling events observed in rivers often do not reach an equilibrium residual temperature (Kalke et al., 2019), which makes D_P undefined for those events.

A MATLAB program was developed to identify and catalogue supercooling events in the collected time series data. A supercooling event was defined as a continuous series of negative water

temperatures that lasted for 10 minutes or more. The minimum duration requirement was implemented to screen out events that would have an insignificant impact on river ice processes. The accuracy of the RBR Solo T loggers is ± 0.002ºC and therefore events that were detected within this range -0.002ºC < T_P < 0ºC were discarded. In addition, events with peak supercooling < -0.2ºC, as well as events occurring within an hour before or after them, were also discarded because these events were always associated with anomalous time series behaviour. This cut off value was established as a relatively conservative threshold that is below the range of observed peak supercooling temperatures in previous studies (Kalke et al., 2019; McFarlane et al, 2019).

Lastly, to facilitate investigation of supercooling behavior associated with different river ice processes freeze-up and break-up periods were defined. A freeze-up period is defined as the period when frazil ice formation processes are dominant. In typical years, this period extends from when the first supercooling event is observed, to the time when an ice cover forms over the study reach. Similarly, the break-up period is defined as the period from when the ice cover starts to break-up in the study reach, to when the last supercooling event was observed. Freeze-up period end dates and break-up period start dates were estimated depending on the availability of field data on ice front location. For all NSR sites, images from the University of Alberta Earth and Atmospheric Sciences (EAS) camera located near the Emily Murphy site were used for defining freeze-up and break-up periods. For the PR, B.C. Hydro provided ice front location data for all four seasons. It is important to note that during the 2016-2017 season, the ice cover did not reach the study reach on the PR due to milder winter air temperatures. Extreme daily hydropeaking prevented the formation of an ice cover in the KR study reach during the 2016-2017 season. Therefore, for the 2016-2017 season on the KR and PR, the freeze-up period extended throughout the entire winter since the presence of open water in the two study reaches allowed frazil ice formation processes to occur throughout the winter.

2.4 Results

A total of 34 continuous winter water temperature time series were successfully collected in the three rivers between 2015 and 2020 (see Table 2.2). A total of 39 detected supercooling events were discarded either because they had T_P lower than the cut-off threshold of -0.2°C or because they occurred within one hour of an anomalous event. It is important to note that 39 events is only ~5% of the total number of detected events. Some possible causes of these anomalous events

include anchor ice formation on the sensor, lifting of the sensor to the water surface via anchor ice release, and incorporation of the sensor into thermal ice.

Table 2.3 presents a synopsis of supercooling events per river for each deployment season including the date of the first and the last supercooling events, the duration of both the freeze-up and break-up periods, as well as the number of events recorded during each period. Table 2.3 also presents the seasonal cumulative degree minutes of supercooling (*SCDMS*) for each freeze-up and break-up period, defined as the summation of the total *CDMS* observed on the river during a given freeze-up and break-up period divided by the number of measurement sites that recorded supercooling events. During the 2016-2017 season, a stable ice cover did not form at the measurement sites on the KR and PR, and therefore the freeze-up period lasted for 137 and 133 days, respectively. For the rest of the seasons, the freeze-up period duration ranged between 8 - 46 days and 15 - 42 days on the NSR and the PR, respectively. Similarly, the break-up period ranged between 5 - 12 days and 0 - 50 days on the NSR and the PR, respectively. Break-up occurred on the PR in 2019-2020, but because no supercooling events were detected after break-up, the break-up period duration is defined as zero.

A total of 350 and 264 events were observed on the NSR and PR over the four seasons of measurements and 82 in the single season on the KR. Freeze-up period events comprised ~90 % of observed events on both the NSR and PR. Generally, the longer the freeze-up period, the greater the *SCDMS*. The exception to this is the single data point from the KR where the longest freeze-up period of 137 days produced a *SCDMS* of 151 ºC·minutes which is the fourth smallest value. The PR 2016-2017 season, in which the freeze-up period also lasted all winter, had a seasonal *CDMS* of 698 ºC·minutes, approximately 4.6 times greater than the KR season. Performing a linear regression using the NSR and PR data from Table 2.3 (i.e., excluding the outlier from the KR) gives the following equation,

$$SCDMS = 5.13 * d_f + 6.58 \quad (2.1)$$

where d_f is the freeze-up period duration in days and *SCDMS* is the seasonal cumulative degree minutes of supercooling in °C·minutes and R^2 = 0.977. The break-up periods generally have

significantly smaller *SCDMS* values, ranging between 4.52 to 24.1 ºC·minutes. Note that there is no correlation between break-up period duration and *SCDMS* (R^2 = 0.025).

Examples of water temperature time series from the 2016 – 2017 season from each river are plotted in Figure 2-5. Daily occurrences of supercooling events can be seen in the 16-day period at the Village site on the KR plotted in Figure 2-5a. The relatively strong positive temperatures between events, along with the fairly regular start and end times, is due to the daily hydropeaking from the Pocaterra Dam. Approximately half of the events have a 'classical' supercooling shape similar to laboratory observations (Figure 2-1). Events during this period ranged in duration between 14.1 minutes to 22.8 hours, with a mean duration of 10.7 hours and a median of 13.5 hours. The gap between supercooling events ranged between 2.8 minutes and 16.4 hours and T_P ranged between -0.002 ºC and -0.031ºC, with a mean value of -0.014ºC.

The plot in Figure 2-5b shows a freeze-up period observed at the Genesee site on the NSR, which lasted approximately 12 days and ended on December 6th, 2016. The events during this period are more irregular in their start and end times, but there still tends to be one event every day. The most extreme supercooling event measured in this study, with T_P of -0.106ºC was observed at this site on November 25, 2016 at 4:25 AM. During the first 6 days (from November 24th to 30th) T_P values tended to be larger and the events closer to the classical shape. After the first two events T_W warmed to above 0.1ºC, but quickly cooled to start the following supercooling event. The average CR_P, D, and T_P for the first 6 days were -4.14x10^{-4} ºC/minute, 14.0 hours, and -0.058ºC, respectively. For the last 6 days (November 30th to December 6th) T_P magnitudes and the intermittent positive water temperature peaks were both smaller and the average CR_P, D, and T_P were -9.76x10^{-5} ºC/minute, 21.9 hours, and -0.021 ºC, respectively. From November 30th until the completion of the freeze-up period, the water temperature never rose above freezing for more than 5 hours in a day and temperatures did not exceed 0.025ºC.

In Figure 2-5c a time series spanning 13 weeks from December 7, 2016 to March 8, 2017 recorded by the sensor near the right bank at Sta. 293.5 km on the PR is plotted. As noted previously, the ice front never reached this site, so the freeze-up period lasted the entire winter. Supercooling events occurred during 7 distinct periods that ranged in duration from 2 to 12 days, with

intermittent warm periods between these periods reaching peak temperatures from 1 – 2ºC. Many of these events have residual periods lasting for multiple days. Individual event duration ranged between 27 minutes and 9.74 days with T_P varying between -0.011ºC and -0.044ºC, with a mean value of -0.025ºC.

In Figure 2-6 some shorter duration plots showing freeze-up period events from the 2016-2017 season on each river are presented. Figure 2-6a shows six events (labeled as A1 to A6) recorded at the Village site on KR over a five-day period. In terms of shape, two events (A1 and A4) display classical supercooling behaviour, while A2, A3 and A5 have a more rectangular shape with minimal difference between T_P and the residual temperatures. The classical supercooling events (A1 and A4) have a T_P of approximately -0.06ºC, while the other events (A2, A3, and A5) have significantly smaller magnitude T_P ranging from -0.004 to -0.011ºC. In terms of duration, D ranged from approximately 1 to 20 hours, with two events (A1 and A4) starting around midnight and the remainder starting in the afternoon, with most of them ending around mid-day. The residual temperatures ranged from -0.007 to -0.010ºC for all events.

Figure 2-6b shows a series of five events on the NSR at the Quesnell site over a span of 3.5 days. Event B3 has a shape that is similar to a classical supercooling event with T_P of -0.105ºC, the second largest value observed in this study and a residual temperature of approximately -0.007ºC. Event B5 has an approximately symmetric shape, with T_P of -0.007ºC, comparable to the previous events' residual temperature range. The rest of the events (B1, B2, and B4) did not have as clear of a classical or symmetric shape. The durations of all events ranged between 5 and 19 hours; two events (B1 and B5) started in early morning, while the other three events started in the afternoon. All events except B4, ended around noon.

Figure 2-6c shows three supercooling events measured over a 13-day period on the PR. The longest event C1 had a duration of ~10 days. During the first 2.5 days, this event behaved similar to a classical constant heat flux event, with the water temperature dropping to a T_P of -0.043ºC before rising to a residual of approximately -0.007ºC. After this point the water temperature behaved more erratically, with temperatures fluctuating between -0.003 and -0.014ºC until the event ended.

The two events that followed, C2 and C3, were significantly shorter, with durations of 19.4 hours and 8.6 hours and T_P of -0.015 and -0.013ºC, respectively.

A 9-day time series collected during the break-up period on the NSR at the end of April 2018 is plotted in Figure 2-7. Supercooling events occurred on 7 of the 9 days and following every event, the water temperature increased, typically peaking at 0.5 to 1.3ºC in the early afternoon (See Figure 2-7a). The expanded vertical scale plot presented in Figure 2-7b shows that none of the events had a well-defined residual stage. Supercooling events started between midnight and 4 AM and ended between 5 and 8 AM. Events D1, D5 and D6 have a symmetric shape, falling to T_P and then rising at approximately the same rate. The remainder of the events had shapes that indicate that a residual temperature was being established just before the events ended. The magnitude of T_P was quite small for these break-up period events compared to freeze-up period events, with the strongest event reaching only -0.02ºC.

Table 2-4 presents results of a statistical analysis of all the supercooling parameters calculated from for the entire data set and for each river. Peak supercooling temperatures (T_P) on the KR reached -0.059ºC with a mean of -0.013ºC, while on the NSR and PR T_p peak values reached -0.106ºC and -0.087ºC with mean values of -0.020ºC and -0.019ºC, respectively. The average event durations (D) on the KR and NSR were comparable, with mean values of 12.3 and 9.02 hours, respectively. However, events on the PR were considerably longer with a mean D of 47.1 hours. The mean principal supercooling duration (D_P) only varied from 2.16 to 3.02 hours in the three rivers with an overall average of 2.50 hours. The mean *CDMS* for the KR and NSR were comparable at 5.53 and 5.65°C·minutes, while on the PR it was significantly larger at 20.1 °C·minutes. The average principal supercooling cooling rate (CR_P) for all rivers was -5.5×10^{-4} ºC/minute, the maximum CR_P for the NSR and PR was approximately $-1.16\text{x}10^{-2}$ ºC/minute but on the KR, it was significantly smaller at $-2.52\text{x}10^{-3}$ ºC/minute. The five parameters in Table 2.4 have mean to median ratios that vary from approximately 1.5 to 5 indicating that the underlying distributions are all skewed.

Figure 2-8 presents histograms of the percentage of supercooling events that occurred during each month for each river. On the KR, supercooling events occurred every month from November to April. Approximately 40 % of the events occurred in December, and 20 % and 25 % occurred in February and March, respectively (See Figure 2-8a). Events occurred in two distinct time periods on the NSR, October to December during freeze-up periods and March to April during break-up periods with no events occurring in January and February (see Figure 2-8b). During freeze-up periods, November was the month with the most events with 57 % followed by 25 % in December and 7 % in October. The remaining 11 % of NSR supercooling events occurred in March and April during break-up periods. The PR supercooling events spanned 6 months from December to May. Unlike the other two rivers, only 10 % of the events occurred in December, with January, February and March, accounting for 44 %, 18 % and 22 % of events, respectively. The remaining 6 % of the events occurred mostly in April, with a few in May.

Figure 2-9 presents histograms of the start and end times, t_s and t_e for both freeze-up and break-up period events calculated from the entire data set from all three rivers. The data in Figure 2-9a show the start of freeze-up period events was distributed uniformly throughout the day, with 2 to 5 % of events starting each hour except between 4 and 7 PM, when 23 % of freeze-up period events started. Figure 2-9b shows that 48 % of freeze-up period events ended between 9 AM and 3 PM. During the rest of the day, between 1 % and 5 % of freeze-up period events ended each hour. Figure 2-9c shows that 89 % of the break-up period events started between 9 PM and 8 AM. Figure 2-9d indicates that the majority of break-up period events, 54 %, ended shortly after sunrise between 7 AM and 10 AM.

Histograms of each supercooling parameter calculated for all events are plotted in Figure 2-10. In Figure 2-10a, it can be seen that T_P has a skewed distribution with a peak of 9.6 % of T_P observations between -0.003 and -0.004ºC, and with 92.5 % of all observed events having $T_P \geq$ -0.05ºC. The distribution of event duration D in Figure 2-10b is also skewed with 86.4 % of events having $D \leq 24$ hours and 29 % of events having $D \leq 1$ hour. The histogram of principal

supercooling duration D_P in Figure 2-10c has a maximum of 2.4 % at 9-10 min and 33.6 % of supercooling events take less than 30 min to reach T_P. Figure 2-10d shows *CDMS* also has a skewed distribution with a peak of 5.6 % of events in the range of 0.017 to 0.033 °C·minutes, and 34 % of all events having $CDMS \leq 0.30$ °C·minutes. Finally, the distribution in Figure 2-10e shows that 73.6 % of events have average cooling rates $CR_P \geq -5x10^{-4}$ ºC/minute with a peak of 3.7 % of the events having CR_P between $-1x10^{-4}$ and $-1.1x10^{-4}$ ºC/minute.

Empirical cumulative distribution functions (CDF) were generated from the data set for the five supercooling parameters and compared to four theoretical CDFs; log-normal, Weibull, exponential, and normal. However, only the lognormal distribution appeared to be a reasonable fit to any of the observed distributions. In Figure 2-11 the five empirical CDFs are compared to the theoretical log-normal distribution of each supercooling parameter. The log-normal distribution appears to be a good approximation of the empirical CDFs of T_P, D_P, and CR_P in Figures 2-11a, c, and e and to a lesser extent the empirical CDFs of *D* and *CDMS* in Figures 2-11b and d.

2.5 Discussion

In this study, field measurements of peak supercooling temperatures, T_P ranged between -0.002 and -0.106ºC with median and mean values of -0.013 and -0.019ºC, respectively (Table 2-4). Field measurements of T_P reported in the literature range from -0.001 to -0.145ºC (e.g., Matousek 1992; McFarlane et al. 2019). Laboratory observations of T_P reported in previous studies typically range from -0.03 to -0.09ºC (e.g., Ye et al. 2004; McFarlane et al. 2015) with Carstens (1966) and Altberg (1936) reporting values as low as -0.18 and -0.22ºC, respectively. Therefore, the field measurements of T_P reported in this study do fall within the ranges observed previously in both the field and laboratory. This comparison suggests that turbulent water bodies with cooling free surfaces supercool in a similar manner in both the laboratory and field.

Altberg (1936) described an extreme supercooling event which reached -0.22ºC and noted that no ice was present in the water until it reached a temperature of -0.18ºC. Matousek (1992) states that ice particles begin to form on the water surface at a temperature of -0.18ºC. Also, McFarlane et al. (2019) observed an extreme event where the supercooling temperature reached -0.145ºC. During this event, the air temperature reached a minimum of -2.3ºC, there was no snowfall, and

no suspended frazil was observed. Large sheets of skim ice as well as rapid growth of anchor ice on the bed and submerged equipment was observed. The absence of suspended frazil indicates that the generation of frazil ice particles, via secondary nucleation, was likely being suppressed due to a lack of seed particles (McFarlane et al. 2019). There are three conclusions that can reasonably be drawn from this discussion. First, extreme supercooling events tend to occur when there is a scarcity of seed particles. Second, when the generation of suspended frazil is suppressed, skim ice forms and in situ growth of anchor ice starts. Third, the fact that the most extreme supercooling event out of the ~700 observed in this study only reached a temperature -0.106ºC is evidence that seed particles are ubiquitous, and that the generation of suspended frazil is the norm.

The duration of supercooling events is a parameter of interest, since longer events should lead to greater volumes of ice production. Previous studies reported average event durations that ranged from 2.2 to 23 hours with a maximum of 42.7 hours (Nafziger et al. 2013; McFarlane and Clark 2021). As shown in Table 2.4, there is a significant variation in the duration of events observed in each river in this study. The NSR has the smallest median and mean durations of 3.17 and 9.02 hours and a maximum duration of 218 hours or 9.1 days. The KR has the next largest median and mean durations of 6.11 and 12.3 hours, respectively with a maximum duration of 162 hours or 6.8 days. The longest events were observed on the PR with a median duration of 7.80 hours, a mean of 47.1 hours, and maximum duration of 338 hours or 14.1 days. The median and mean durations observed in this study are largely comparable to previous studies with the exception being the PR where the mean was 47.1 hours. The maximum durations of approximately 7 to 14 days observed in this study are considerably larger than previous measurements.

Extremely long events were routinely observed on the PR but much less often on the other two rivers. During these extremely long duration events the water temperatures typically remained at an approximately constant residual temperature. The most likely explanation for this is that the cooling of the water surface was being balanced by the release of latent heat due to the generation of ice. As the cooling heat flux at the water surface increased, the rate of ice production subsequently increased, and vice-versa. Tidal cycles and dam regulation may prevent

the occurrence of extremely long supercooling events on some rivers e.g., Richard and Morse (2008). However, in the absence of these external forcing mechanisms, supercooling events can persist when the water surface is continuously cooled for long time periods. Also, there may be additional factors such as the thermal inertia of the water column that play a role in the timing and duration of events. Deeper water columns will respond more slowly and therefore may take longer to begin supercooling but may also remain supercooled for longer durations.

Measurements of cooling rates (CR_P) from laboratory experiments ranged from $-2x10^{-3}$ to $-1.3x10^{-2}$ °C/minute (Ye et al. 2004; Schneck et al. 2019). The CR_P values observed in this field study ranged from $-1.03x10^{-6}$ to $-1.18x10^{-2}$ °C/minute with median and mean values of $-2.48x10^{-4}$ °C/minute and $-5.51x10^{-4}$ °C/minute, respectively (Table 2.4). This shows that cooling rates in rivers are approximately an order of magnitude less on average than those reported in laboratory tests. This is consistent with the observation by Osterkamp (1978) that laboratory cooling rates are an order of magnitude greater than field observations. This large difference may in part be due to the fact that laboratory experiments are frequently conducted in shallower channels and tanks with uninsulated sides and bottoms. Under these conditions, cooling of the water column will occur through the sides and bottom in addition to the water surface. Therefore, the difference in cooling rates may simply be reflective of the much larger effective cooling surface area and shallow depths in many laboratory experiments.

The *CDMS* of a given supercooling event is clearly an integral measure of its "strength" since it accounts for variations in both the magnitude of supercooling and the duration of an event. However, it would be much more useful if it could also be used as an indicator or measure of frazil ice production. Frazil ice production is exothermic, thus the change in water temperature during a supercooling event is a balance between the latent heat of fusion and the net heat flux at the water boundaries. Howley et al. (2019) presented an empirical equation relating peak frazil ice concentration to the square root of the *CDMS* fitted to their dataset but additional simultaneous measurements of supercooling and frazil concentrations are needed to validate this equation. If measurements of *CDMS* prove to be a viable method for estimating frazil ice production, real-time temperature sensors deployed upstream of water intakes could be used as an early warning system for potential blockages by frazil ice.

The significant differences in the start and end times of the supercooling events during the freeze-up and break-up periods evident in Figure 2-9 are likely due to seasonal variations in solar radiation. During break-up periods, ~85 % of events started between 9 PM and 6 AM when the heat flux due to solar radiation would be negligible (Figure 2-9c). In addition, 54 % of break-up period events ended between 7 AM and 10 AM (Figure 2-9d) as the sun was rising or shortly thereafter. This is evidence that the heat flux due to solar radiation plays a dominant role in the timing of break-up period events. Freeze-up period events started almost any time of the day (Figure 2-9a) with a slight peak from 4 PM to 7 PM and they tended to end at mid-day with approximately half ending between 9 AM and 3 PM. This indicates that solar radiation is also having an impact on the timing of freeze-up period events but not to the same extent as on break-up period events.

The spatial distribution of supercooling events can be investigated since temperature loggers were deployed along the study reaches and, in a few cases, pairs were deployed close to both banks. All three rivers are regulated by upstream dams, with the KR study reach being the closest to the upstream dam, followed by the NSR and PR. The NSR study reach is also notable as the only study reach to have sites within an urban environment. On the KR, 8, 2, and 72 events were observed at the Fortress, Opal, and Village sites, respectively. These sites were 13, 21, and 36 km downstream of the dam as shown in Figure 2-2b. During the extreme hydropeaking, water at 4 to 6 ºC is released and this warm water must travel a considerable distance downstream before it starts to supercool. These field observations show that the Pocaterra Dam regulation influences frazil ice production within the study reach, though the thermal impact of the dam is greatly reduced by the time the water reaches the Village site.

Comparing the distribution of events at the three sites on the NSR with multiple seasons (3 seasons at Genesee and 4 seasons each at Quesnell and Emily Murphy) can be used to determine if there is a significant longitudinal gradient in the study reach. The Genesee site is located 46 km outside Edmonton's urban area while the Quesnell and Emily Murphy sites are 6 and 10 km inside the city, respectively. There was an average of 38, 26 and 11 events per season with corresponding *SCDMS* of 156, 147, and 75.8 °C·minutes at the Genesee, Quesnell and Emily

Murphy sites, respectively. The decrease in both the number and strength of supercooling events from outside Edmonton's large urban area to well inside implies that urbanization impacts supercooling in rivers. The two most likely mechanisms responsible for this are the urban heat island effect and stormwater outfalls. The warmer air temperatures that occur inside the heat island and the relatively warm stormwater runoff entering the river both would tend to increase water temperatures and suppress supercooling.

Calibrated numerical modelling on the PR has shown that the zone of influence of the dams on the PR ice regime extends 550 km downstream (Jasek and Pryse-Phillips 2015). The PR study reach is 275 km downstream of Peace Canyon Dam and therefore, the thermal regime of this reach would still be influenced by convection of warm water from upstream. However, because the study reach was relatively short, only 16 km in length, no significant longitudinal variations in the occurrence or properties of supercooling events were observed.

The deployment of five sensor pairs allowed for a preliminary assessment of the differences in supercooling on each side of the river. A comparison of the event statistics on each side of the PR (i.e., the left bank and the right bank) is presented in Table 2.5, with Figure 2-12 showing a temperature time series recorded by a pair of sensors at Sta. 305 km for the 2016-2017 season. As shown in Table 2.5, the left bank recorded 102 events compared to the 59 events on the right bank. As shown in Figure 2.12, one of the reasons for the difference in the number of events is due to multiple events starting and ending on the left bank during a single longer event on the right bank. Furthermore, the daily peaks in water temperature near the left bank occurred around mid-day, when incoming solar radiation would be at its peak. This behaviour is quantified in Table 2-5 with the ratio between the mean and median event duration D on the left and right bank being 0.57. The events on the left bank were also found to be slightly smaller in magnitude with average T_P of -0.02°C on the left bank compared to -0.023°C on the right bank. The smaller magnitudes for D and T_P on the left bank compared to the right bank resulted in a ratio of 0.59 between the average left and right bank *CDMS*. However, when the total *CDMS* of all supercooling events observed during this time period are compared, the two sides are nearly identical, with a ratio of 1.02.

The most likely explanation for the difference between the two sides of the river is the difference in exposure to solar radiation. As shown in Figure 2-2d, the PR flows in an easterly direction for the entirety of the study reach, therefore the left and right banks are the north and south banks, respectively. At this latitude (55.6°) the inclination of the sun means that the left/north bank would receive more solar radiation, while the right/south bank would be comparatively shaded by the river valley. Therefore, near the left bank daily heating by solar radiation may heat the water sufficiently to end a supercooling event. If the right bank is sufficiently shaded, then this heating by solar radiation will not occur and supercooling will continue. These differences in supercooling between the left and right bank could indicate that the concentration of suspended frazil may also be varying in the cross-stream direction in rivers.

2.6 Conclusions

This study reports on an analysis of 696 supercooling events observed in three Alberta rivers over multiple winter seasons. The data show that a typical event lasts less than 24 hours with peak supercooling between -0.01 and -0.02ºC. Freeze-up period events were found to start any time of day but typically ended between 9 AM and 3 PM. In contrast, almost all break-up period events occurred overnight, starting after 7 PM and ending between 7 and 10 AM. The mean principal cooling rate was calculated to be -5.51×10^{-4} ºC/minute which is approximately an order of magnitude smaller than rates observed in laboratory experiments. The cumulative degree minutes of supercooling (*CDMS),* a parameter that was proposed by Howley et al. (2019) to quantify the 'strength' of a supercooling event, had median, mean and maximum values of 2.31, 11.1 and 158 °C·minutes, respectively.

The two most extreme supercooling events observed in this study in terms of peak supercooling and duration reached a temperature of -0.106ºC and lasted ~14 days. By comparison, the most extreme events observed in previous studies reached approximately -0.2ºC and a duration of ~2 days. During these long duration events the water temperature remained at an approximately constant residual temperature. Another significant finding from this study is that extreme peak supercooling temperatures below -0.1ºC are very rare since only two events out of the 696 events measured in this study (or 0.3 %), supercooled below this threshold.

This study also examined if the frequency of events varied along and across the rivers. Significant longitudinal variations were observed on both the KR and the NSR and these were attributed to the impacts of hydropeaking and urbanization. Transverse variations were investigated on the PR and it was found that on the shaded south bank there were fewer events, but they were of longer duration compared to the north bank.

Analysis of this large dataset of supercooling parameters has revealed a number of new and significant results but additional research is clearly still needed. Future work will investigate the validity of a quantitative relationship between *CDMS* and frazil ice production as proposed by Howley et al. (2019). Simultaneous measurements of supercooling and suspended frazil concentrations are needed to accomplish this goal. If a relationship is found, then relatively inexpensive and easily deployed precision temperature loggers could be used to monitor frazil ice production. In addition, further studies that combine comprehensive measurements of the various heat flux components and supercooling observations are also a priority. The recent study by McFarlane and Clark (2021) is an excellent example of the type of research needed to more fully understand the heat fluxes that drive supercooling events.

River ice process models are becoming more widely used for research and by practicing engineers. As a result, there has been a concerted effort to improve their accuracy and scope (e.g., Blackburn and She 2019; Wazney et al. 2019). However, the ability of these models to accurately predict the occurrence and properties of supercooling events has not been thoroughly investigated. Data from this study and similar future ones could be used to conduct this type of investigation and lead to model improvements.

Acknowledgements

The map developed for this publication was produced with QGIS software (QGIS Development Team 2021) using the data provided by © OpenStreetMap contributors (http://www.osm.org/copyright). We thank Perry Fedun for his valuable technical assistance. We also thank Kerry Paslawski, Vincent McFarlane, Hayden Kalke, Chris Schneck and Rhodri Howley for their assistance with the deployment and retrieval of field instrumentation. This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) (grant nos. RGPIN-2020-04358, RGPIN-2015-04670 and RGPAS 477890-2015).

2.7 Chapter 2 Supplementary materials

Chapter 2 Tables

Table 2-1: Summary of the study reach properties for Kananaskis KR, North Saskatchewan NSR, and Peace PR Rivers (McFarlane et al. (2017); Kellerhals et al. (1972); Buehler, H. (2013))

River	Drainage Basin Area at study site [km^2]	Average Annual Flow Rate [m^3/s]	Average Slope	Average Width [m]	Average Depth [m]
KR	748	13	0.0034	30	0.60
NSR	5,670	220	0.00035	140	1.40
PR	130,000	1,590	0.00025	230	2.60

Table 2-2: Summary of the number of temperature logger deployment sites during all years of measurements on each river

River	Year				
	2015-2016	2016-2017	2017-2018	2018-2019	2019-2020
KR	-	3	-	-	-
NSR	2	4	3	-	6
PR	-	4	7	4	1

Table 2-3: Synopsis of supercooling events observed on each river during each deployment season

River	Year	Date of first event	Date of last event	Number of freeze-up period events	Number of break-up period events	Freeze-up period duration (days)	Break-up period duration (days)	Seasonal freeze-up *CDMS* (ºC·minutes)	Seasonal break-up *CDMS* (ºC·minutes)
KR	2016-2017	18-Nov-16	04-Apr-17	82	0	137	-	151	-
NSR	2015-2016	18-Nov-15	03-Apr-16	30	7	8	5	33.1	4.52
	2016-2017	19-Nov-16	15-Apr-17	53	5	17	12	151	2.95
	2017-2018	03-Nov-17	30-Apr-18	16	14	8	9	32.5	24.1
	2019-2020	29-Oct-19	28-Apr-20	212	13	46	10	189	7.4
PR	2016-2017	07-Dec-16	19-Apr-17	132	0	133	-	698	-
	2017-2018	25-Dec-17	11-May-18	80	21	36	50	216	9.87
	2018-2019	23-Dec-18	08-Apr-19	24	1	42	0.62	212	9.46
	2019-2020	04-Jan-20	15-Jan-20	6	0	15	0	85.5	0
Total				635	61				

Table 2-4: Summary of the statistics of supercooling event parameters including peak supercooling T_P, duration D, principal supercooling duration D_P, cumulative degree minutes of supercooling $CDMS$ and principal supercooling average cooling rate CR_P. Minimum and maximum values refer to magnitudes

Set	Statistical parameter	T_P (°C)	D (hours)	D_P (hours)	*CDMS* (°C·minutes)	CR_P (°C/minute)
All Rivers [696 Events]	Min.	-0.002	0.169	8.2×10^{-3}	6.77×10^{-3}	-1.03×10^{-6}
	Med.	-0.013	4.81	1.23	2.31	-2.48×10^{-4}
	Mean	-0.019	23.8	2.50	11.1	-5.51×10^{-4}
	Max.	-0.106	338	106	158	-1.18×10^{-2}
	Std. Dev.	-0.018	56.8	6.34	23.3	-1.22×10^{-3}
KR [82 Events]	Min.	-0.002	0.179	8.67×10^{-2}	2.59×10^{-2}	-5.86×10^{-6}
	Med.	-0.009	6.11	1.04	2.25	-2.00×10^{-4}
	Mean	-0.013	12.3	2.16	5.53	-3.54×10^{-4}
	Max.	-0.059	162	15.6	70.3	-2.52×10^{-3}
	Std. Dev.	-0.013	20.9	3.27	9.24	-4.81×10^{-4}
NSR [350 Events]	Min.	-0.002	0.169	8.20×10^{-3}	9.87×10^{-3}	-2.67×10^{-6}
	Med.	-0.012	3.17	1.14	1.49	-2.47×10^{-4}
	Mean	-0.020	9.02	2.18	5.65	-6.13×10^{-4}
	Max.	-0.106	218	48.6	158	-1.14×10^{-2}
	Std. Dev.	-0.020	19.5	4.12	11.9	-1.37×10^{-3}
PR [264 Events]	Min.	-0.002	0.171	0.014	6.77×10^{-3}	-1.03×10^{-6}
	Med.	-0.015	7.80	1.42	4.76	-2.61×10^{-4}
	Mean	-0.019	47.1	3.02	20.1	-5.31×10^{-4}
	Max.	-0.087	338	106	137	-1.18×10^{-2}
	Std. Dev.	-0.017	83.7	8.95	33.0	-1.17×10^{-3}

Table 2-5: Summary of the statistics of the supercooling events measured simultaneously by sensor pairs on the left bank (LB) and right bank (RB) of the PR. Parameters include peak supercooling T_P, duration D, cumulative degree minutes of supercooling $CDMS$ and total cumulative degree minutes of supercooling

Parameter	*LB*	*RB*	*LB/RB*
Number of Events	102	59	1.73
Median T_P (ºC)	-0.017	-0.021	0.81
Mean T_P (ºC)	-0.020	-0.023	0.88
Median D (hours)	11.9	21.0	0.57
Mean D (hours)	42.3	73.7	0.57
Mean $CDMS$ (ºC·minutes)	19.4	32.8	0.59
Total $CDMS$ (ºC·minutes)	1,980.26	1,933.20	1.02

Chapter 2 Figures

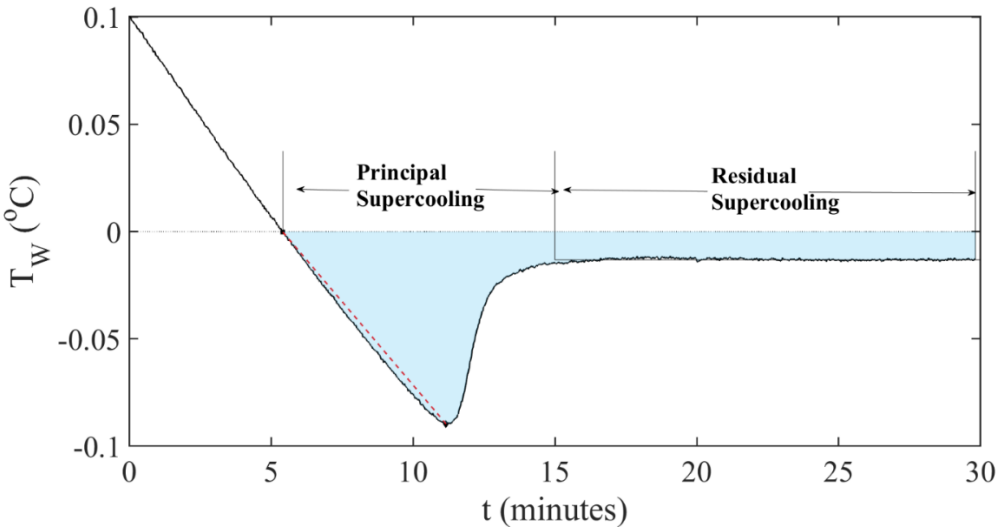

Figure 2-1: A water temperature (Tw) time series recorded in a laboratory tank being cooled by a constant surface heat flux

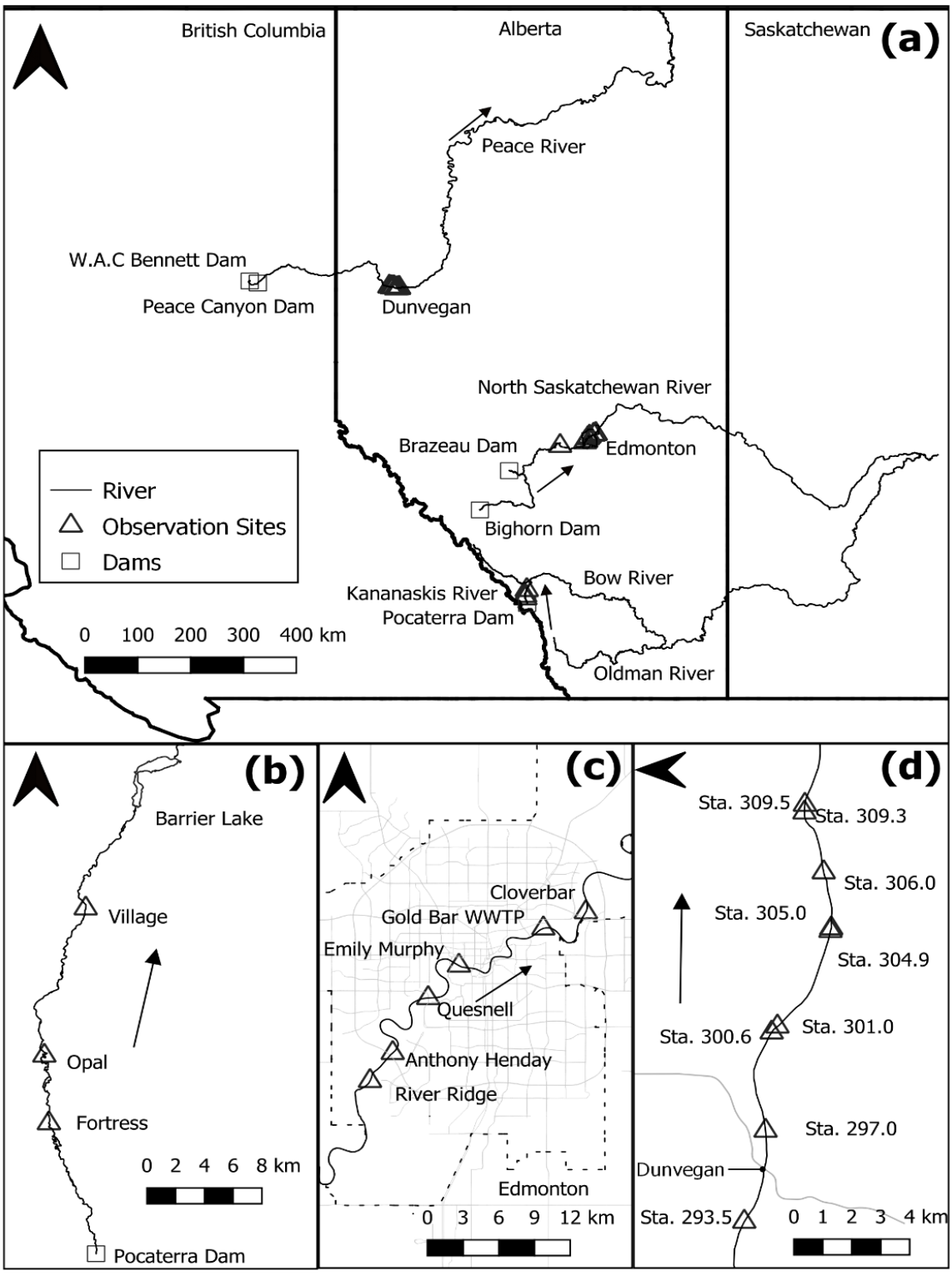

Figure 2-2: Maps showing.: (a) Geographical location of the three study reaches in Alberta. Lower maps are enlarged views of the study reaches, (b) Kananaskis River (KR), (c) North Saskatchewan River (NSR) within the City of Edmonton limits, and (d) Peace River (PR)

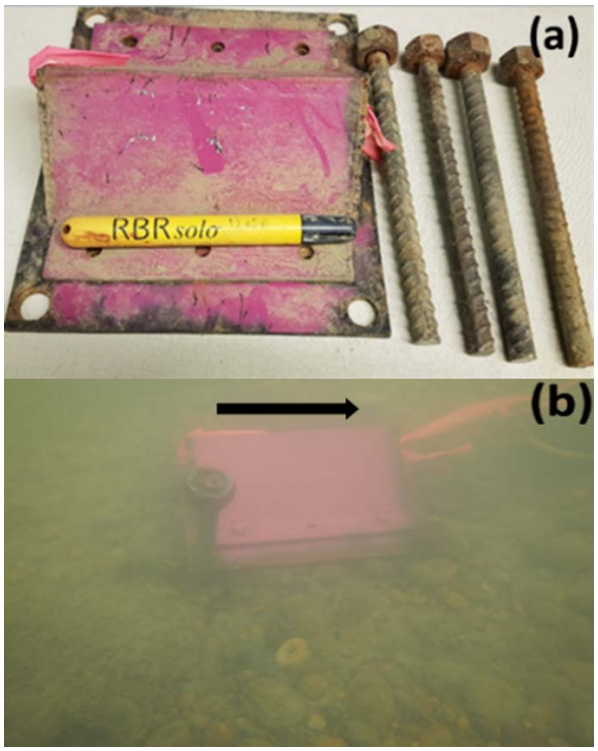

Figure 2-3: Equipment used in KR and NSR deployments: (a) RBR Solo T Temperature Logger along with protective case and anchoring pins and, (b) Case being anchored to the riverbed and approximately aligned with the river current as shown by the arrow

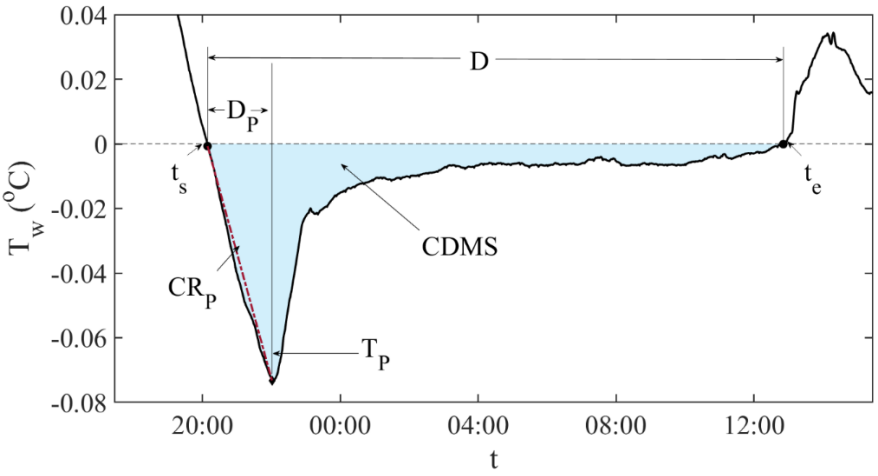

Figure 2-4: A time series of a supercooling event observed at the Genesee site on the NSR on Nov. 25 - 26, 2016. Water temperature T_w plotted as a function of time-of-day t. Graphical definitions of start time t_s, end time t_e, peak supercooling T_P, principal supercooling duration D_P, event duration D, cumulative degree minutes supercooling *CDMS*, and average principal supercooling cooling rate CR_P

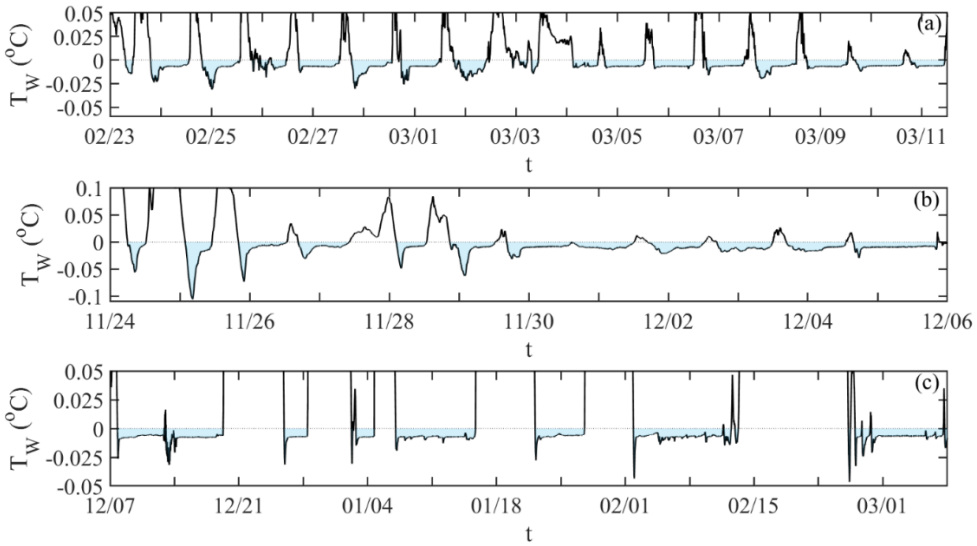

Figure 2-5: Water temperature time series from the 2016-2017 season: (a) KR, Village site, (b) NSR, Genesee site and, (c) PR Sta. 293.500 km on the right bank

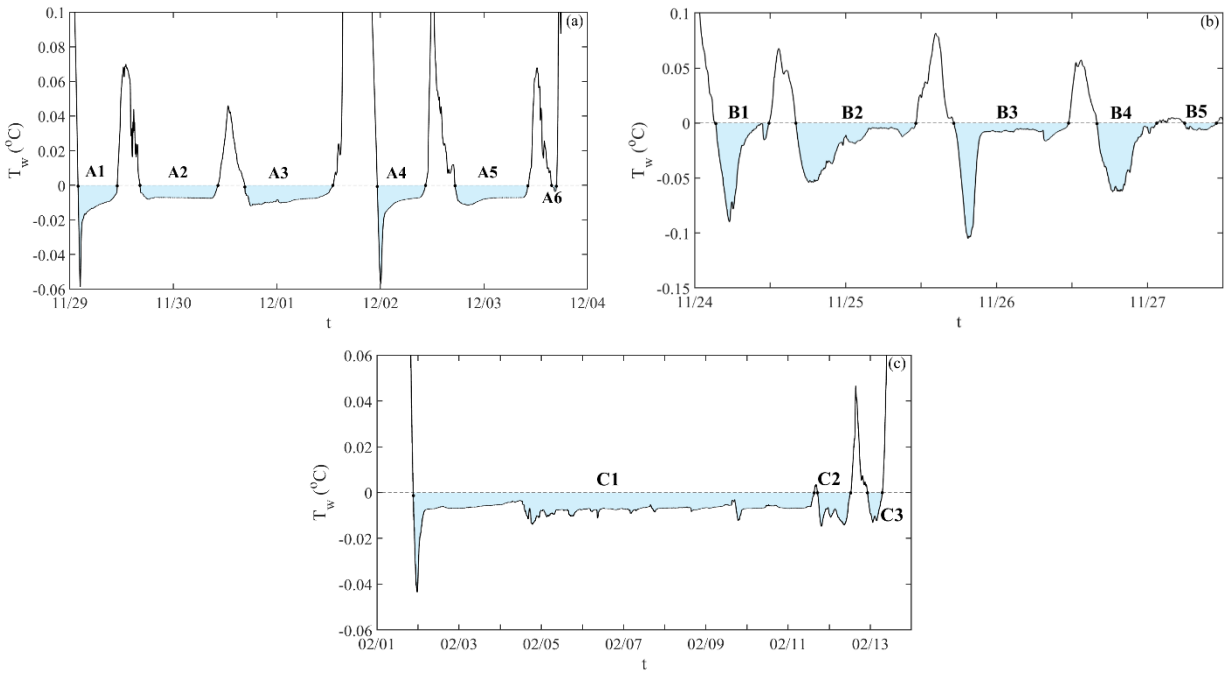

Figure 2-6: Water temperature time series showing freeze-up period supercooling events from 2016-2017: (a) KR Village site, (b) NSR Quesnell site, (c) PR Sta. 293.500 km on the right bank. Note: the start and end of each event is marked by black dots at the intersection of the water temperature time series and 0 °C

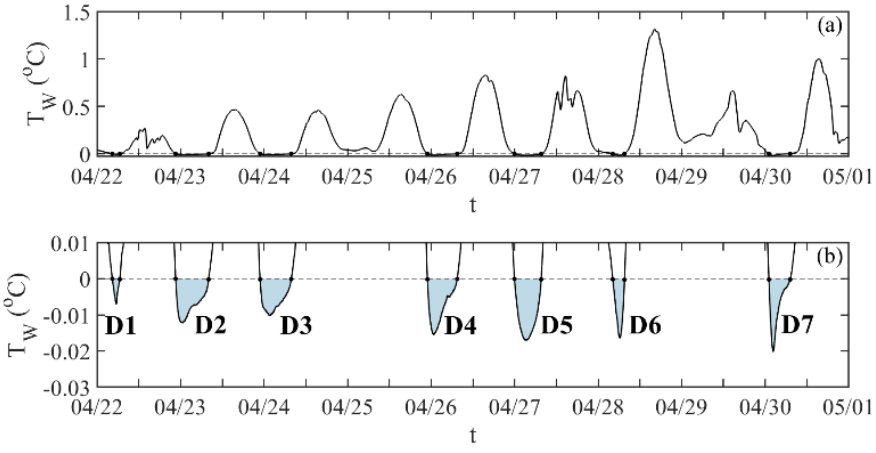

Figure 2-7: Water temperature time series during the break-up period on the NSR from April 22 to April 30th, 2018 showing: (a) the full range of the daily water temperature

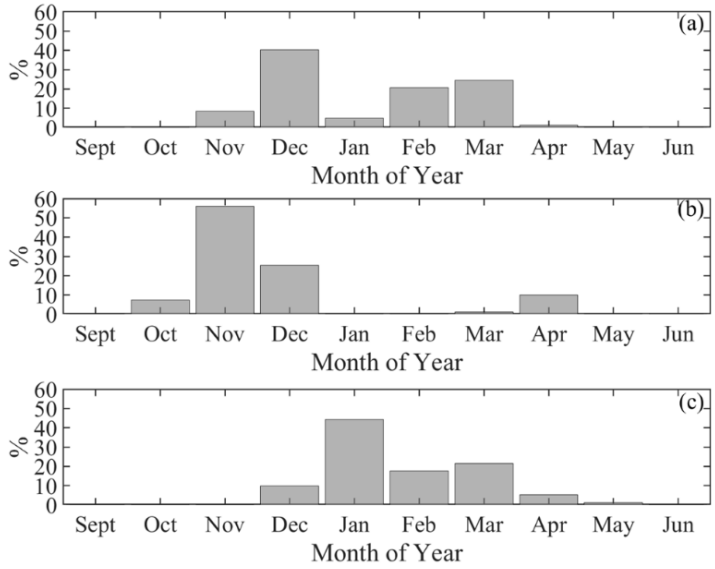

Figure 2-8: Monthly frequency distribution of supercooling events for (a) KR, (b) NSR, and (c) PR

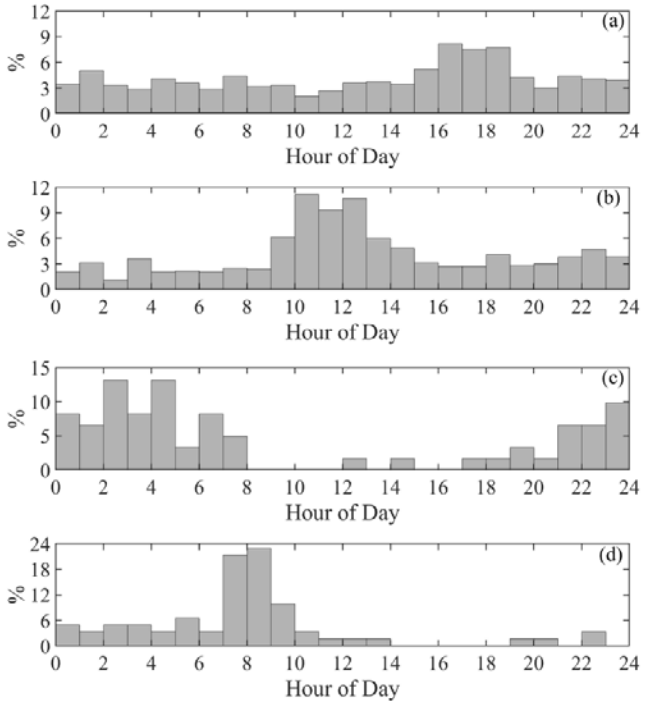

Figure 2-9: Hourly frequency distribution of supercooling events showing: (a) start and (b) end times of freeze-up period events, and (c) start and (d) end times of break-up period events calculated from the combined data set from all rivers and all years of measurements

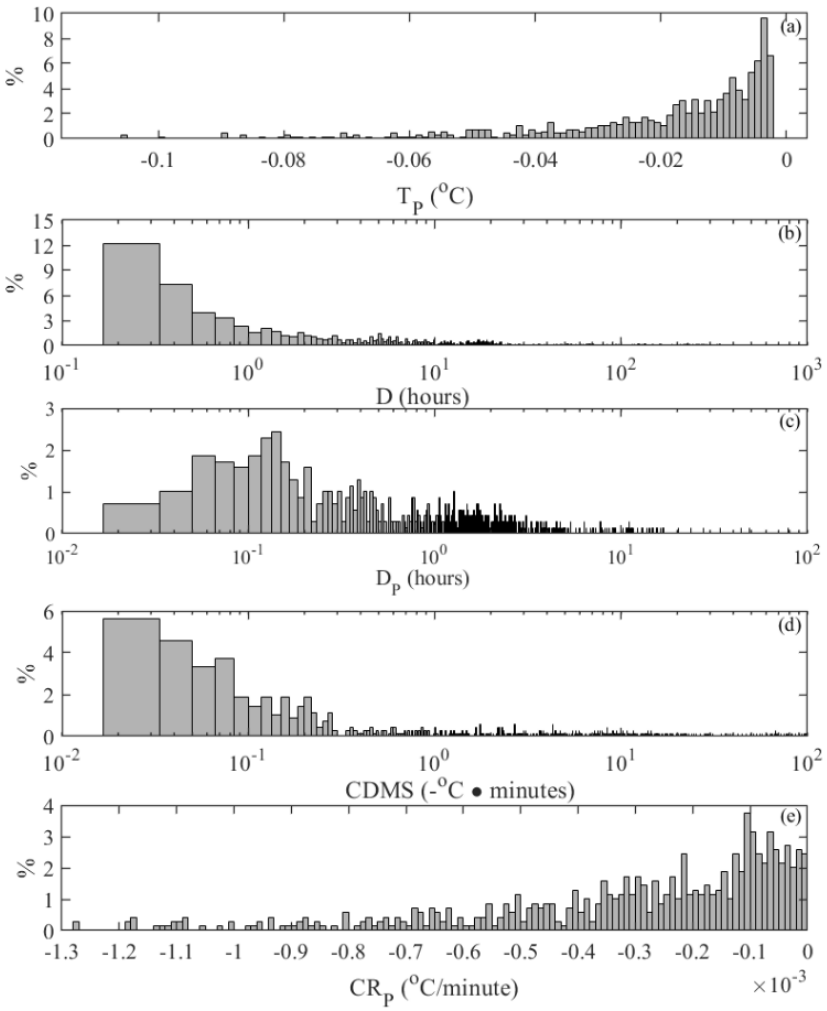

Figure 2-10: Histograms of supercooling parameters calculated from the combined data set from all rivers and all years of measurements showing: (a) peak supercooling T_P, (b) duration D, (c) principal supercooling duration D_P, (d) cumulative degree minutes of supercooling $CDMS$, and (e) principal supercooling average cooling rate CR_P

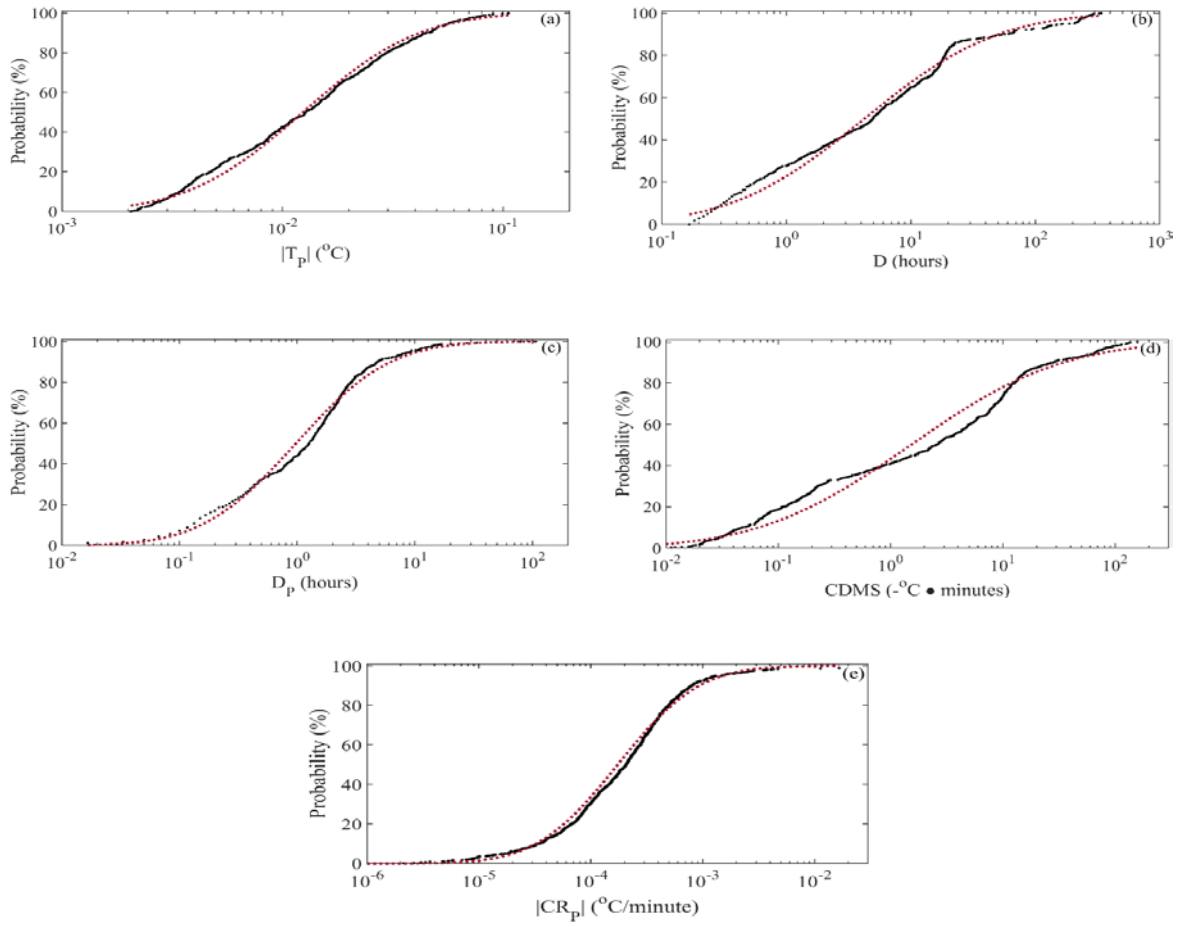

Figure 2-11: Empirical cumulative distribution functions (CDFs) of supercooling parameters (black) compared to a fitted theoretical lognormal distribution (red dotted) for (a) peak supercooling T_P, (b) duration D, (c) principal supercooling duration D_P, (d) cumulative degree minutes of supercooling $CDMS$, and (e) principal supercooling average cooling rate CR_P

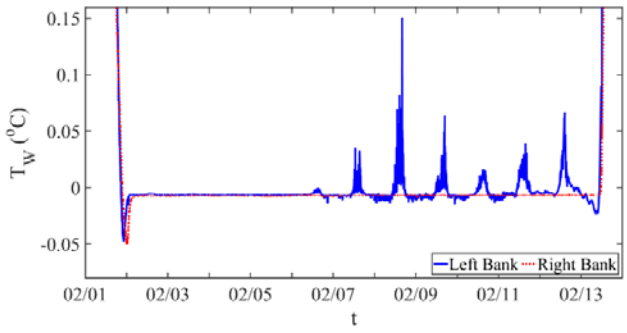

Figure 2-12: Water temperature (Tw) time series measured near the left (blue) and right (red) banks at Sta. 305 km on the PR in the 2016-2017 season

Chapter 3 : Surface energy budget of 2016-2017 supercooling events

3.1 Introduction

From late fall to early spring, northern rivers are exposed to a freezing climate that induces supercooling in the turbulent water and is a key driver of the river's ice regime. The frazil ice that forms in the turbulent water column forms the floating frazil pans which contribute to the consolidating of an ice cover, or anchor ice on the river bed (Daly 1994). Suspended frazil has been documented to cause water intake blockages (Richard and Morse 2008), while anchor ice has been identified as a significant contributor to sediment transportation in northern rivers (Kalke et al. 2017). As such, a better understanding of the conditions that are conducive to supercooling and in particular the heat fluxes across the air-water interface may inform future practice regarding river ice engineering during the freeze-up period.

Ashton (1986, 2013) stated that there are up to eleven modes of heat transfer that can regulate river water temperatures: shortwave radiation Q_{sw}, longwave radiation Q_{lw}, sensible heat flux Q_s, evaporative heat flux Q_e, precipitation Q_p, heat exchange with the bed Q_{bed}, friction of flow Q_f, turbulence Q_t, heat from groundwater inflow and reservoir releases Q_{res} and Q_{gw}, and finally the heat of fusion from the production and melting of ice Q_{ice}.

Most previous field studies of the energy budget of rivers tended to focus on ecological impacts of temperature fluctuations; few studies focused on supercooling specifically. However, these studies may still provide insights into the dominant factors of a river's energy budget. The net radiative heat flux (summation of the incoming and outgoing shortwave and longwave heat fluxes), was found to be the most significant heat flux in these previous studies (Brown 1969;Evans et al. 1998;Webb and Zhang 2004), with Brown et al. (1969) noting that net radiation was the dominant heat flux while evaporation and convection had minimal influence. Evans et al. (1998) studied the heat flux dynamics on the River Blithe in Staffordshire, UK, finding that net shortwave heat flux contributed to 97.6 % of the total positive heat flux, while the net longwave heat flux accounted for 54.0 % of the total negative heat flux for the study period. The evaporative heat flux was found to account for 23.6 % of all heat losses, and found to be comparable to previous studies on UK rivers (Evans et al. 1998). The sensible heat flux was the smallest of the surface heat fluxes, accounting for only 1.2 % and 5.3 % of all positive and negative heat fluxes, respectively. Hannah

et al. (2004) studied the heat budget of a stream in Scotland and found that all heat loss and 39 % of heat gains occurred across the air-water surface, while 61 % of gains were through the channel bed. Similar to previous studies, net radiation was found to be negative (net heat loss from the water column) due to decreasing daylight until well after the winter solstice. Air temperature was above freezing for most of the study, ranging from -6.01 to 11.40 ^{o}C with a mean of 3.17 ^{o}C. Thus, sensible heat flux was found to be the largest heat source in this study at 38.7 % of all positive heat fluxes. Evaporative heat flux and net radiation were the largest heat sinks at 73.1 % and 26.9 %, respectively. Hannah et al. (2004) suggest that the reason for a lower evaporation contribution in other studies is due to the wind shelter caused by forested studies sites. Groundwater driven streams are found to be warmer and more thermally stable than glacier run-off (Brown et al. 2006), though the degree of groundwater influence on downstream water temperature depends on the strength of the source and distance upstream (O'Driscoll and DeWalle 2006). Multiple studies found that rivers with higher flowrates and thus deeper water depths are less sensitive to changes in heat fluxes (Brown 1969; Evans et al. 1998; Clark et al. 1999; Cozzetto et al. 2006; Webb et al. 2008), which is important when considering cross-flow temperature dynamics (Clark et al. 1999).

Studies reporting detailed investigations of heat fluxes during supercooling in rivers are rare. One study reported supercooling observations made during the 2005-2006 winter on the St. Lawrence River near Quebec City, QC, Canada (Richard and Morse 2008; Richard et al. 2015). Richard and Morse (2008) observed ~100 supercooling events while monitoring suspended frazil, surface ice concentration, and meteorological parameters. Richard and Morse (2008) observed that supercooling events often occurred at night when the outgoing longwave would exceed the incoming net radiation. On average, the air temperatures during the events were below -5.4^{o}C and relative humidity was less than 86 %. However, supercooling was also observed to happen in warmer air conditions up to -2.3^{o}C, during a clear night, conditions that would lead to a strong longwave heat loss (Richard and Morse 2008). For 80 % of supercooling events, the precipitation in the preceding 12 hours was less than 1.8 mm. Richard and Morse (2008) also noted trends of events starting at night implying that the net negative longwave radiation and zero shortwave radiation being the predominant cooling effect on the river.

Richard et al. (2015) modelled the frazil ice production during this season and observed that supercooling is rare when it is snowing due to increased cloud cover reducing the heat loss from

longwave radiation. For the 2005-2006 season, the net radiative heat flux (shortwave and longwave) had an average value of -44 W/m^2 at night, with a seasonal average of +25 W/m^2. Sensible and evaporative heat fluxes were also found to be significant with a reported greatest magnitude of heat loss of -105 and -178 W/m^2 for evaporative and sensible heat flux, respectively. Supercooling events often occurred on nights when the net heat flux dropped below -200 W/m^2, with the greatest magnitude net heat flux <-500 W/m^2. Richard et al. (2015) observed significant variability between the various heat flux components throughout the day, with differences up to ~200 W/m^2 between day and night.

McFarlane and Clark (2021) investigated the heat budget of the Dauphin River in Manitoba, Canada during supercooling from the end of October to mid-November 2019. The study directly measured incoming and outgoing shortwave and longwave radiation over land to avoid risk to equipment, as well as water temperatures, air temperature, wind speed and direction, relative humidity and barometric pressure. For the six supercooling events observed during the study, McFarlane and Clark (2021) reported that the largest event averaged cooling heat flux component was the net longwave component that ranged from -78.1 to -30.2 W/m^2, followed by the sensible heat flux that ranged from -42.3 W/m^2 to -5.6 W/m^2. The evaporative component had the smallest range, from -31.0 to -9.0 W/m^2. The largest positive heat flux was the net shortwave radiation that varied between 0.8 and 9.8 W/m^2, while the heat flux due to friction from the bed and banks varied from 2.8 to 3.0 W/m^2. The resulting event averaged net heat flux varied from -127 to -37.4 W/m^2 (McFarlane and Clark 2021). Five of the six observed events started around 6 PM and ended between 6 AM and noon the following day, with one event occurring in the early morning. The event durations are estimated to have ranged from 1 to 18 hours. They concluded that the net longwave was the most significant heat flux during supercooling events. McFarlane and Clark (2021) estimated that the difference between outgoing radiative heat flux from the land can vary up to 158 W/m2 on a sub-daily time scale from the water surface, and thus accurate outgoing radiation measurements should be collected over the water. In cases where such measurements are not possible, incoming shortwave and longwave radiation can be measured from the bank, while the outgoing components are calculated.

From these previous studies, it is evident that more studies regarding the energy budget during supercooling are required to improve our understanding of the relationship between the water temperature and heat flux dynamics. Boyd et al. (2022) observed 696 supercooling events and analyzed their properties including peak supercooling temperatures, durations, start and end times and cumulative degree minutes of supercooling. In this study a subset of that dataset, the 2016-2017 season on the North Saskatchewan and Peace Rivers, was used along with locally deployed weather station data to calculate the surface heat fluxes and investigate potential relationships between these heat fluxes and supercooling events. This study aims to improve our understanding of the interactions between surface heat fluxes and supercooling in rivers.

3.2 Study Area

The supercooling measurements were collected at sites on the North Saskatchewan and Peace Rivers as shown in Figure 3-1. The North Saskatchewan is the smaller of the two rivers, with an annual flow rate of 220 m^3/s and an average width and depth of 136 m and 1.40 m, respectfully (Table 3-1). There were four water temperature measurement sites on the North Saskatchewan at Genesee, River Ridge, Quesnell, and Emily Murphy as shown in Figure 3-1b. Meteorological measurements were collected by a weather station installed at the Royal Mayfair Golf Club adjacent to the North Saskatchewan River in Edmonton AB, see Figure 3-1b. Supplemental meteorological data was acquired from the Environment and Climate Change Canada (ECCC), EDMONTON INTERNATIONAL CS ALBERTA (I.D. 3012206).

The Peace River is a significantly larger river with an average annual flow rate of $1.59x10^3$ m^3/s and an average width and depth of 227 m and 2.56 m, respectfully (Table 3-1). Two pairs of sensors were deployed at Sta. 293.5 km and Sta. 305 km (Figure 3-1c). As shown in Figure 3-1c, meteorological measurements were collected by a weather station installed at Sta. 309.3 km near the water intake for Fairview AB and the closest ECCC station PEACE RIVER An ALBERTA (I.D. 3075041).

3.3 Methodology

3.3.1 Overview of methodology

The water temperature time series that were analyzed for this study were taken from the 2016-2017 season on the North Saskatchewan and Peace Rivers collected for the supercooling study described in Boyd et al. (2021). RBR Solo T data loggers (accuracy $\pm$ 0.002 ºC) were anchored to the river bed in protective metal casings to measure water temperature in 1-minute intervals. On the North Saskatchewan, the sensors were deployed at a wading depth of ~0.75 m, while the Peace River sensors were deployed in deeper water and cabled back to shore (Boyd et al. 2021).

To collect meteorological data for this study, two weather stations were installed: the Royal Mayfair Golf Club (53°32'7.61"N, 113°32'27.84"W) approximately 100 m south of the North Saskatchewan River and near the Fairview AB water intake (55°54'34.70"N 118°23'34.63"W). The make, model and specifications of the data logger and sensors are listed in Table 3-2. These weather stations sampled air temperature, relative humidity, wind speed, wind direction and shortwave radiation at 10 minute intervals. In addition, hourly cloud coverage (in tenths) from human observations were obtained from the ECCC from the EDMONTON INTERNATIONAL CS ALBERTA (I.D. 3012206) and PEACE RIVER An ALBERTA (I.D. 3075041) stations.

The distances between weather stations and observation sites shown in Figure 3-1 are summarized in Table 3-3. On the North Saskatchewan, Genesee was the most distant at 47 km and 52 km from the airport and Mayfair weather stations, respectively. The next farthest observation site from the weather stations is River Ridge site was 12 km and 16 km from the Mayfair and airport weather stations respectively. The Quesnell and Emily Murphy sites both are 25 km from the airport station and 5 km and 1 km respectively from the Mayfair station. On the PR, the measurement site at Sta. 293.5 km were 15 km and 82 km from the Fairview and Peace River weather stations, respectively. The Sta. 305 km site was 7 km and 73 km from the Fairview and Peace River weather stations, respectively. It should also be noted that both weather stations are located north of the river, along with some distance between the weather station and the observation point (the station at the intake is close to the edge of the river, but Peace River A is several kilometers away).

McFarlane and Clark (2021) used seven modes of heat transfer to describe the river water temperatures: short – wave radiation Q_{sw}, long – wave radiation Q_{lw}, sensible heat flux Q_s,

evaporative heat flux Q_e, precipitation Q_p, heat exchange with the bed Q_{bed}. Therefore, the net heat flux is given by,

$$Q_{net} = Q_{sw} + Q_{lw} \pm Q_s \pm Q_e \pm Q_p \pm Q_{bed} \quad (3.1)$$

This study is focused on the heat fluxes during supercooling and therefore the heat flux from snow fall was assumed to be 0 W/m^2 as the snow crystals would not melt (Ashton 1986, 2013). The heat flux due to rainfall was assumed to be negligible because Ashton (2013) showed that a 2 mm/hour rainfall at a 5 ^{o}C temperature difference between the air and water resulting in a heat flux of 11.6 W/m^2. This is a relatively small heat flux in addition to the fact that rainfall during the time periods when supercooling events occur is rare. When analyzing their measurements of both air-water and ground heat transfer, Hannah et al. (2004) and Evans et al. (1998) concluded that 71.7 % and 82 % of total heat transfers occur at the air water interface, respectively.

Therefore, the net heat flux can be approximated by,

$$Q_{net} \approx Q_{sw} \pm Q_{lw} \pm Q_s \pm Q_e \quad (3.2)$$

The heat flux due to short-wave radiation Q_{sw} is given by,

$$Q_{sw} = (1 - \alpha_{sur})\varphi_s \quad (3.3)$$

where α_{sur} is the albedo of the water surface and φ_s the incident short-wave solar radiation. Hicks et al. (2008) found that the albedo of water ranged from 0.05-0.15. Since this study assumes open water conditions, a constant surface albedo of α_{sur} = 0.1 was assumed, reducing Eq. 3.3 to,

$$Q_{sw} = 0.9\varphi_s \quad (3.4)$$

The set of equations developed by König-Langlo and Augstein (1994) and used by Richard et al. (2015) was used to estimate the net longwave heat flux Q_{lw} as follows,

$$Q_{lw} = (1 - \alpha_W^{LW})(0.765 + 0.22n^3)\sigma(T_a + 273.15)^4 - \epsilon_w\sigma(T_w + 273.15)^4 \quad (3.5)$$

where T_w is the water temperature, T_a is the air temperature, and n is the fraction of cloud cover. The constants in Eq. 3.5: emissivity of water ϵ_w, Stefan-Boltzmann constant σ, and longwave albedo of water α_W^{LW}, were assumed to be 0.97, 5.67x10^{-8} W/m^{2}K^4, and 0.03 respectively (Richard et al. 2015). The fraction of cloud cover was taken from the hourly observation of cloud tenths from the associated ECCC weather station. Substituting the constants into Eq. 3.5, gives,

$$Q_{lw} = 5.5 \cdot 10^{-8}[(0.765 + 0.22n^3)(T_a + 273.15)^4 - (T_w + 273.15)^4] \quad (3.6)$$

The evaporative and sensible heat fluxes were calculated using a 'Dalton-type' equation, which has been used in multiple previous studies (Aston 2013; McFarlane and Clark 2021). These types of equations assume a relationship between Q_s and Q_e as given by,

$$Q_s = BQ_e \quad (3.7)$$

where B is the Bowen ratio B (Bowen 1926) given by,

$$B = C\left(\frac{T_w - T_a}{e_s - e_a}\right); C = 0.46\left(\frac{P_a}{760}\right) \quad (3.8)$$

where e_s and e_a are the saturated and actual vapour pressures, respectively, and P_a is the barometric pressure.

The evaporative heat flux Q_e was estimated using the empirical equation suggested by Ryan et al. (1974) given by,

$$Q_e = -[2.70(\Delta T_v)^{\frac{1}{3}} + 3.20V_z](e_s - e_z) \quad (3.9)$$

where ΔT_v is the difference in 'virtual temperature' or the effective temperature difference for free convection, V_z is the wind speed, and e_z is the saturated vapour pressure (Ryan et al. 1974)). The equation for ΔT_v is given by,

$$\Delta T_v = \frac{T_w+274.15}{1-0.378\left(\frac{e_s}{P_a}\right)} - \frac{Ta+274.15}{1-0.378\left(\frac{e_z}{P_a}\right)} \tag{3.10}$$

According to Ashton (1986, 2013), the saturated vapour pressure over open water can be calculated by,

$$e_s = 6.11 \exp\left(\frac{17.62\, T_a}{243.12 + T_a}\right) \tag{3.11}$$

Substituting Eqs. 3.8 and 3.9 into Eq. 3.7 results in the following equation for sensible heat flux,

$$Q_{s=} - P_a[1.63x10^{-3}(\Delta T_v)^{\frac{1}{3}} + 1.94 \times 10^{-3} V_z\,](T_w - T_a) \tag{3.12}$$

3.3.2 Limitations of methodology

While this study does provide insight into the dynamics of heat fluxes at the water surface during supercooling, it is important to keep in mind the assumptions and limitations of the methodology that introduce uncertainty in the results. The uncertainty can be divided into quantitative uncertainties that can be estimated as a cumulative uncertainty when discussing the results, and qualitative limitations whose impact can be understood but not estimated.

3.3.2.1 Quantitative uncertainty

The estimated quantitative uncertainty for each heat flux component and the net heat flux is summarized in Table 3-4. The uncertainty is estimated based on the accuracy of the sensors used as well as a literature review of the accuracy of the sensors.

Shortwave radiation was measured by using a Silicon Pyranometer Smart Sensor S-LIB-M003, which is stated by Onset (2021)c to have an accuracy of ±10 W/m^2 (Table 3-2). Additional factors

that may influence the measurements of shortwave radiation include shading from local topography and snow piling on the sensors, which may reduce the measurement of incoming solar radiation, but cannot result in a negative value. During deployment of both weather stations the measured shortwave radiation consistently reached a minimum of 0.60 W/m^2 suggests a potential bias in the equipment sensor, but without being certain about the actual reason for the non-zero reading at night, no adjustment was made to the data. In the interest of readability, the minimum value of any shortwave radiation statistic of 0.54 W/m^2 (90 % of the 60 W/m^2 minimum) will be interpreted as zero.

The longwave radiation equation developed by König – Langlo and Augstein (1994) was found to have a root mean deviation of less than 16 W/m^2 when compared to data collected in both Arctic and Antarctic conditions. Since the equation is based on the Stefan-Boltzmann law for longwave radiation, there should not be any regional bias inherent to the set of equations. However, König – Langlo and Augstein (1994) note that similar methods attempt to take into account effects of ice crystals and other atmospheric conditions that may not be as common in the polar regions.

Ashton (1986, 2013) reviewed four "Dalton type" equations, including Ryan et al. (1974), for evaporative heat flux, converting the equations to the sensible heat flux equation using the Bowen Ratio. Ashton (1986, 2013) concluded that Ryan et al. (1974) was the recommended means to compute evaporative heat flux because it did not assume a constant 'free convection' term for still-air conditions. This meant that the equation predicts a changing heat flux with changing temperature differences as well as wind speed, rather than assuming a constant value.

The evaporative heat flux developed by Webb and Zhang (1997) and reviewed by McFarlane and Clark (2021) described by,

$$Q_e = \frac{0.165(0.8 + 0.864V_z)(e_s - e_a)L_v}{86.4 \times 10^3} \tag{3.13}$$

Where L_v is the latent heat of vaporization of water (~22.6x10^5 J/kg) can be consolidated to,

$$Q_e = (3.45 + 3.71V_z)(e_s - e_a) \tag{3.14}$$

resulting in a constant free convection term of ~3.5. Comparing to the equations reviewed by Ashton (1986, 2013), it is quite comparable to the 'Kuzmin' formula given by,

$$Q_e = (3.4 + 3.62V_z)(e_s - e_z) \quad (3.15)$$

with Eq. 3.14 predicting an effective wind speed (and thus evaporative heat flux) 2 to 3 % larger than Eq. 3.15 for wind speeds between 0 to 100 km/hr.

Considering the low wind speed measured from both weather stations, Ryan et al. (1974) would be less likely to over-predict the evaporative heat flux compared to the other evaporative heat fluxes. However, since there is no means to determine a 'true' uncertainty value, the uncertainty for Eq. 17 as determined by McFarlane and Clark (2021) are used as conservative estimates. This gives a conservative estimate of the uncertainty of ±20.0 and ±14.7 W/m^2 for the sensible and evaporative heat flux, respectively.

For both sensible and evaporative heat fluxes, the wind speed is assumed to be measured 2 m above the water surface. For this study, it is assumed that the wind speed measured at the weather station is representative of the wind speed at 2 m height, with no adjustments made.

The cumulative range of uncertainty in the methods employed is ±31.2 W/m^2 (Table 3-4). It should be noted that these estimates regarding empirical equations assume that the original uncertainty estimates have comparable equipment uncertainty for parameter measurements.

3.3.2.2 Qualitative uncertainty

The first assumption made in the methodology are that the water surface remains free of ice and keeps a constant shortwave albedo of 0.1 for the entire season. Since surface ice pans insulates the water surface from the air, it both dampens the effect of sensible heat flux, and alters the effective albedo for shortwave and longwave radiation. For shortwave radiation the albedo of frazil pans could likely be compared to snow ice, giving an albedo range between 0.30 – 0.55 (Hicks et al. 2008). The emissivity of the frazil pans to longwave radiation would also be variable, reducing both the outgoing and incoming longwave radiation (Richard et al. 2015).

The second assumption was that the conditions measured at the relevant weather stations were accurate for the conditions at the supercooling observation site. Hubbard (1994) noted that for the High Plains of the United States of America (eastern Nebraska to the Rocky Mountains in Colorado) a 5 year study concluded that for air temperature, solar radiation, and relative humidity, a weather station is 90 % accurate for a ~30 km radius. In order to have similarly accurate measurement of wind speed, a weather station is only accurate for any location less than 10 km away (Hubbard 1994). Quiñones et al. (2019) analyzed the accuracy of air temperature networks on the Columbia Plateau, a region where the topography traps cool air in a 'valley effect' making significant difference between two points due to elevation. The study found that the radius of influence (RI) of weather station in complicated terrain fluctuates over time as well as space with the mean RI for minimum and maximum air temperature were 20 km and 21 km, respectively (Quiñones et al. 2019). Quiñones et al. (2019) also noted that winter months may reduce the variation due to snowfall and lack of green vegetation making the local landscape more homogenous.

Since both U of A weather stations used in this study were located within the river valley, the impact of elevation differences on air temperature should be minimal. Comparing these ranges of acceptable distances to the straight-line distances between observation sites and the weather stations in Table 3-3, it is apparent that most of the supercooling observation sites are within the more conservative RI proposed by Quiñones et al. (2019), with the exception of Genesee on the NSR and all the PR sites regarding cloud cover data. Since the meteorological parameters used for heat flux calculations were assumed to be the same for all observation sites, and the only differences in *heat flux* is due to the local differences in *Tw*, the difference between the local *heat flux for* each site is negligible. the difference between calculated heat flux components at different sites is minimal (less than 5 W/m^2), with a max difference in net heat flux < 10 W/m^2

The third assumption made is the applicability of the empirical equations from literature to the conditions found in this study. While none of the empirical equations specify that they require adjustment of parameters depending on the location of measurements, it is currently unknown if the equations carry any significant biases that would translate into additional uncertainty.

3.4 Results

3.4.1 Graphical analysis of timeseries

During the 2016-2017 winter season, the freeze-up on the NSR lasted from November 19th to December 6th. Time-series of air and water temperatures, shortwave radiation and heat fluxes for the 2016 freeze-up period on the NSR are plotted in Figure 3-2. Air temperatures ranged from -19.3 to 5.74ºC with an average of -4.4ºC and shortwave radiation varied from 0 to 286 W/m^2 with an average of 17.0 W/m^2 (Figure 3-2a). During the first 14 days of freeze-up, long-wave radiation was the dominant negative heat flux with Q_{lw} and Q_s having mean values of -32.9 and -11.3 W/m^2, respectively. As the air temperature dropped from approximately -5ºC to -19ºC over the last three days of freeze-up, sensible heat flux became the dominant with minimum values of -156 W/m^2 and -125 W/m^2 for Q_s and Q_{lw}, respectively.

In Figures 3-2c to 3-2f the net heat flux is plotted against the water temperatures at the four NSR sites. It is evident that net heat flux was $\geq$ -100 W/m^2 for most of freeze-up, with a mean between -28.7 and -30.3 W/m^2 for the first 14 days. In the final three days of the freeze-up season, Q_n dropped to a minimum between -253.6 and 254.3 W/m^2 with an average value around -94 W/m^2. While supercooling events during this time series tend to reach peak supercooling T_P of around -0.05ºC or warmer, on November 25, 2016, at 4:25 AM a T_P of -0.106ºC (the lowest value measured in this study), was recorded at Genesee (Fig 2-c) and a T_p of – 0.105ºC was recoded at Quesnell at 7:28 PM (Figure 3-2e). Q_n was fairly small magnitude preceding and during these principal supercooling periods, reaching a minimum of -82.7 W/m^2 at the time of T_P. It is interesting to note that T_p did not drop to even lower temperatures during times of periods of greater negative Q_n in the time series, such as in the last three days of freeze up. This implies that the local ice conditions such as surface ice concentration, the rate of seeding particles and frazil production may have a major role in controlling the degree of supercooling. The daily maximum in Q_n is between -37.0 and 248 W/m^2 happening around mid-day during the peak in solar radiation. Most supercooling events ended during or shortly after a daily peak in Q_n.

During break-up on the NSR, three supercooling events were observed at Genesee as shown in Figure 3-3. From Figure 3-3a, it can be seen that wind speed was less than 2 m/s during the recorded periods of supercooling, and a cloud cover fraction was high during events (average values of 1, 1, and 0.8 for the three events respectively). The first event was observed on April 15th

from 3:53 AM to 7:38 AM, ending 2.2 hours after sunrise (Figure 3-3d). The air temperature is below freezing during this event, reaching a minimum of -2.7°C (Figure 3-3b). The longwave heat flux was the greatest negative heat flux reaching a minimum of -16.3 W/m^2, followed by sensible heat flux at -5.71 W/m^2 (Figure 3-3c). Even though sunrise started around 5:30 AM, the water continued to cool to a minimum temperature until after the net heat flux changed sign. The other two break-up events occurred between evening of April 15th to the morning of April 17th during the same period of negative net heat flux. The second break-up event started roughly 2 hours after sunset (Figure 3-3b) lasting from 9:19 PM to 10:20 PM during a period of fairly constant negative heat flux (-35.2 to -32.5 W/m^2). The longwave again dominates this event with an average heat flux of -21.5 W/m^2 while sensible heat had an average value of -9.9 W/m^2. It is notable that the heat flux during this event is strictly negative even as the event ends and rises above 0°C, a sign of the uncertainty in the heat flux calculations. The final break-up event at Genesee started at 10:48 PM (28 min after the second event ended) and lasted until 8:39 AM on April 16th, approximately 3 hours after sunrise. Like the other two events, longwave radiation contributed the majority of the negative heat flux with an average of -48.0 W/m^2 (Figure 3-3d). The net heat flux was relatively constant for the first 3 hours of the event, until a drop in the longwave heat flux to around -65 W/m^2 occurred due to an updated cloud cover estimate (Figure 3-3a). The net heat flux then follows the air temperature trend dropping to a minimum value of -94.0 W/m^2 before being swiftly warmed and turned positive by the increasing shortwave radiation at sunrise (Figure 3-3c). It is of interest to note that the following evening, a similar drop in heat flux is observed with the net heat flux dropping to ~-60 W/m^2, but no supercooling was observed.

Time-series of air and water temperatures, shortwave radiation and heat fluxes for the site near the left bank at Sta. 293.5 km on the PR from December 1st, 2016, to March 14th, 2017 is plotted in Figure 3-4. In Figure 3-4b, it can be seen that T_a ranged from -28.7 to +10.5 °C and the average T_a was -9.7°C. The time series of short-wave radiation has a gradually increasing trend in the daily peaks from 100 to 200 W/m^2 in December and from 300 to 400 W/m^2 in February and early March. In Figure 3-4c the time series of the longwave and sensible heat flux components appeared to follow the same trend and have comparable mean values of -71.4 W/m^2 and -52.6 W/m^2, respectively. The sensible heat flux reached a peak value of -233 W/m^2 compared to longwave peak of -158 W/m^2. Evaporative heat flux has no significant magnitude (mean -4.96 W/m^2 in this

period) except for when the air temperature is above 0ºC (Figure 3-4b) and the wind speed is significant (Figure 3-4a). The period of warm air temperature on February 11-12, 2017, coincide with the period of greatest negative evaporative heat flux, with the average evaporative heat flux of -23.6 W/m^2. During this period, wind speed was significantly higher, with an average of 7.8 m/s, and the greatest average wind speed and lowest average evaporative heat flux during supercooling is observed during this time period.

The time series in Figure 3-4d shows extended multi-day events or periods of near-continuous supercooling tended to occur when the air temperature drops below -10ºC, with the mean air temperature during the events that lasted for two days or more being -13.8 ºC. For the first two extended supercooling events (December 7th 18:19 to December 12th 23:50 and December 14th 1:32 to December 19th 6:23), the net heat flux is strictly negative and only going positive near the end of the event. This period also sees the net heat flux reach its lowest value of -354 W/m^2 as the T_a reached its lowest temperature in the season of – 28.7 ºC. The other multi-day events had periods where the net heat flux rises above zero every day, following the peaks in solar radiation around mid-day. The longwave and sensible heat fluxes are strongly negative during these extended events with event averaged longwave and sensible heat fluxes ranging between -69.3 and -107 W/m^2 and between -33. to -123 W/m^2, respectively.

Figure 3-5 zooms in on a 10-day long supercooling event from Figure 3-4 which occurred during the first two weeks of February. The average air temperature during this event was -14.1ºC (Figure 3-5b) and the longwave was the dominant heat flux during this event with an average value of -89.3 W/m^2 compared to the average sensible heat flux of -66.4 W/m^2. The evaporative heat flux was insignificant until the air temperature rose above freezing near the end of the event (Figure 3-5c). This period of significant evaporative heat flux is also a period when wind speed are significantly greater than the average wind speed of the period (average wind speed from February 11th – February 13th 2017 was 2.54 m/s compared to the preceding wind speed average of 0.34 m/s from February 1st – 11th). From Figure 3-5d, it can be seen that the event had classical supercooling behaviour with a T_P of -0.044ºC. The daily fluctuations in Q_n had a gradually decreasing trend during the first 5 days of the event, then the trend was reversed to increase gradually until the end of the supercooling event. Approximately 7.5 hours after the start of the event, the water temperature maintained a residual temperature of around -0.008ºC for a week despite large

fluctuations in Q_n during this period. From Figure 3-5d, cycles between positive peaks from ~30 to 180 W/m^2 at mid-day to negative heat fluxes on the order of -100 to -350 W/m^2 at night are not reflected in a significant change in the water temperature.

3.4.2 Statistics and distribution of heat fluxes

Table 3-5 presents the average environmental conditions during supercooling events. The air and water temperature are the most important of these parameters, as they are used in the longwave, sensible and evaporative heat fluxes, with wind speed being important for both sensible and evaporative heat fluxes. The cloud cover fraction has a significant impact of the longwave heat flux, while relative humidity is only considered for the evaporative heat flux. The event averaged water temperatures ($\overline{T_w}$) in the two rivers are very similar with both having a mean $\overline{T_w}$ of -0.010ºC and with medians that only differ by of 0.001ºC. There was a significant difference in event averaged air temperatures, $\overline{T_a}$, with mean values of -10.9 ºC and -4.72 ºC on the PR and NSR, respectively. The event averaged cloud fraction ($\overline{n}$) had means of 0.789 and 0.680 and medians of 0.893 and 0.686 on the NSR and PR, respectively. Event averaged wind speeds ($\overline{V_z}$) were quite low and skewed on the NSR where they ranged from 0 to 2.13 m/s, and had median and mean values of 0.010 m/s and 0.237 m/s, respectively. The wind speeds were significantly larger but still quite low and skewed on the PR where $\overline{V_z}$ ranged from 0 to 4.93 m/s and had a median and mean of 0.592 m/s and 0.941 m/s, respectively. The event averaged relative humidity ($\overline{RH}$), had a mean of 88.9 % and 77.0 % on the NSR and PR, respectively.

The average of the heat flux components and net heat flux during each supercooling event was calculated. In addition, the integral of the net heat flux over each event E_{net} (J/m^2) defined as the total thermal energy transferred out of the river over the duration of a supercooling event was also calculated. One of the means to estimate the total ice production during supercooling is to determine the integral of the net heat flux on the water column and divide by the latent heat of fusion (Osterkamp 1978). Table 3-6 presents the statistical properties of these event averaged heat fluxes as well as E_{net}. Comparing the two rivers, longwave is the dominant negative heat flux with mean of event averaged heat flux of -45.1 and -74.7 W/m^2 for the NSR and PR, respectively. The sensible heat flux serves as a secondary negative heat flux on both rivers with mean event averaged values of -20.6 and -58.1 W/m^2 for the NSR and PR, respectively. Evaporative heat flux is minimal

with the event averaged evaporative heat flux averaging -1.19 and -5.98 W/m^2 for the NSR and PR, respectively. The shortwave radiation heat flux tended to be higher on the PR than the NSR, with a mean event averaged shortwave heat flux of 63.7 W/m^2 on the PR compared to 28.9 W/m^2 on the NSR. Comparing the mean event averaged heat fluxes of the PR to the NSR, the event averaged longwave, sensible, and net heat flux are 1.7, 2.8, and 2.0 times larger on the PR than the NSR. Considering the larger magnitude of net heat flux, it is unsurprising that the mean net energy on the PR (-1.72×10^7 J/m^2) is 7.4 times larger than on the NSR (-2.34×10^6 J/m^2).

Boyd et al. (2022) analyzed the distribution of the start and end time of events throughout the day, and suggested that the solar radiation cycling was a driver of supercooling events with freeze-up events starting more often during the decline of solar radiation in the afternoon and evening, and ending around mid-day when solar radiation would be at its peak. Figures 3-6 and 3-7 show the distribution of the distribution of the start and end time of day of supercooling events for the NSR and PR, respectively, along with the average hourly value of each heat flux component and the resulting net heat flux. Starting with the NSR start time (Figure 3-6a), there is a peak in supercooling events starting between 3 PM and 5 PM compared to the rest of the day with 34 % of events observed on the NSR starting in this 2 hour period. The distribution of end time of supercooling events (Figure 3-6b) show 67 % of NSR events ending between 10 AM and 4 PM. Comparing these peaks in start and end times to the distribution of heat fluxes (Figure 3-6c), the peak in event start frequency aligns with the decline in shortwave radiation an hour after the average net heat flux drops below zero. Likewise, the uptick in events ending correspond to the period when shortwave radiation is significant enough to have a positive net heat flux. When the average net heat flux becomes a negative heat flux at 2 PM, the number of events ending in that hour is only one more (4 events) then the other maximum value for events ending in the night (ex between 10 and 11 PM). Figure 3-7 has a smoother but comparable distribution to Figure 3-6 in terms of start and end times, with 36 % of events starting between 3 PM and 7 PM (Figure 3-7a), right as the average net heat flux changes to a negative heat flux (Figure 3-7c). Comparing the end time of events (Figure 3-7b) to the heat flux distribution, 67 % of PR events ended between 9 AM and 3 PM, in the period where the shortwave radiation increases to a peak and then starts to decline.

Comparing the average heat flux distributions (Figures 3-6c and 3-7c), it is of interest to note that the average value for the cooling heat fluxes (longwave, sensible, and evaporative) are fairly

constant, with the shortwave heat flux changing the shape with the diurnal cycling. On the PR Q_n was positive on average for 10 hours a day, between 8 AM and 6 PM, but only for 7 hours a day, between 10 AM and 5 PM, on the NSR. This is likely to do with the fact that since the supercooling occurred all winter, the average shortwave radiation reflects the increasing solar radiation from the winter solstice onwards, while the NSR distribution only reflects the declining solar radiation prior to the winter solstice.

Table 3-7 summarizes the statistics of the supercooling parameters of the 190 events used in this study. The average values of T_P and D_P are comparable between the two rivers with a difference less than 30% between the mean values. The durations of supercooling events were significantly different between the two rivers, with the NSR having a mean duration of 16.8 hours compared to the PR with a mean duration of 41.0 hours. This ratio of 2.4 is reflected in *CDMS* values with a mean value of 10.6 and 21.2 ºC·minutes for the NSR and PR respectively (ratio of 2.0). Comparing the mean CR_P values for each river (-2.08×10^{-4} and -5.16×10^{-4} ºC/minute respectively for the NSR and PR), shows that that the PR tends to cool at a faster rate than the NSR. The longer duration of events on the PR would also contribute to the larger E_{net}, as these events would have a longer period to integrate over.

Frequency distributions of the event averaged net heat flux ($\overline{Q_n}$) for all supercooling events as well for each river are shown in Figure 3-8. Figure 3-8a shows a wide range of values from a minimum of -264 W/m^2 to a maximum of 201 W/m^2. On the PR 39 % of the events had $\overline{Q_n}$ below -100 W/m^2, while on the NSR only 12.1 % did. Across both rivers, roughly 64 % of all events occurred during periods when $\overline{Q_n}$ was between -100 W/m^2 and 0 W/m^2 (Figure 3-8a), with 87 % of NSR supercooling events (Figure 3-8b) and 53 % of PR supercooling events (Figure 3-8c) falling into this category. It is interesting to note that ~14 % of supercooling events (6 events and 21 events on the NSR and PR, respectively) were calculated to have been observed during a period where the average Q_n was positive. These events consequentially also have a positive E_{net}.

Figure 3-9 presents a log-log scatter plot of the *CDMS* and E_{net} for all 163 events where $E_{net} \leq 0$ J/m^2 to determine if there any correlation between the parameters. From the figure it can be seen that there is a roughly linear trend of *CDMS* increasing in magnitude with increasing magnitude E_{net}. The line of best fit has the equation,

$$log_{10}(CDMS) = 0.8815 \cdot log_{10}|E_{net}| - 4.8887 \quad (3.16)$$

with 11 events (~6.8 % of the data) outside the 95 % prediction interval. The correlation between the two parameters was calculated to be R^2 = 0.86.

An analysis was done to determine the dominant negative and positive event averaged heat fluxes. Pie charts showing the percentage of events when the different heat flux components were dominant during supercooling events are plotted in Figure 3-10. From Figure 3-10a, it can be seen that Q_{lw}, Q_s, and Q_e were the dominant negative heat flux in 80.0 %, 18.4 % and 1.6 % of the time, respectively across both rivers. On the NSR Q_{lw} and Q_s were the dominant negative heat fluxes for 91.4 % and 8.6 % of the events, respectively (Figure 3-10c). While on the PR $\overline{Q_{lw}}$ was the dominant negative heat flux for only 75.0 % of the events and Q_s was dominant for 22.7 % e events (Figure 10e). Figures 3-10c and 3-10e also show the only events in which Q_e was the dominant negative heat flux were observed on the PR for 2.3 % of the events observed.

In Figure 3-10b Q_{sw} is shown to be the most dominant positive heat flux during 97.4 % of all supercooling events on both rivers. On the NSR it was the dominant positive heat flux for 100 % of events (Figure 10d). Figure 3-10e shows that on the PR the dominant positive heat flux was Q_{sw}, Q_s, and Q_e for 96.2 %, 3.0 % and 0.8 % of the events.

3.4.3 Linear correlation and multiple linear regression

To determine if any linear equations could be developed between the calculated heat fluxes and any supercooling parameters, both linear and multiple correlations were investigated using MATLAB (2021). The R^2 values for the linear correlation are summarized in Appendix A. In all cases, the correlations were found to be relatively weak with the largest R^2 value of the linear correlation being -0.35 between T_P and principal supercooling shortwave heat flux. A scatter plot of principal supercooling net heat flux and T_P is shown in Figure 3-11 visualizes the linear correlation between the two parameters and emphasizes that there is little correlation between the net heat flux cooling the water it's lowest temperature.

Multiple linear regression models were developed between the supercooling parameters observed in this study and the average heat flux components for the supercooling event using MATLAB *fitlm* function. The summary for each linear regression model coefficients and the model assessment are shown in Appendix A. The multiple linear regression model calculated a maximum R^2 value of 0.188 for T_P in response to the principal supercooling heat fluxes, meaning that the most accurate of these models could only explain at ~19 % of variability in supercooling parameter. In the case D_P and CR_P, a constant value estimate was deemed to be a more accurate estimate than the model. Comparing the model intercept values of these parameters gives an estimated D_P of 4.06 hours with a CR_P of -2.88×10^{-4} ºC/minute. This is a longer duration and slower cooling rate than the mean value across both rivers, but predicts a T_P of -0.070 ºC, significantly colder than the mean value of -0.022 to -0.023 ºC for either river (Table 3-7).

3.5 Discussion

3.5.1 Heat flux dynamics during supercooling

When determining the dominant negative heat fluxes for all events (Figure 3-10), the longwave heat flux was the primary cooling heat flux for 80.0 % (152 events), with sensible heat flux the next most dominant heat flux at 17.9 % (35 of 190) of events. For events dominated by longwave, the mean event averaged longwave and sensible heat fluxes were -64.6 and -35.3 W/m^2, respectively, while sensible dominated events have mean event averaged longwave and sensible heat fluxes of -75.8 and -102 W/m^2, respectively. This is a ~11 W/m^2 difference in mean longwave heat flux between the two sets and a ~67 W/m^2 difference in mean sensible heat flux. Looking at the air temperature between the two sets, longwave dominated events had a mean event averaged air temperature or -7.93 ºC, which is between the mean event averaged air temperature of the two rivers in Table 3-7. For sensible dominated events, the mean event averaged air temperature was -14.8 ºC, below the mean event averaged air temperature of either river.

Across the NSR and PR in 2016-2017, net heat flux was observed to range between -354 to +548 W/m^2 during supercooling events. The event averaged net heat flux on the PR ranged between -264 and +176 W/m^2, with a mean of -75.0 W/m^2, while on the NSR it ranged between -197 to 201 W/m^2, with a mean of -38.0 W/m^2. The PR tended to be exposed to colder air temperatures due to the higher latitudes. In addition, upstream dam regulation on the PR delayed the ice cover

formation and exposed the PR study reach to more severe weather conditions of mid winter compared to late fall and early winter of freeze-up on the NSR. The differences between the study reaches reaffirms that the heat flux dynamics of a river is highly site specific, and that site assessment of study reach is required for any approximation of the energy budget for a specific study reach.

The uncertainty of the calculated heat flux components was estimated to range from ±10 to ±20 W/m^2 for an estimated net heat flux uncertainty of ± 31.2 W/m^2 (Table 3-4) as the root sum of squares of the estimated heat flux components of the individual heat fluxes. The uncertainty in Q_n is primarily due to uncertainty in the sensible and longwave heat fluxes at ± 20 and ± 16 W/m^2, respectively based on the uncertainty of the empirical equations used. Shortwave heat flux is relatively small at ± 10 W/m^2 due to being directly measured, while evaporative heat flux has a larger uncertainty at ± 14.7 W/m^2, but only a significant heat flux when air temperatures are above zero. The most straightforward way to reduce uncertainty is to perform direct measurements of all radiative heat fluxes using radiometers deployed over the water surface (away from any shading and influence from land, trees, or structures). Blonquist (2009) showed that longwave radiometers can be accurate to within 3.2 % depending on the sensor, though in more varying conditions, accuracy of 10 to 20 % may be more likely. McFarlane and Clark (2021) emphasize that if such measurements are not possible, then direct measurements of incoming radiation should be measured and outgoing radiation should be calculated from field parameters. Direct measurements of sensible heat fluxes are possible (Buke et al. 2002), but require additional instrumentation that would be difficult to deploy.

In Figure 3-7 it was shown that ~14 % of events (27 events) had a positive $\overline{Q_n}$, which would imply that the weather conditions were warming the water when it was supercooling, an apparent contradiction. The behaviour of Q_n of these 27 events can be divided into two categories as summarized in Table 3-8. In the first category there was a small negative Q_n at the start of the event followed by a larger magnitude positive Q_n, resulting in an overall positive heat flux, as shown in Figure 3-12. These events had average peak supercooling of -0.021 ºC and duration of 6.0 hours compared to -0.023 ºC and 33.6 hours for all events. Therefore, they were of relatively short duration but were approximately average in terms of peak supercooling. The positive net heat flux was caused by a rapid increase in Q_n during the event. The mean event averaged Q_n was

16.8 W/m^2, roughly half of the expected uncertainty, thus many of these events could have actually had a negative event averaged Q_n. For the 13 events in the second category (Table 3-8), the net heat flux during the entire event generally stayed positive for the whole event. These 13 events on averaged were short in duration, 0.70 hours and had small peak supercooling magnitudes of -0.005 °C. The most likely explanation for these events is that they are persisting supercooling events that had advected from upstream and were being gradually warmed rather than an active supercooling event at the time of observation. This explanation seems supported by Figure 3-13, where a positive $\overline{Q_n}$ event follows closely after a series of events with negative $\overline{Q_n}$, and has two instances of supercooling occur after it less than 10 minutes in duration.

As mentioned previously, net energy can be used to estimate the scale of ice production in the water column as the energy removed from the water by a surface heat flux is balanced by frazil ice produced (Osterkamp 1978). According to Osterkamp (1978), this can be approximated by,

$$M_{ice} = \frac{Qt}{h_f} \tag{3.17}$$

where M_{ice} is the mass of ice per surface area of water, Q is the constant surface heat flux, t is the time elapsed and h_f is the heat of fusion of water at 3.34x10^5 J/kg and d is the depth of water. Equation 17 can be converted to a volumetric fraction of ice to water f by dividing the above equation by the density of ice ρ_{ice} = 916 kg/m^3. An additional assumption of well mixed water column assumes that a unit depth d is representative of the entire water column, or that the concentration of suspended frazil does not change significantly with depth resulting in,

$$f = \frac{Qt}{\rho_{ice} h_f} d \tag{3.18}$$

The final step to make E_{net} applicable to the ice production equations as written is to note that Qt is the same as the time integration of the net heat flux Q_n that is not a constant value, enabling the final form of the equation to be,

$$f = \frac{-E_{net}}{\rho_{ice} h_f} d \tag{3.19}$$

where a -E_{net} term is used to note that a negative net energy leads to ice production, while a positive net energy leads to melting ice.

Using the median and mean values of E_{net} of -1.32x10^{6} and -1.27x10^{7} J/m^{2} (Table 3-6), assuming a unit, and noting that a negative value would denote ice production, this equates to an ice concentration range of 4.31x10^{-3} to 4.15x10^{-2} m^{3}/m^{3}. Compared to the field measurements of ice concentration from 1.8x10^{-5} m^{3}/m^{3} (McFarlane et al., 2019) to 1x10^{-4} m^{3}/m^{3} (Marko et al., 2015), these values are approximately two to three orders of magnitude larger. One possible reason for this method does not account for frazil ice flocculating and rising out of suspension to form frazil pans, as well as the formation of other forms of ice such as anchor ice, border ice, and skim ice. In addition, the estimated net energy may not account for presence of surface ice, additional shading from topography, or other heat sources and sinks that would alter the effective net energy lost from the water during the supercooling event.

3.5.2 Comparison to literature

The results of this study agree with previous observations that the shortwave and longwave heat fluxes are the most significant surface heat source and sink, respectively (Brown 1969;Evans et al. 1998;Webb and Zhang 2004; McFarlane and Clark 2021). Hannah et al. (2004) study on salmon spawning river is Scotland reported that the sensible heat flux was the most significant heat source due to low shortwave radiation heat fluxes and the positive air temperatures during the study. This was not the case in the current study because during most supercooling events air temperatures were below zero and thus the sensible heat flux was negative, i.e., cooling the water (Table 3-7). However, for the four supercooling events where sensible heat flux was the dominant positive heat flux (Figure 3-10), the event averaged air temperature ranged from 3.80 to 7.60 ºC, which is comparable to or greater than the mean air temperature of 3.17 ºC observed by Hannah et al. (2004). The conditions where sensible heat flux were the dominant positive heat flux tended to contribute to the evaporative heat flux to become the dominant negative heat flux; two of the three evaporative cooled supercooling events had sensible heat flux as the dominant heat source. Therefore, it is possible that when the non-radiative heat fluxes dominate the surface energy budget (above freezing but cool air temperature, higher wind speeds, etc.) supercooling

events can be generated. However, more studies with smaller uncertainty range would be needed to confirm the significance of these observations.

This study supports previous ones that showed that the net longwave radiation was the dominant cooling heat flux during supercooling events (Richard and Morse, 2008; Richard et al., 2015; McFarlane and Clark, 2021). While not quantified for all events, the select time series presented by Richard et al. (2015) showed the overnight behaviour of the net heat flux to be roughly parabolic with peak values between -200 and -250 W/m^2, matching their conclusion that most supercooling events occurred on nights where Q_n dropped below -200 W/m^2. Compared to the current results, Richard et al. (2015) observed net heat fluxes of order of twice the magnitude of the event averaged net heat flux observed during supercooling across the NSR and PR (mean event averaged net heat flux of -63.7 W/m^2). Richard et al. (2015) did not report average values of air temperatures, but Richard and Morse (2008) reported that air temperatures were $\leq$ -5.4ºC for the majority of observed supercooling events during the same measurements season.

McFarlane and Clark (2021) reported the event averaged heat fluxes for the six events they observed on the Dauphin River between -37.4 to and -127.9 W/m^2 (mean event averaged net heat flux of -84.9 W/m^2), with event averaged air temperatures between -1.7 to -8.8ºC. The mean event averaged heat flux is greater than the -63.7 W/m^2 for this study (Table 3-6), though this would be in part due to the small number of events. It is notable from the two above mentioned studies and the current study that air temperatures do not need to be significantly below freezing in order for supercooling to occur. The PR is an outlier from these observation sets (mean event averaged air temperature of -10.9 ºC) in part due to the upstream dam regulation artificially supressing supercooling immediately downstream of the dam, which requires more severe weather conditions to induce supercooling in the water column by the time it reaches the study reach.

3.5.3 Correlating supercooling parameters with heat flux parameters

From the linear correlation and multiple linear regression analysis, it is apparent that no correlation can be found between the supercooling parameters and the event averaged heat fluxes. The multiple linear regression showed that the attempted models only explain at best 19 % of the variability observed in the supercooling parameters. From this, it can be concluded that the relationship between the supercooling parameters and the heat flux components is probably

much more complex. Boyd et al. (2022) hypothesized that ice production may play a significant role in enabling the water column to remain at an essentially constant residual temperature for up to a week in duration (such as shown in Figure 3-5). Considering the range of heat fluxes observed during this event (-344 to 336 W/m^2), it seems clear that the residual temperature is maintained through significant variations in the net surface heat flux. While additional studies are required to confirm the idea of ice production buffering the net heat flux and extending supercooling events, this study does lend credibility to the idea.

3.5.4 Potential role of ice production in supercooling energy budget

Laboratory observations of supercooling with a constant heat flux result in a 'classical' supercooling event where the water cools to a peak temperature before rising to a residual temperature at which point the cooling heat flux is balanced by the latent heat released from frazil ice production. While the field observations occur under much more dynamic heat fluxes, it is reasonable to assume that this balance is still being maintained while the river remains at a constant residual temperature. In order for ice production to respond at a rate comparable to the net heat flux, sufficient seed crystals must be present in the water. A low seed crystal concentration would mean that a cooling heat flux would primarily cool the water temperature, with minimal ice production. The initially small ice production would provide more seed crystals for accelerated ice production rate until the water temperature starts rising. A similar hypothesis for the connection between seed crystal concentration and peak supercooling temperature is proposed by McFarlane et al. (2019). On November 25, 2016 at 7:20 PM, they recorded a supercooling event on the NSR that reached a T_P of -0.145ºC. McFarlane et al. (2019) noted that the previous day would have been warm enough to have melted all sources of seed crystal in and around the water. Since this event happened in the same time frame as presented in Figure 3-2 (within a few minutes of a similar event in Figure 3-2e), it can be seen from Figure 3-2 that the water temperature during the day reached a peak of 0.05ºC, with an air temperature above zero for most of the day, supporting this theory. As the frazil ice production increases, the available seed crystals increases, which in turn enables for a greater proportion of the heat flux to be balanced by the heat of fusion. Once the frazil production outpaces the cooling heat flux (after peak supercooling temperature is reached), the limiting condition would be the cooling heat flux rather than the seed crystals. Smaller magnitude of negative heat fluxes during peak supercooling will have sufficient seed crystal to be

fully balanced by ice production. Greater magnitude negative heat fluxes will have a high enough seed crystal concentration that frazil production would rapidly increase, minimizing the resulting temperature change.

The presence of suspended frazil would also increase the required energy to raise the water temperature from the residual and end the event (Richard et al. 2015). Using the range of frazil concentrations from field measurements of $1.8x10^{-5}$ to $1x10^{-4}$ m^3/m^3 and assuming the residual temperature is around -0.01 ^oC (mean water temperature during supercooling for both rivers), the required energy to end a typical supercooling event would increase from 13 to 73 %, respectively. This is comparable with observations by Brown et al. (2006), where the delay in rising water temperature to rising air temperature was attributed to the presence of ice and meltwater in the stream dampening the water temperature temporarily before the water temperature mirrored the air temperature fluctuations. In conditions where the net heat flux is only positive for a few hours of the day, the melting of ice within the water column (suspended frazil, anchor ice, or surface ice) could be sufficient to buffer the warming heat flux until the net heat flux goes negative, when frazil ice production can resume and regenerate the lost ice mass.

A second impact ice production can have on extending supercooling event duration is by the development of surface ice in the form of frazil pans and border ice to insulate the water column below. Richard et al. (2015) modelled this impact on the water-air interface as inversely proportional to the surface ice concentration, directly measured the surface ice concentration from images collected from an onshore camera. While the presence of an ice cover would dampen the magnitude of all heat fluxes at the water surface, since winter heat fluxes are positive for only a few hours each day, insulation has a greater impact on the periods of positive heat flux rather than negative heat fluxes. Alongside the resumption of ice production of ice production outside the periods of positive heat fluxes restoring a portion of ice mass, a moving ice mass can dampen small periods of positive heat fluxes for multiple days. The dampening effect of the surface pans would also help explain the resilience to drastic changes to negative heat flux, as the pans would dampen the overall heat flux and regulate the air temperature with in-situ ice growth within the surface ice.

3.6 Conclusions

This study presents an analysis of the surface heat flux components estimated during the supercooling events observed on the NSR and PR for the 2016-2017 season. The shortwave and longwave radiative heat fluxes, as well as the sensible and evaporative heat flux components at the water surface were included in the analysis. Due to the significant differences in climatic conditions between the two study sites, the magnitude of net heat flux during supercooling events was found to be significantly different between the two rivers with an event averaged net heat flux of -38.0 and -75.1 W/m^2 for the NSR and PR, respectively.

The radiative heat fluxes were found to make the most significant contribution to the energy budget, with longwave radiation being the dominant negative heat flux and shortwave radiation being the dominant positive heat flux for 80.0 % and 97.4 % of supercooling events, respectively. Sensible heat flux was the next most significant heat flux and was the dominant negative and positive heat flux for 18.4 % and 2.1 % of events, respectively. Sensible heat flux tended to become the dominant negative heat flux at significantly colder air temperatures since the event averaged air temperature for sensible dominated events was -14.8 ^oC compared to the overall average of -8.96 ^oC). The evaporative heat flux was rarely a significant heat flux except when the air temperature was positive and the radiative heat fluxes were small. The daily distribution of the heat fluxes during supercooling events when compared to the distribution of supercooling start and end time are in line with the conclusion by Boyd et al. (2022) that the trend in supercooling start and end time coincide with the diurnal cycling of the shortwave radiation and the resulting fluctuations in the surface net heat flux.

No linear correlation was found between any of the supercooling event parameters and the event averaged heat fluxes. However, visual observations of the time series of supercooling events together with their corresponding surface heat components, do suggest that a change in sign in the net heat flux for an extended period of time results in a significant change in the frequency of supercooling events starting and ending. The change in sign from positive to negative heat flux in the afternoon sees a spike in frequency of supercooling events starting after an hour of a cooling heat flux, while events tended to end with greater frequency during the period when strong shortwave radiation heat flux changed the net heat flux to positive. A strong correlation was found between *CDMS* and net energy exchange at the water surface during the supercooling

event, which may mean that *CDMS* could be used to estimate ice production. However, studies directly measuring ice production along side *CDMS* would be required to confirm this hypothesis.

Future studies that consider the surface energy budget should consider using instrumentation to directly measure the heat flux components rather than approximations through empirical equations. The uncertainties associated with the estimation the different heat flux components can be reduced with more direct measurements of radiative heat flux components. In addition, measurements of surface ice concentrations would also improve estimation of the effective surface heat flux when analyzing the river energy budget.

This study analyzed the surface energy budget during supercooling events, using a significantly larger data set of events than analyzed previously. This study also confirmed the relationship between the distribution of the start and end times of supercooling events and the average behaviour of the surface heat flux to be related to the diurnal cycling of the shortwave heat flux. This study also emphasized the dominance of radiative heat fluxes for the supercooling events, with sensible heat flux only becoming the dominant cooling heat flux when the air temperature is very cold. The non-radiative heat fluxes were primarily found to play a secondary role to the heat budgets in the events studied, but the few events where they were the dominant positive and negative heat fluxes suggest that specific sites may need to take them into consideration as well. This understanding of the impact of the energy budget cycling and the impact of specific site conditions on supercooling events can inform modeling and forecasting of the evolution of the local ice regime. Energy budget studies are a necessary component for river ice process models to improve capabilities to predict the evolution of a river's ice regime during freeze-up using local meteorological forecasts. Studies that improve understanding of the river response to dynamic weather conditions would provide valuable information to improve the accuracy of these models.

3.7 Chapter 3 Supplementary materials

Chapter 3 Tables

Table 3-1: Summary of the study reach properties for North Saskatchewan NSR, and Peace PR Rivers (McFarlane et al. (2017))

River	Drainage Basin Area at study site [km^2]	Average Annual Flow Rate [m^3/s]	Average Slope	Average Width [m]	Average Depth [m]
NSR	$5.67x10^3$	220	0.00035	136	1.40
PR	$1.30x10^5$	$1.59x10^3$	0.00025	227	2.56

Table 3-2: Summary of equipment deployed. (Ruskin (2021); Onset (2021))

Parameter	Make	Model	Accuracy	Time Step
Water Temperature	Ruskin	RBR Solo T	±0.002 °C	1 minute
Wind Speed	Onset	Wind Speed and Direction Set Smart Sensor S-WSET-B[a]	±1.1 m/s	10 minutes
Air Temperature		12-bit Temperature/ Relative Humidity Smart Sensor S-THB-M002[b]	±0.21 °C	10 minutes
Relative Humidity			±2.5 % from RH 10 % - 90 % ±5 % outside range	10 minutes
Solar Radiation		Silicon Pyranometer Smart Sensor S-LIB-M003[c]	±10 W/m^2	10 minutes

Table 3-3: Straight-line distances between supercooling observation sites and weather stations

River	Site	U of A Weather Station – Site (km)	ECCC Weather Station – Site (km)
NSR	Genesee	47	52
	River Ridge	12	16
	Quesnell	5	25
	Emily Murphy	1	25
PR	Sta. 293.5 km (LB, RB)	15	82
	Sta. 305.0 km (LB,RB)	7	73

Table 3-4: Estimated uncertainty for the heat flux components

Heat Flux Component	**Estimated Uncertainty (W/m^2)**	**Source of Uncertainty Estimate**
Shortwave Heat Flux	±10 (min. Q_{sw} 0)	Onset (2021)
Longwave Heat Flux	±16	König – Langlo and Augstein (1994)
Evaporative Heat Flux	±14.7	McFarlane and Clark (2021)
Sensible Heat Flux	±20.0	McFarlane and Clark (2021)
Net Heat Flux	**±31.2**	Cumulative Uncertainty

Table 3-5: Average conditions during supercooling events. These conditions include event averaged water temperature ($\overline{T_w}$), air temperature ($\overline{T_a}$), cloud cover fraction ($\overline{n}$), wind speed ($\overline{V_z}$), and relative humidity ($\overline{RH}$)

Data Set	**Statistics**	$\overline{T_w}$ **(°C)**	$\overline{T_a}$ **(°C)**	$\overline{n}$ **(fraction)**	$\overline{V_z}$ **(m/s)**	$\overline{RH}$ **(percent)**
Both Rivers (190 events)	Minimum	-0.045	-26.3	0	0	49.5
	Median	-0.008	-9.27	0.799	0.338	81.9
	Mean	-0.010	-8.99	0.713	0.726	80.6
	Maximum	-0.001	7.61	1	4.93	98.2
North Saskatchewan River (58 events)	Minimum	-0.039	-17.2	0	0	62.8
	Median	-0.009	-3.94	0.893	0.010	91.9
	Mean	-0.010	-4.72	0.789	0.237	88.9
	Maximum	-0.001	3.13	1	2.13	98.2
Peace River (132 events)	Minimum	-0.045	-26.3	0	0	49.5
	Median	-0.008	-12.0	0.686	0.592	77.9
	Mean	-0.010	-10.9	0.680	0.941	77.0
	Maximum	-0.001	7.61	1	4.93	93.5

Table 3-6: Statistics of the event averaged heat flux components during supercooling events. Heat flux components include event averaged shortwave ($\overline{Q_{sw}}$), longwave ($\overline{Q_{lw}}$),), sensible ($\overline{Q_s}$), and evaporative ($\overline{Q_e}$) heat fluxes. These components cumulate into the event averaged net heat flux ($\overline{Q_n}$), and the net energy (E_{net}) of the event

Data Set	**Statistics**	$\overline{Q_{sw}}$ **(W/m²)**	$\overline{Q_{lw}}$ **(W/m²)**	$\overline{Q_s}$ **(W/m²)**	$\overline{Q_e}$ **(W/m²)**	$\overline{Q_n}$ **(W/m²)**	E_{net} **(J/m²)**
Both Rivers (190 events)	Minimum	0	-133	-159	-55.8	-264	-1.97×10^{8}
	Median	23.4	-69.4	-43.1	-2.83	-60.4	-1.32×10^{6}
	Mean	53.1	-65.7	-46.6	-4.52	-63.7	-1.27×10^{7}
	Maximum	303	5.32	47.3	9.55	201	1.91×10^{6}
North Saskatchewan River (58 events)	Minimum	0	-103	-133	-6.61	-197	-1.88×10^{7}
	Median	15.3	-41.4	-11.0	-1.16	-30.8	-1.03×10^{6}
	Mean	28.9	-45.1	-20.6	-1.19	-38.0	-2.34×10^{6}
	Maximum	224	-14.1	-0.289	9.55	201	7.32×10^{5}
Peace River (132 events)	Minimum	0	-133	-159	-55.8	-264	-1.97×10^{8}
	Median	34.4	-75.7	-57.1	-4.13	-82.3	-2.79×10^{6}
	Mean	63.7	-74.7	-58.1	-5.98	-75.1	-1.72×10^{7}
	Maximum	303	5.32	47.3	3.80	176	1.91×10^{6}

Table 3-7: Statistics of peak supercooling (T_P), duration (D), principal supercooling duration (D_P), cumulative degree minutes of supercooling ($CDMS$), and principal supercooling average cooling rate (CR_P) for the events used in the energy budget analysis

Data Set	Statistics	T_P (°C)	D (hours)	D_P (hours)	CDMS (°C·minutes)	CR_P (°C/minute)
Both Rivers (190 events)	Minimum	-0.002	0.172	0.016	6.77×10^{-3}	-2.67×10^{-6}
	Median	-0.018	11.5	1.97	9.94	-2.13×10^{-4}
	Mean	-0.023	33.6	3.19	17.9	-4.22×10^{-4}
	Maximum	-0.106	297	79.1	134	-1.12×10^{-2}
	Std. Dev.	-0.020	58.8	7.04	27.1	-9.66×10^{-4}
North Saskatchewan River (58 events)	Minimum	-0.002	0.256	0.14	3.27×10^{-2}	-2.67×10^{-6}
	Median	-0.018	9.69	2.30	7.86	-1.10×10^{-4}
	Mean	-0.025	16.8	3.75	10.6	-2.08×10^{-4}
	Maximum	-0.106	123	48.6	49.7	-7.70×10^{-4}
	Std. Dev.	-0.025	22.7	6.64	11.5	-2.06×10^{-4}
Peace River (132 events)	Minimum	-0.002	0.172	0.016	6.77×10^{-3}	-3.42×10^{-6}
	Median	-0.018	14.4	1.74	10.0	-2.52×10^{-4}
	Mean	-0.022	41.0	2.95	21.2	-5.16×10^{-4}
	Maximum	-0.086	297	79.1	134	-1.12×10^{-2}
	Std. Dev.	-0.018	67.7	7.22	31.1	1.14×10^{-3}

Table 3-8: Net Heat Flux behaviour during events with positive $\overline{Q_n}$

Category	Behaviour Description	Number of Events		$\overline{T_P}$ (°C)	$\overline{D}$ (hours)	Min. $\overline{Q_n}$ (W/m²)	Median $\overline{Q_n}$ (W/m²)	Mean $\overline{Q_n}$ (W/m²)	Max. $\overline{Q_n}$ (W/m²)
		NSR	PR						
I	Small negative Q_n at event start; Large positive Q_n during event	3	11	-0.021	6.04	0.648	8.13	16.8	57.9
II	Strictly positive/Positive at start Q_n	3	10	-0.005	0.698	3.10	90.2	90.7	201
TOTAL		6	21						

Chapter 3 Figures

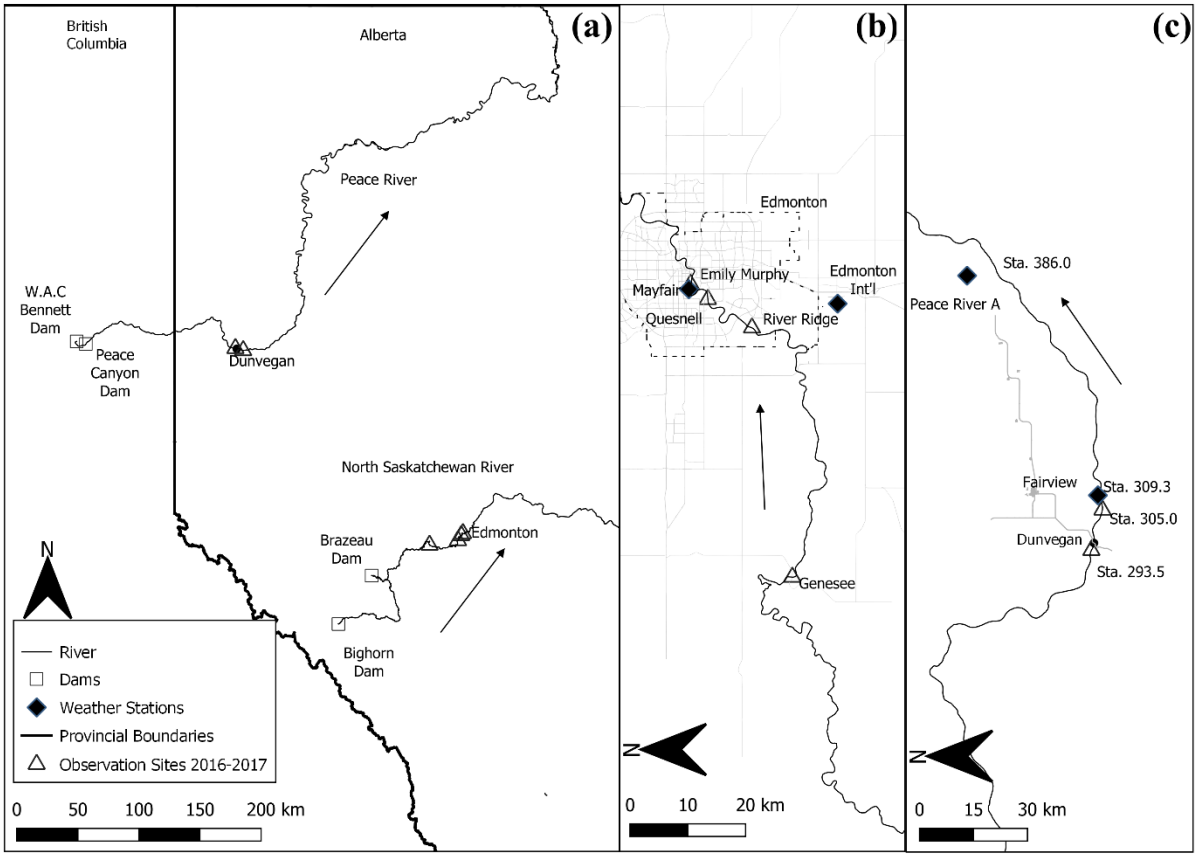

Figure 3-1: Maps showing.: (a) A map showing the geographical location of the two study reaches in Alberta. Lower maps are enlarged views of the study reaches, (b) North Saskatchewan River (NSR), and (c) Peace River (PR). Note the direction of flow is indicated by a black arrow and major roads by grey lines

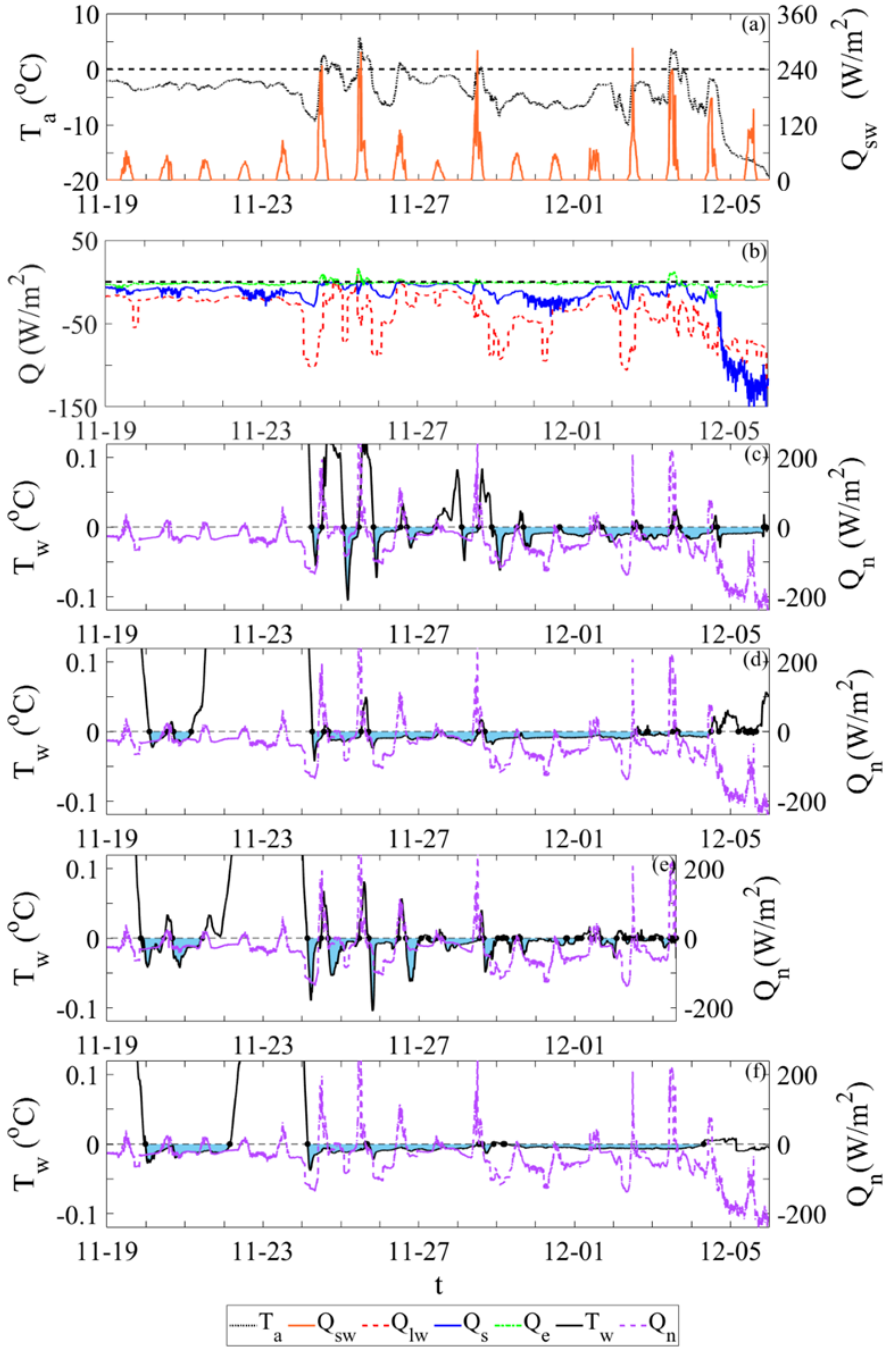

Figure 3-2: North Saskatchewan Freeze-up season: (a) Air temperature (T_a) and shortwave radiation (Q_{sw}), (b) Heat flux components. Water temperature (T_w) with shaded supercooling events and net heat flux (Q_n) for (c) Genesee, (d) River Ridge, (e) Quesnell, and (f) Emily Murphy. The sensor at Quesnell was removed from the river on December 3rd prior to freeze-up ending

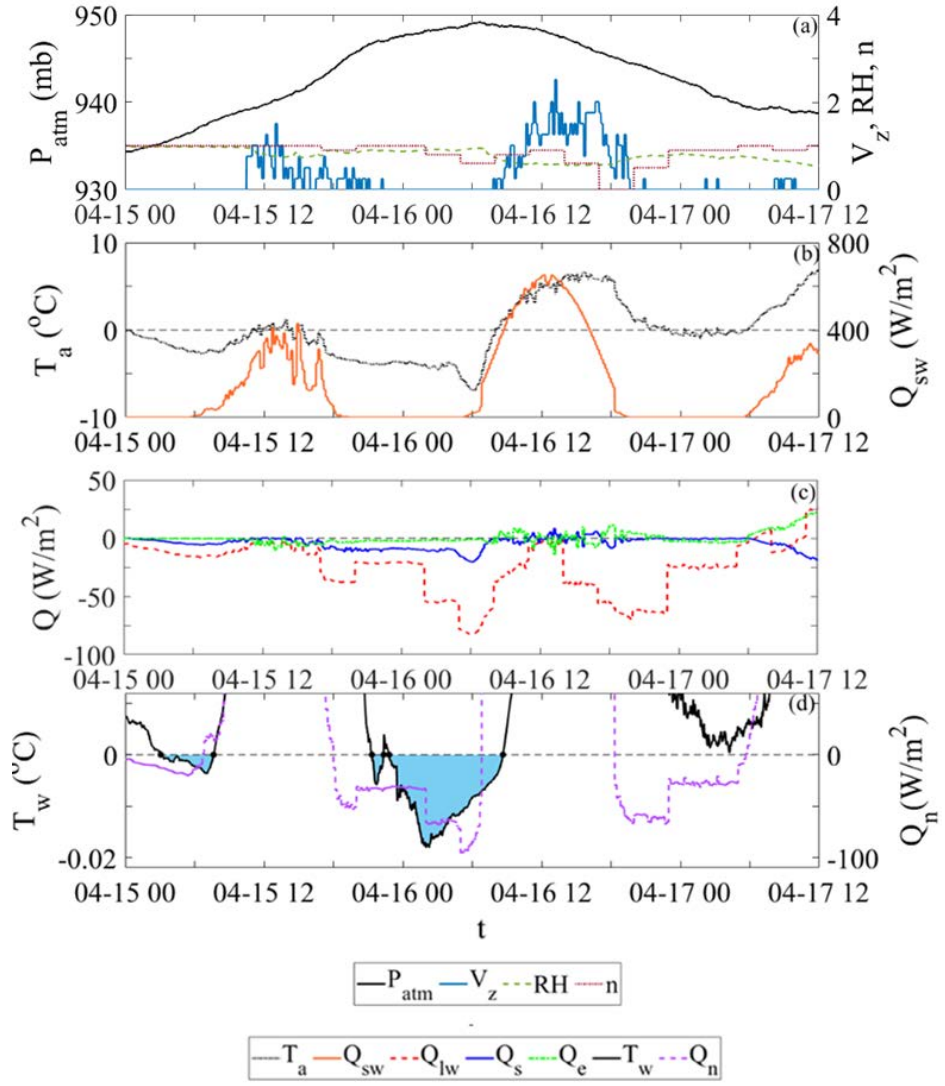

Figure 3-3: Genesee Break-up Season (a) Barometric Pressure (P_{atm}), wind speed (V_z) in m/s, relative humidity (RH) as a fraction, and cloud cover (n) as a fraction (b) Air temperature (T_a) and shortwave radiation (Q_{sw}), (c) Heat flux components (d) Water temperature (T_w) with shaded supercooling events and net heat flux (Q_n)

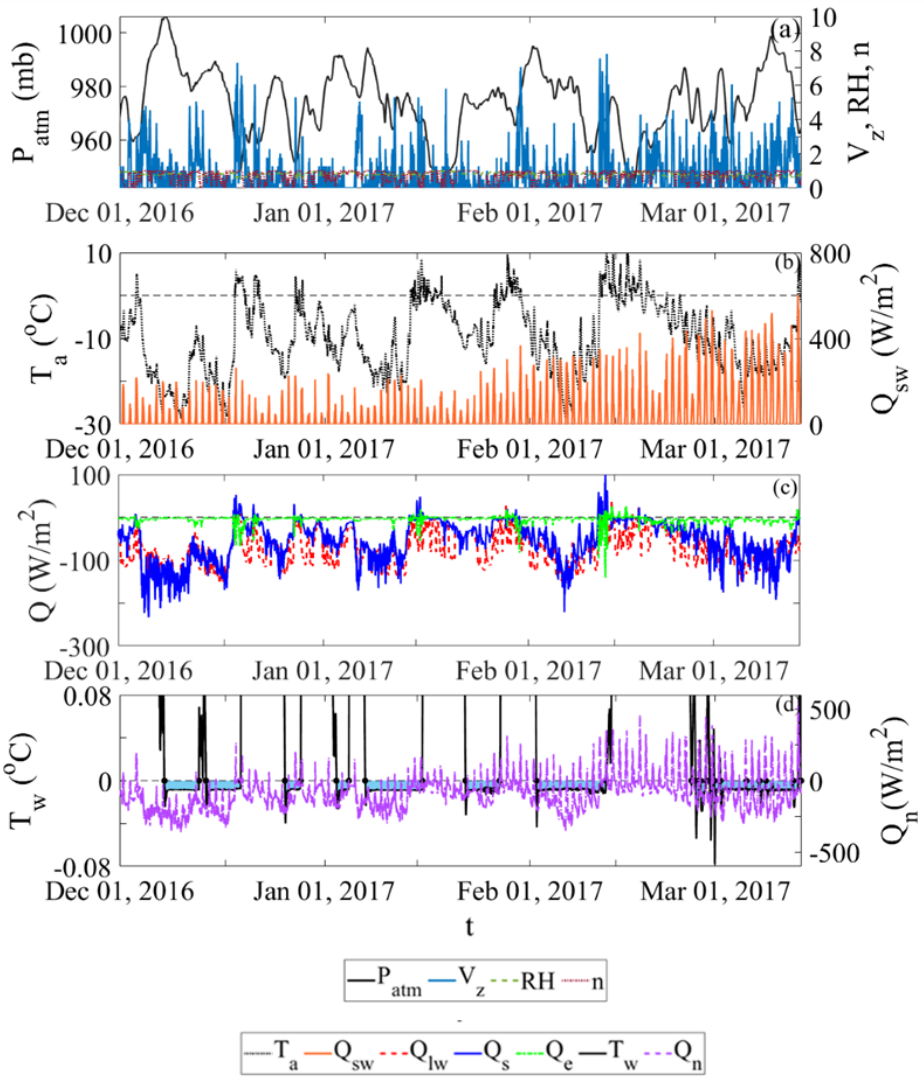

Figure 3-4: Sample time-series of Peace River Sta 293.5 left bank season (a) Barometric Pressure (P_{atm}), wind speed (V_z) in m/s, relative humidity (RH) as a fraction, and cloud cover (n) as a fraction (b) Air temperature (T_a) and shortwave radiation (Q_{sw}), (c) Heat flux components (d) Water temperature (T_w) with shaded supercooling events and net heat flux (Q_n)

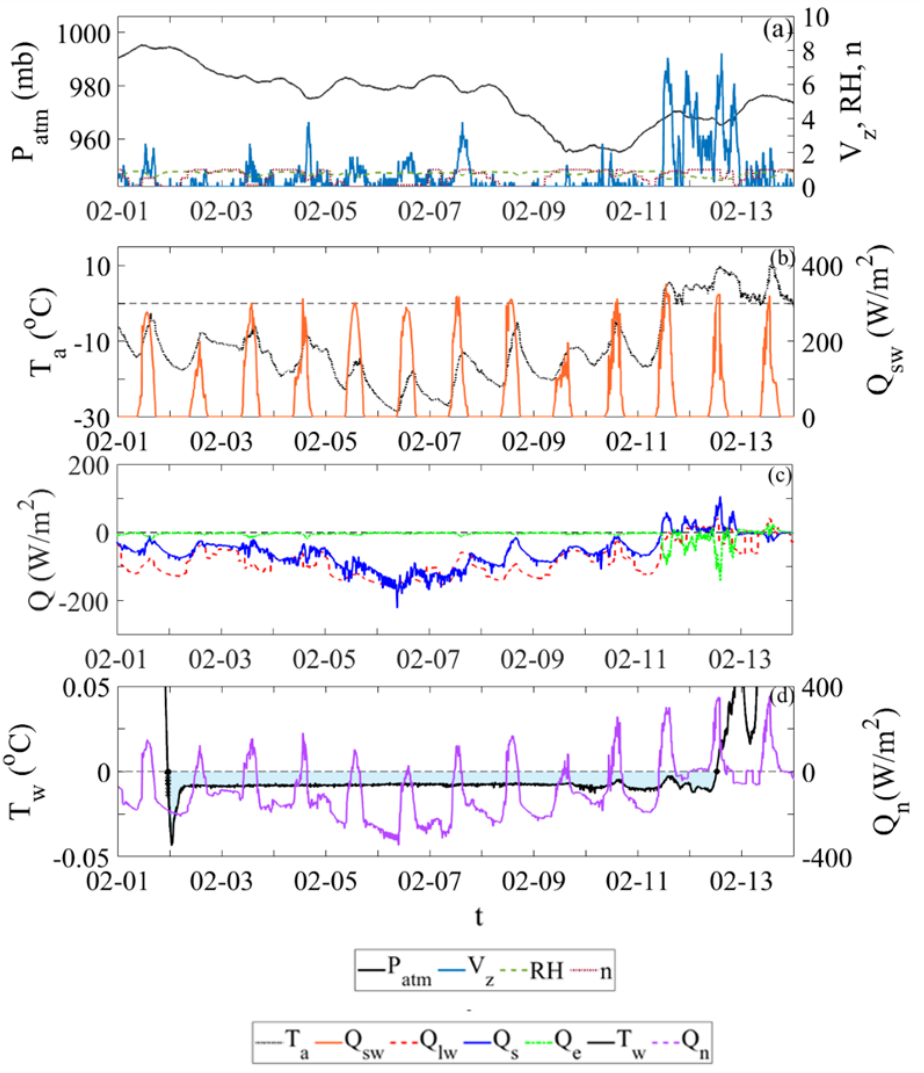

Figure 3-5: Extended event on the Peace River at Sta. 293.5 km on the left bank (a) Barometric Pressure (P_{atm}), wind speed (V_z) in m/s, relative humidity (RH) as a fraction, and cloud cover (n) as a fraction (b) Air temperature (T_a) and shortwave radiation (Q_{sw}), (c) Heat flux components (d) Water temperature (T_w) with shaded supercooling events and net heat flux (Q_n)

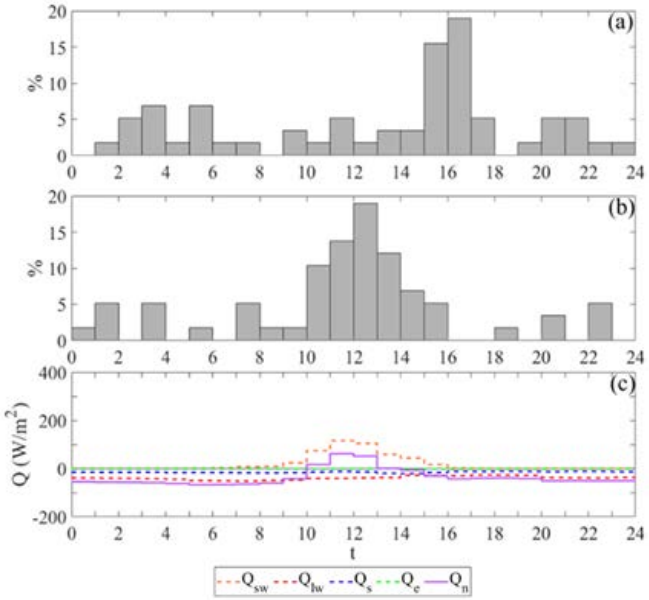

Figure 3-6: Distribution of (a) Start Time and (b) End Time of supercooling events on the NSR over the course of the day. (c) Average heat fluxes throughout the period of supercooling observations for the NSR events

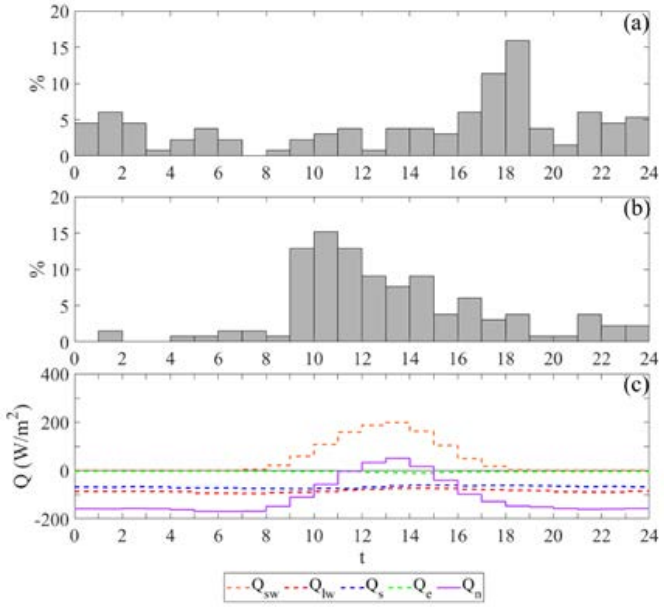

Figure 3-7: Distribution of (a) Start Time and (b) End Time of supercooling events on the PR over the course of the day. (c) Average heat fluxes throughout the period of supercooling observations for the PR events

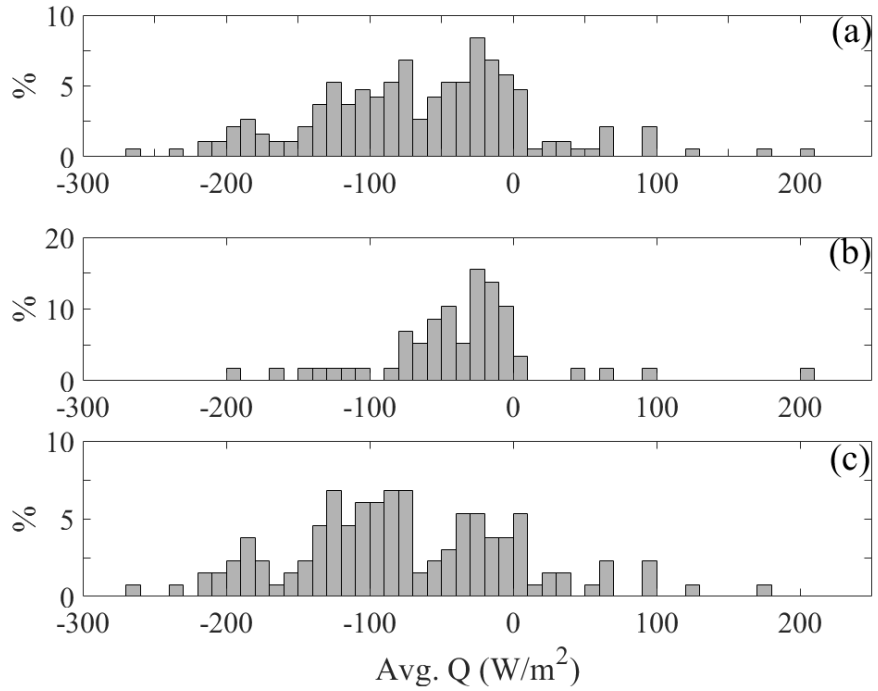

Figure 3-8: Frequency distributions of event averaged net heat fluxes during supercooling events on (a) Both Rivers, (b) NSR and (c) PR

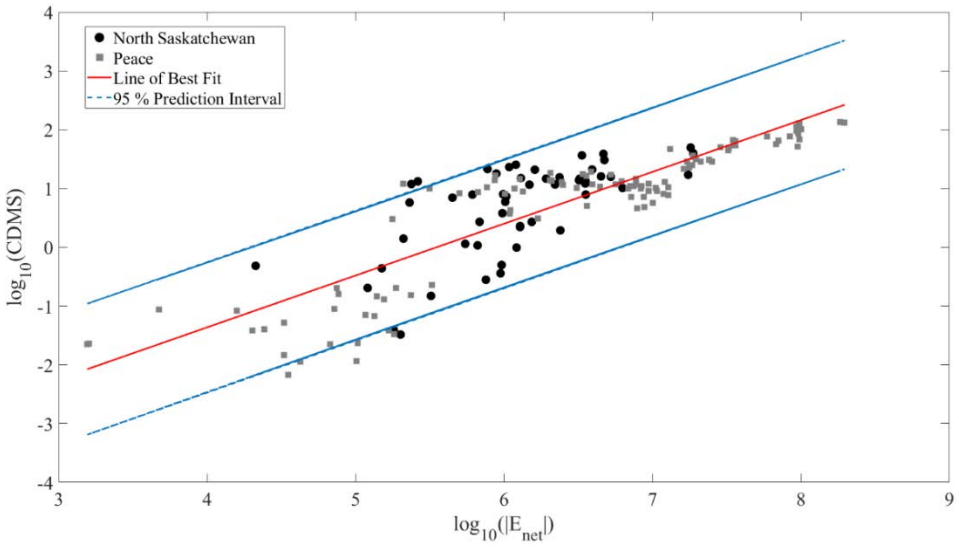

Figure 3-9: Log-log scatter plot of absolute values of the net energy ($|E_{net}|$) and cumulative degree minutes supercooling (*CDMS*) of North Saskatchewan (black) and Peace River (grey) along with a line of best fit (red) and the 95 % prediction interval for the line of best fit (blue)

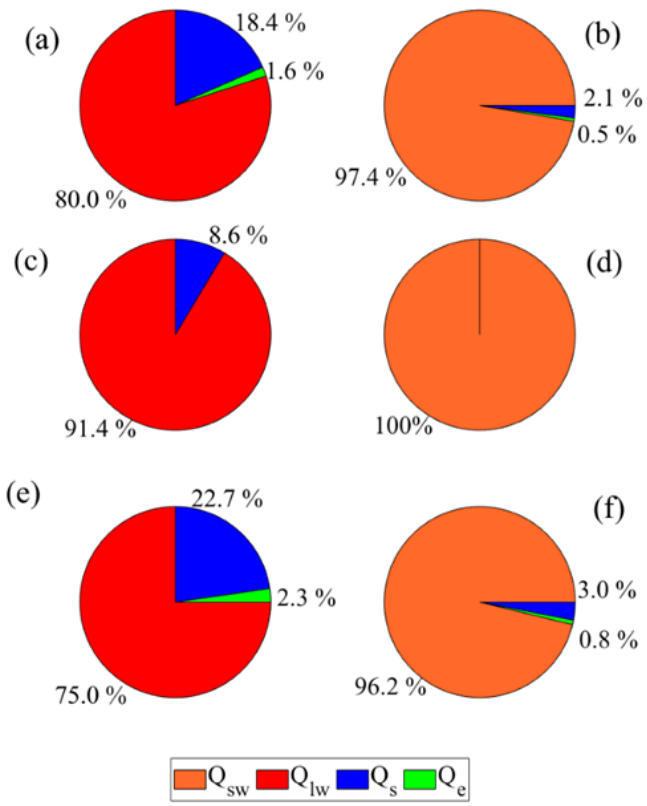

Figure 3-10: Dominance of heat flux components during negative (left) and positive (right) heat fluxes for (a)-(b) Both Rivers, (c)-(d) North Saskatchewan River and (e)-(f) Peace River

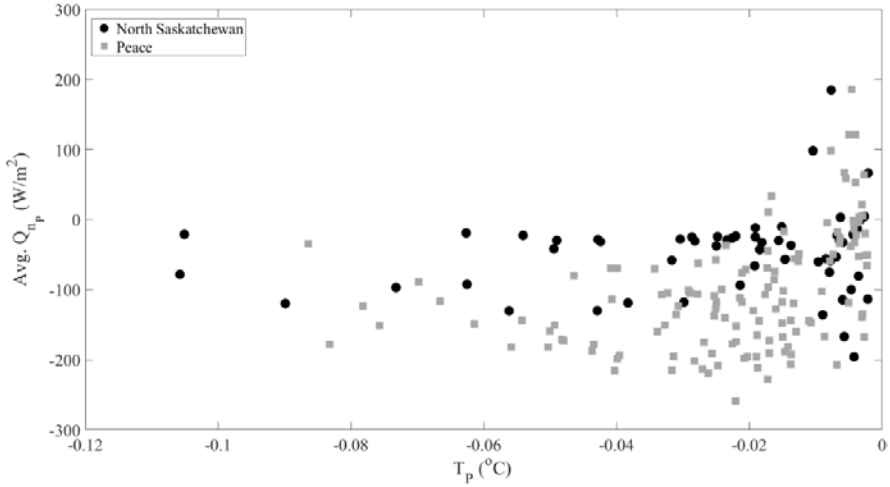

Figure 3-11: Scatter plot of average net principal supercooling heat flux (Q_{np}) and peak supercooling (T_p) of North Saskatchewan (black) and Peace River (grey)

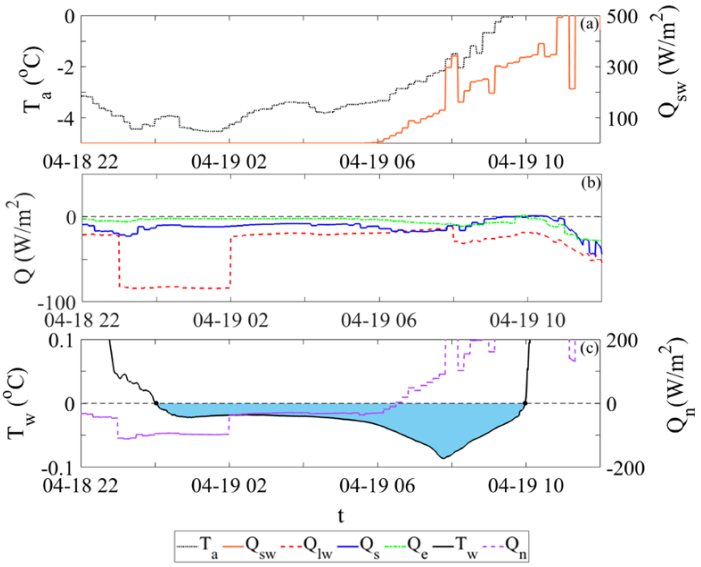

Figure 3-12: Event with positive $\overline{Q_n}$ observed on the Peace River at the left bank of Sta. 305 km. This event has the lowest T_P value of all events observed with a with positive $\overline{Q_n}$ at T_P = -0.086 °C

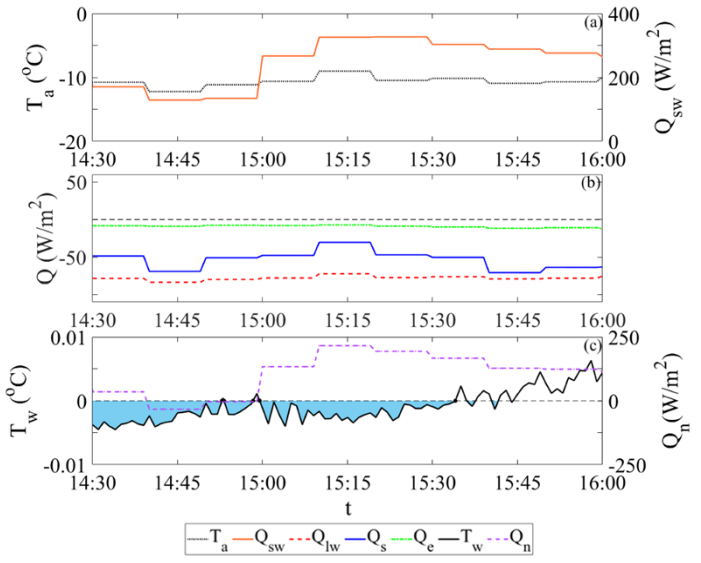

Figure 3-13: A time series from the left bank of Sta 293.5 km on the Peace River on March 1st, 2017 containing a typical duration supercooling event with a positive $\overline{Q_n}$ (3:00 – 3:35 PM). This figure also shows the end of a series of supercooling events that have a negative $\overline{Q_n}$ (events prior to 3 PM), as well as two instances of supercooling with duration < 10 minutes (thus not counted in the data set)

Chapter 4 Conclusions

4.1 Study of supercooling parameters

In Chapter 2, 696 supercooling events were observed on three Alberta rivers, and catalogued along with parameters of supercooling events established by previous literature. These parameters included the start and end time of events, duration, peak supercooling temperature, time to reach peak supercooling temperature, resulting principal supercooling cooling rate, and the cumulative degree minutes of supercooling (*CDMS)*.

Most of supercooling events (approximately 91 %) were observed during the freeze-up season. The events started any time of day, but primarily ended between 9 AM and 3 PM. Break-up events made up less that 9 % of total number event with almost all of them starting during the night and ending by the following morning. The events in this study were primarily less than 24 hours in duration (86.4 % of events) with a mean duration of 23.8 hours. The peak supercooling typically reached values between -0.01 and -0.02ºC. The principal supercooling cooling rates ranged from -1.03×10^{-6} to -1.18×10^{-2} ºC/minute with a mean value of -5.51×10^{-4} ºC/minute; this is roughly an order of magnitude smaller than reported laboratory studies, which is expected from previous observations by Osterkamp (1978). The *CDMS* of the supercooling events had median, mean, and maximum values of 2.31, 11.1 and 158 ◦C·minutes, respectively, and is within the range previously reported by Howley et al. (2019).

The most notable observations regarding supercooling events observed in this study are two events which measured peak supercooling temperatures of -0.105 and -0.106 ºC on the NSR on November 25th, 2016 at 7:28 PM and 4:25 AM, respectively. Events lasting for over 3 days were observed on all three rivers, exceeding the upper limit of supercooling event durations reported in previous literature. These long supercooling events tended to establish a constant residual temperature and maintain the temperature for multiple days at a time. Comparing the distribution of these long events across the rivers studied, 85 % of these long events were observed on the PR, the largest river used in the study, while 11 % and 4 % of these longer duration events were observed on the NSR and KR, respectively.

Analysis of the variation of supercooling behaviour between sequential sensors showed significant longitudinal variation on the KR and NSR that were likely explained by the impact of hydropeaking and influence of urbanization, respectively. While it is to be expected to observe this longitudinal variation on the KR, additional studies are required to confirm the behaviour observed on the NSR. Lateral differences between sensor pairs on the PR was also observed and is likely due to the impact of shading on the southern bank, which often experienced single longer duration events during multiple smaller duration events on the north bank.

4.2 Study of surface energy budget during supercooling

In Chapter 3, the 2016-2017 season of supercooling data was re-analyzed alongside the surface energy budget calculated from local meteorological data. Results from this analysis showed that the shortwave and longwave radiative heat fluxes were the most significant of the surface heat fluxes. Shortwave radiation was found to be the dominant positive heat flux for 97.4 % of all events, while longwave radiation dominated the negative heat flux for 80.0 % of supercooling events. Sensible heat flux was a strong secondary heat flux, and tended to be the dominant heat flux when air temperatures were significantly colder at an average air temperature of -14.8 ^{o}C compared to the overall average of -8.99 ^{o}C. The evaporative heat flux is only significant when the air temperature is above freezing, and was found to be the dominant positive and negative heat flux during a few events.

Linear correlation and multiple linear regression analysis between supercooling parameters and event averaged heat flux components found no significant results; the greatest magnitude linear correlation was -0.35 between peak supercooling and average principal shortwave radiation, while the best multiple linear regression equation (peak supercooling and principal supercooling averaged heat flux components) could only explain ~19 % of the variability observed in the data set. However, the visual plotting of the heat fluxes alongside the supercooling events and the start and end time distribution along side the average hourly heat flux show that some relationship between when supercooling events tend to end and the surface net heat flux. There appears to be a strong correlation between *CDMS* and E_{net}, (R^2 = 0.86), which may indicate a promising relationship between *CDMS* and ice production. However, since both of the parameters are related to the duration of an event, this is not conclusive evidence of the applicability of *CDMS* for ice production.

4.3 Future work

While in this study a larger data set of supercooling events was analyzed (compared to previously reported events), a comparison of supercooling behavior across a variety of rivers (unregulated rivers, more variable sizes, different climatic environment, etc.) would provide insights into how different properties of each river may impact supercooling events. This can also be extended to more detailed longitudinal and lateral studies that could give a better sense of the spatial evolution of supercooling events within a study reach.

The surface energy budget analysis showed that there is some relationship that can be derived between the start and end of supercooling events. Such relationships should be the focus of a study dedicated to direct measurements of the different heat components on the water surface. This research would reduce error in estimating heat fluxes and could improve our understanding of forecasting the start and end time of supercooling events. More detailed studies to estimate local ice production could then improve our understanding of how ice production evolves during supercooling events. This in return would improve the capacity of numerical models to estimate ice production locally (e.g., water intake structures) and ice transport (e.g., advancement of the ice front).

While *CDMS* is a useful shorthand for the scale of a supercooling event, it is still uncertain how effective the relation between the *CDMS* and ice production as proposed by Howley et al. (2019). In order to confirm this relation, more field studies are required to measure simultaneously the *CDMS* and frazil ice concentration. If this relation is confirmed, the *CDMS* would potentially be used as a surrogate for local ice production without the use of heat models.

References

Altberg, W.J., 1936. Twenty years of work in the domain of underwater ice formation, International Association of Scientific Hydrology Bulletin, 23, 373–407

Arden, R. S., Wigle, T. E., 1973. Dynamics of ice formation in the upper Niagara River. Int. Symp. Role of Snow and Ice in Hydrol., Banff, Alberta 1972, 2:1296-1313. Geneva: UNESCO-WMO-IAHS. 1483 pp.

Ashton, G. D. (1986). *River Lake Ice Engineering*. (G. D. Ashton, ed.), Water Resources Publications, Chelsea, Michigan.

Ashton, G. D. (2013). "Thermal Processes." *River Ice Formation*, S. Beltaos, ed., Committee on River Ice Processes and the Environment, CGU-HS, Edmonton, Alberta, 19–76.

Barnes, H. T., 1908. Formation of Ground- or Anchor-Ice, and other Natural Ice, Nature, 78, 102–104, doi: https://doi.org/10.1038/078102c0.

Blackburn, J. and She Y., 2019. A comprehensive public-domain river ice process model and its application to a complex natural river. Cold Regions Science and Technology, 163, 44-58. doi: https://doi.org/10.1016/j.coldregions.2019.04.010.

Bowen, I.S. (1926). "The Ratio of Heat Losses by Conduction and by Evaporation from Any Water Surface." *Phys. Rev*. 27(6), 779–787.

Boyd, S., Ghobrial, T., Loewen, M., Jasek, M., Evans J., 2022. A study of supercooling in rivers. *Cold Regions Science and Technology*, 194(2022), 1034-1055, doi: https://doi.org/10.1016/j.coldregions.2021.103455

Boyd, S., Ghobrial, T., Loewen, M., 2020. Observations of Supercooling in Rivers. IAHR Ice Symposium Trondheim, Norway. November 23 – 25 2020

Brown, G. W. (1969). "Predicting Temperatures of Small Streams." *Water Resources Research*, 5(1), 68–75.

Brown, L. E., Hannah, D. M., and Milner, A. M. (2006). "Thermal variability and stream flow permanency in an alpine river system." *River Research and Applications*, 22(4), 493–501. doi: 10.1002/rra.915

Buehler, H., 2013. Impact of a hydropeaking dam on the Kananaskis River : changes in geomorphology, riparian ecology, and physical habitat (T). Electronic Theses and Dissertations (ETDs) 2008+. University of British Columbia. doi: http://dx.doi.org/10.14288/1.0073548.

Carstens, T., 1966. Experiments with supercooling and ice formation in supercooled water: Geofysiske Publikasjoner, v. XXVI, no. 9, p. 1-18.

Clark, E., Webb, B. W., and Ladle, M., 1999. Microthermal gradients and ecological implications in Dorset rivers. *Hydrological Processes*, 13(3), 423–438.

Cozzetto, K., McKnight, D., Nylen, T., and Fountain, A., 2006. Experimental investigations into processes controlling stream and hyporheic temperatures, Fryxell Basin, Antarctica. *Advances in Water Resources*, 29(2), 130–153.

Daly, S., 1994. Report on frazil ice. US Army Corp of Engineers, Cold Regions Research & Engineering Laboratory, Hanover, New Hampshire, Report 94-23, 50 pages.

Evans, E. C., McGregor, G. R., and Petts, G. E., 1998. River energy budgets with special reference to river bed processes. *Hydrological Processes*, 12(April 1997), 575–595.

ECCC-Environment and Climate Change Canada (2020)a, Edmonton International Weather Station Cloud Data, Received Jul 31, 2020

ECCC-Environment and Climate Change Canada (2020)b, Peace River Weather Station Cloud Data, Received May 22, 2020

Ghobrial, T. R., Loewen, M. R., Hicks, F., 2012. Laboratory calibration of upward looking sonars for measuring suspended frazil ice concentration, *Cold Regions Science and Technology*, Volume 70, 2012, Pages 19-31, doi: https://doi.org/10.1016/j.coldregions.2011.08.010 .

Hannah, D. M., Malcolm, I. A., Soulsby, C., and Youngson, A. F., 2004. Heat exchanges and temperatures within a salmon spawning stream in the Cairngorms, Scotland: Seasonal and sub-seasonal dynamics. *River Research and Applications*, 20(6), 635–652.

Hicks, F., Cui W., and Ashton, G., 2008. Heat Transfer and Ice Decay. *River Ice Breakup*, S. Beltaos, ed., Water Resources Publications, LLC, 480 pp.

Howley, R., Ghobrial, T. R., She, Y., 2019. Thermal regime in the North Saskatchewan River in Edmonton. CGU HS Committee on River Ice Processes and the Environment 20th Workshop on the Hydraulics of Ice Covered Rivers Ottawa, Ontario, Canada, May 14-16, 2019, 2019.

Hubbard, K. G. (1994). “Spatial variability of daily weather variables in the high plains of the USA.” *Agricultural and Forest Meteorology*, 68(1–2), 29–41. doi: https://doi.org/10.1016/0168-1923(94)90067-1

Jasek, M., Ghobrial, T. R., Loewen, M. R., Hicks, F., 2011. Comparison of CRISSP modeled and SWIPS measured ice concentrations on the Peace River. CGU HS Committee on River Ice Processes and the Environment 16th Workshop on River Ice Winnipeg, Manitoba, September 18 – 22, 2011.

Jasek, M., Pryse-Phillips, A., 2015. Influence of the proposed Site C hydroelectric project on the ice regime of the Peace River. *Canadian Journal of Civil Engineering*. 42(9): 645-655. https://doi.org/10.1139/cjce-2014-0425

Jasek, M., Shen, HT., Pan, J., Paslawski, K., 2015. Anchor Ice Waves and their Impact on Winter Ice Cover Stability. CGU HS Committee on River Ice Processes and the Environment 18th Workshop on the Hydraulics of Ice Covered Rivers Quebec City, QC, Canada, August 18-20, 2015

Kalke H., McFarlane V., Schneck C., Loewen, M., 2017. The transport of sediments by released anchor ice, *Cold Regions Science and Technology*, 143(2017) 70-80. doi: http://dx.doi.org/10.1016/j.coldregions.2017.09.003

Kalke, H., McFarlane, V., Ghobrial, T. R., Loewen, M. R., 2019. Field Measurements of Supercooling in the North Saskatchewan River. CGU HS Committee on River Ice Processes and the Environment 20th Workshop on the Hydraulics of Ice Covered Rivers Ottawa, Ontario, Canada, May 14-16, 2019.

Kellerhals, R. C., C. R. Neil, D. I. Bray., 1972. Hydraulic and geomorphic characteristics of rivers in Alberta. Cooperative Research Program in Highway and River Engineering Technical Report. Alberta Environment, Edmonton, 1972.

König-Langlo, G., and Augstein, E., 1994. Parametrization of the downward longwave radiation at the Earth's surface in polar regions. *Meteorologische Zeitschrift*, 3: 343–347. doi:10013/epic.12338.d001.

MATLAB, 2021. *version 9.10.0.1710957 (R2021a)*, Natick, Massachusetts, The MathWorks Inc.

MathWorks, 2021. MathWorks documentation for Correlation Coefficients-corrcoef, URL: https://www.mathworks.com/help/matlab/ref/corrcoef.html, Retrieved: August 30, 2021

MathWorks, 2021. MathWorks documentation for Interpret Linear Regression Results, URL: https://www.mathworks.com/help/stats/understanding-linear-regression-outputs.html, Retrieved: August 30, 2021

Matousek, V., 1992. Frazil and skim ice formation in Rivers, IAHR Ice Symposium Banff, Alberta.

McFarlane, V., Clark, S. P. 2021. A detailed energy budget analysis of river supercooling and the importance of accurately quantifying net radiation to predict ice formation, *Hydrological Processes*, 2021; 35:e14056. doi: https://doi.org/10.1002/hyp.14056.

McFarlane, V., Loewen, M. R., Hicks, F., 2015. Measurements of the evolution of frazil ice particle size distributions. *Cold Regions Science and Technology*, 120, 45-55. doi: https://doi.org/10.1016/j.coldregions.2015.09.001

McFarlane, V., Loewen, M. R., Hicks, F., 2017. Measurements of the size distribution of frazil ice particles in three Alberta rivers. *Cold Regions Science and Technology*, 142, 100-117. doi: https://doi.org/10.1016/j.coldregions.2017.08.001.

McFarlane, V., Loewen M. R., Hicks, F., 2019. Field measurements of suspended frazil ice. Part II: Observations and analyses of frazil ice properties during the principal and residual supercooling phases, *Cold Regions Science and Technology*,165, 102-796, doi: https://doi.org/10.1016/j.coldregions.2019.102796.

Michel, B., 1967. Morphology of Frazil Ice. Proceedings of the International Conference on Low Temperature Science, Hokkaido University, Sapporo, Japan, pp. 119-128.

Michel, B., 1971. Winter regime of rivers and lakes. Cold Regions Science and Engineering Monograph III-B1a. U.S. Army Cold Regions Research and Engineering Laboratory, p. 131.

Nafziger, J., Hicks, F., Thomas, P., McFarlane, V., Banack, J., Cunjak, R.A., 2013. Measuring supercooling prevalence on small regulated and unregulated streams in New Brunswick and Newfoundland, Canada, CGU HS Committee on River Ice Processes and the Environment. Edmonton, AB., 2013.

Osterkamp, T., 1978. Frazil Ice Formation: A Review. *Journal of the Hydraulics Division,* 104, pp. 1239-1255. doi: https://doi.org/10.1061/JYCEAJ.0005060

Onset, 2021b. HOBO S-THB-M002 Sensor Datasheet, URL: https://www.onsetcomp.com/datasheet/S-THB-M002. Received: August 31, 2021

Onset, 2021c. HOBO S-LIB-M003 Sensor Datasheet, URL: https://www.onsetcomp.com/datasheet/S-LIB-M003 . Received: August 31, 2021

Prowse, T., 2001. River-Ice Ecology II: Biological Aspects, *Journal of Cold Regions Engineering., 15(1),* pp. 17-33. doi: https://doi.org/10.1061/(ASCE)0887-381X(2001)15:1(17)

QGIS Development Team, 2021. QGIS Geographic Information System.
Open Source Geospatial Foundation Project. URL. http://qgis.osgeo.org.

Quiñones, A., Cordoba B., Gutierrez, M., Hoogenboom, M., 2019. Radius of influence of air temperature from automated weather stations installed in complex terrain, *Theoretical and Applied Climatology*, 2019;137:1957–1973, doi: https://doi.org/10.1007/s00704-018-2717-9

Richard, M., Morse, B., 2008. Multiple frazil ice blockages at a water intake in the St. Lawrence River. *Cold Region Science and Technology* 53, 131–149. doi: https://doi.org/10.1016/j.coldregions.2007.10.003.

Richard, M., Morse, B., Daly, S., 2015. Modeling Frazil Ice Growth in the St. Lawrence River, *Canadian Journal of Civil Engineering*, 2015; 42: 592–608 doi: http://dx.doi.org/10.1139/cjce-2014-0082

Ruskin 2021. RBR Solo T Temperature Logger Datasheet, URL: https://rbr-global.com/products/compact-loggers/rbrsolo-t. Received: August 31, 2021

Ryan, P., Harleman, D.R., Stolzenbach, K.D. (1974). Surface Heat Loss from Cooling Ponds *Water Resour. Res.* 10(5), 930–938. doi: https://doi.org/10.1029/WR010i005p00930

Schneck, C. C., Ghobrial, T. R., and Loewen, M. R., 2019. Laboratory study of the properties of Frazil ice particles and flocs in water of different salinities, *The Cryosphere*, 13, 2751–2769, doi: https://doi.org/10.5194/tc-13-2751-2019.

Wazney, L., Clark, S., Malenchak, J., Knack, I. and Shen, H.T., 2019. Numerical simulation of river ice cover formation and consolidation at freeze-up. Cold Regions Science and Technology, 168, https://doi.org/10.1016/j.coldregions.2019.102884.

Webb BW, Zhang Y. (2004). Intra-annual variability in the non-advective heat energy budget of Devon streams and rivers. *Hydrol Process*. 2004;18:2117–46.

Webb, B. W., Hannah, D. M., Moore, R. D., Brown, L. E., and Nobilis, F., 2008. Recent advances in stream and river temperature research. *Hydrol Process*, 22(7), 902–918.

Ye, S., Doering, J., Shen, H., 2004. A laboratory study of frazil evolution in a counter-rotating flume. *Canadian Journal of Civil Engineering*. 31(6): 899-914, doi: https://doi.org/10.1139/l04-056

Appendix A – Summary of linear correlation & multiple linear regression analysis

List of Summary Tables

A-1 Summary of linear correlation analysis method

Linear correlation was found between all supercooling parameters and all heat flux components and net heat flux (Table A.1) as well as between principal supercooling parameters and the average principal supercooling heat fluxes (Table A.2). In both cases, the correlation was calculated using the MATLAB function *corrcoef()* documented by MathWorks (2021a).

The *fitlm* function was used to generate a multiple linear regression equation between supercooling parameters and the event average of each heat flux component (Table A.3 to A.18). For the parameters defined in principal supercooling (T_P, D_P, and CR_P), an additional model was developed between the parameter and the principal supercooling averaged heat flux components. The function uses a *t*-test for each estimated coefficient in the linear regression model compared to ignoring the input parameter, as well as an *F*-test for the overall model against assuming a constant value (MathWorks 2021b). For both *p*-values, if the *p*-value is less than 0.05, the test is significant at the 5% level (MathWorks 2021b). The final point of assessment is that the R^2 value can be interpreted as an indicator of what fraction of the variability in the response data is explained by the linear regression model (MathWorks 2021b).

A-2 Summary Tables

Table A.1: Summary of R^2 values of supercooling event parameters and average heat flux

Average Heat Flux component (W/m^2)	T_P (°C)	D (hours)	D_P (hours)	CDMS (°C·minutes)	CR_P (°C/minute)
$\overline{Q_{sw}}$	-0.15	-0.13	-0.11	-0.15	0.10
$\overline{Q_{lw}}$	-0.09	-0.23	0.04	-0.23	-0.05
$\overline{Q_s}$	0.15	-0.20	0.08	-0.17	-0.14
$\overline{Q_e}$	0.10	0.03	0.04	0.03	-0.08
$\overline{Q_n}$	-0.08	-0.31	-0.04	-0.31	-0.01

Table A.2: Summary of R^2 values of principal supercooling parameters and average principal heat flux components

Average Principal Heat Flux Components (W/m^2)	T_P (°C)	D_P (hours)	CR_P (°C/minute)
$\overline{Q_{sw_p}}$	-0.35	-0.09	0.11
$\overline{Q_{lw_p}}$	-0.14	0.05	-0.05
$\overline{Q_{s_p}}$	0.15	0.09	-0.17
$\overline{Q_{e_p}}$	0.12	0.03	-0.09
$\overline{Q_{n_p}}$	0.28	-0.01	-0.01

Table A.3: Summary of multiple linear regression model between peak supercooling (T_P) and average heat flux components

Parameter	Intercept	$\overline{Q_{sw}}$ (W/m²)	$\overline{Q_{lw}}$ (W/m²)	$\overline{Q_s}$ (W/m²)	$\overline{Q_e}$ (W/m²)
Coefficients	$1.83\text{x}10^{-2}$	$4.01\text{x}10^{-5}$	$2.35\text{x}10^{-4}$	$-1.76\text{x}10^{-4}$	$-7.28\text{x}10^{-5}$
Standard Error	$3.74\text{x}10^{-3}$	$2.30\text{x}10^{-5}$	$6.51\text{x}10^{-5}$	$5.11\text{x}10^{-5}$	$2.38\text{x}10^{-4}$
t-statistic	-4.88	1.75	3.61	-3.44	-0.31
t-statistic p-value	$2.31\text{x}10^{-6}$	$8.26\text{x}10^{-2}$	$3.88\text{x}10^{-4}$	$7.23\text{x}10^{-4}$	$7.68\text{x}10^{-1}$

Table A.4: Peak supercooling (T_P) and average heat flux components multiple linear regression model statistics

Number of Observations:	190	**Degrees of Freedom:**	185
R^2:	0.103	**Adjusted R^2:**	0.0838
F-statistic vs. constant	5.29	**F-statistic p-value:**	$4.65\text{x}10^{-4}$
Root Mean Squared Error	$1.96\text{x}10^{-2}$		

Table A.5: Summary of multiple linear regression model between peak supercooling (T_P) and average principal supercooling heat flux components

Parameter	Intercept	$\overline{Q_{sw_p}}$ (W/m²)	$\overline{Q_{lw_p}}$ (W/m²)	$\overline{Q_{s_p}}$ (W/m²)	$\overline{Q_{e_p}}$ (W/m²)
Coefficients	$-1.91\text{x}10^{-2}$	$9.16\text{x}10^{-5}$	$1.93\text{x}10^{-4}$	$-1.33\text{x}10^{-4}$	$3.90\text{x}10^{-6}$
Standard Error	$3.26\text{x}10^{-3}$	$2.17\text{x}10^{-5}$	$5.15\text{x}10^{-5}$	$5.07\text{x}10^{-5}$	$2.45\text{x}10^{-4}$
t-statistic	-5.83	4.23	3.75	-2.63	$1.59\text{x}10^{-2}$
p-value	$2.38\text{x}10^{-8}$	$3.72\text{x}10^{-5}$	$2.39\text{x}10^{-4}$	$9.32\text{x}10^{-3}$	$9.87\text{x}10^{-1}$

Table A.6: Peak supercooling (T_P) and average principal supercooling heat flux components multiple linear regression model statistics

Number of Observations:	190	**Degrees of Freedom:**	185
R^2:	0.188	**Adjusted R^2:**	0.171
F -statistic vs. constant	10.7	**F -statistic p-value:**	7.7×10^{-8}
Root Mean Squared Error	1.87×10^{-2}		

Table A.7: Summary of multiple linear regression model between duration (D) and average heat flux components

Parameter	**Intercept**	$\overline{\mathbf{Q_{sw}}}$ **(W/m²)**	$\overline{\mathbf{Q_{lw}}}$ **(W/m²)**	$\overline{\mathbf{Q_s}}$ **(W/m²)**	$\overline{\mathbf{Q_e}}$ **(W/m²)**
Coefficients	9.00	-2.02×10^{-1}	-3.83×10^{-1}	-2.19×10^{-1}	5.75×10^{-2}
Standard Error	10.7	6.59×10^{-2}	1.87×10^{-1}	1.46×10^{-1}	6.81×10^{-2}
t-statistic	8.39×10^{-1}	-3.07	-2.05	-1.49	8.44×10^{-2}
t-statistic p-value	4.02×10^{-1}	2.46×10^{-3}	4.16×10^{-2}	1.37×10^{-1}	9.33×10^{-1}

Table A.8: Duration (D) and average heat flux components multiple linear regression model statistics

Number of Observations:	190	**Degrees of Freedom:**	185
R^2:	0.105	**Adjusted R^2:**	0.0861
F-statistic vs. constant	5.45	**F-statistic p-value:**	3.6×10^{-4}
Root Mean Squared Error	56.2		

Table A.9: Principal supercooling duration (D_P) and average heat flux components multiple linear regression model statistics

Parameter	Intercept	$\overline{Q_{sw}}$ (W/m²)	$\overline{Q_{lw}}$ (W/m²)	$\overline{Q_s}$ (W/m²)	$\overline{Q_e}$ (W/m²)
Coefficients	3.97	-9.96×10^{-3}	-5.13×10^{-3}	1.12×10^{-2}	1.43×10^{-2}
Standard Error	1.35	8.27×10^{-3}	2.34×10^{-2}	1.84×10^{-2}	8.56×10^{-2}
t-statistic	2.95	-1.20	-2.19×10^{-1}	6.11×10^{-1}	1.67×10^{-1}
t-statistic p-value	3.63×10^{-3}	2.30×10^{-1}	8.27×10^{-1}	5.42×10^{-1}	8.67×10^{-1}

Table A.10: Principal supercooling duration (D_P) and average hat flux components multiple linear regression model statistics

Number of Observations:	190	**Degrees of Freedom:**	185
R^2:	0.015	**Adjusted R^2:**	-6.38×10^{-3}
F -statistic vs. constant	0.701	**F -statistic p-value:**	0.593
Root Mean Squared Error:	7.06		

Table A.11: Summary of multiple linear regression model between principal supercooling duration (D_P) and average principal heat flux components

Parameter	Intercept	$\overline{Q_{sw_p}}$ (W/m²)	$\overline{Q_{lw_p}}$ (W/m²)	$\overline{Q_{s_p}}$ (W/m²)	$\overline{Q_{e_p}}$ (W/m²)
Coefficients	4.06	-6.98×10^{-3}	7.10×10^{-4}	1.23×10^{-2}	6.44×10^{-3}
Standard Error	1.24	8.21×10^{-3}	1.95×10^{-2}	1.92×10^{-2}	9.28×10^{-2}
t-statistic	3.29	-8.51×10^{-1}	3.64×10^{-2}	6.45×10^{-1}	6.95×10^{-2}
p-value	1.23×10^{-3}	3.96×10^{-1}	9.71×10^{-1}	5.20×10^{-1}	9.45×10^{-1}

Table A.12: Principal supercooling duration (D_P) and principal supercooling average heat flux components multiple linear regression model statistics

Number of Observations:	190	**Degrees of Freedom:**	185
R^2:	0.013	**Adjusted R^2:**	-8.79×10^{-3}
F -statistic vs. constant	0.588	**F -statistic p-value:**	0.677
Root Mean Squared Error	7.07		

Table A.13: Summary of multiple linear regression model between principal supercooling average cooling rate (CR_P) and average heat flux components

Parameter	**Intercept**	$\overline{Q_{sw}}$ **(W/m²)**	$\overline{Q_{lw}}$ **(W/m²)**	$\overline{Q_s}$ **(W/m²)**	$\overline{Q_e}$ **(W/m²)**
Coefficients	-3.13×10^{-4}	-8.01×10^{-7}	-2.31×10^{-6}	4.08×10^{-6}	6.65×10^{-6}
Standard Error	1.84×10^{-4}	1.13×10^{-6}	3.19×10^{-6}	2.50×10^{-6}	1.17×10^{-6}
t-statistic	-1.68	-7.15×10^{-1}	-6.87×10^{-1}	1.58	5.84×10^{-1}
t-statistic p-value	9.03×10^{-2}	4.78×10^{-1}	4.71×10^{-1}	1.05×10^{-1}	5.69×10^{-1}

Table A.14: Principal supercooling average cooling rate (CR_P) and average heat flux components multiple linear regression model statistics

Number of Observations:	190	**Degrees of Freedom:**	185
R^2:	0.028	**Adjusted R^2:**	7.4×10^{-3}
F -statistic vs. constant	1.35	**F -statistic p-value:**	2.52×10^{-1}
Root Mean Squared Error	9.62×10^{-4}		

Table A.15: Summary of multiple linear regression model between principal supercooling average cooling rate (CR_P) and average principal supercooling heat flux components

Parameter	Intercept	$\overline{Q_{sw_p}}$ (W/m²)	$\overline{Q_{lw_p}}$ (W/m²)	$\overline{Q_{s_p}}$ (W/m²)	$\overline{Q_{e_p}}$ (W/m²)
Coefficients	-2.88×10^{-4}	-6.40×10^{-7}	-2.22×10^{-6}	5.18×10^{-6}	6.77×10^{-6}
Standard Error	1.67×10^{-4}	1.11×10^{-6}	2.64×10^{-6}	2.60×10^{-6}	1.26×10^{-5}
t-statistic	-1.72	-5.76×10^{-1}	-8.43×10^{-1}	2.00	5.40×10^{-1}
p-value	8.63×10^{-2}	5.65×10^{-1}	4.01×10^{-1}	4.74×10^{-2}	5.90×10^{-1}

Table A.16: Principal supercooling average cooling rate (CR_P) and principal supercooling average heat flux components multiple linear regression model statistics

Number of Observations:	190	**Degrees of Freedom:**	185
R^2:	0.041	**Adjusted R^2:**	1.99×10^{-2}
F -statistic vs. constant	1.96	**F -statistic p-value:**	1.03×10^{-1}
Root Mean Squared Error	9.56×10^{-4}		

Table A.17: Summary of multiple linear regression model between cumulative degree minutes of supercooling (*CDMS*) and average heat flux components

Parameter	Intercept	$\overline{Q_{sw}}$ (W/m²)	$\overline{Q_{lw}}$ (W/m²)	$\overline{Q_s}$ (W/m²)	$\overline{Q_e}$ (W/m²)
Coefficients	6.67	-9.79×10^{-2}	-1.95×10^{-1}	-7.42×10^{-2}	-1.41×10^{-3}
Standard Error	4.94	3.03×10^{-2}	8.59×10^{-2}	6.74×10^{-2}	3.14×10^{-1}
t-statistic	1.35	-3.23	-2.27	-1.10	-4.49×10^{-3}
t-statistic p-value	1.79×10^{-1}	1.48×10^{-3}	2.42×10^{-2}	2.72×10^{-1}	9.96×10^{-1}

Table A.18: Cumulative degree minutes of supercooling (*CDMS*) Supercooling and average heat flux components multiple linear regression model statistics

Number of Observations:	190	**Degrees of Freedom:**	185
R^2:	0.104	**Adjusted R^2:**	8.46×10^{-2}
F -statistic vs. constant	5.37	**F -statistic p-value:**	4.31×10^{-4}
Root Mean Squared Error	25.9		

Appendix B – Code documentation

B-1: Overview

This appendix provides a brief overview of the core functions/programs developed over the course of the research and analysis of this thesis. In all cases, the code should be easily adaptable for future research.

B-2: Supercooling analysis programs

B-2.1 Summary of functions

The programs used for analysis in Chapter 2 are were developed to automate the processing of data from the raw data files, and should be easily adapted for continued use. The documentation for a specific function for things to keep in mind when adding additional elements to the analysis.

These functions were developed with the specific work environment of MATLAB in mind. In the code presented, three specific sub-folders are referenced: Database, Generated Results, and Raw Data Files. These folders hold: the database structure, any generated results from analysis, and the raw data files, respectively.

The programs can be divided into two periods of application: entering and processing newly collected water temperature data files, and data management tools. Table B-1 gives a brief summary of the programs used to input and process new raw data files, while Table B-2 presents a summary of the functions developed for analysis of the data or information management.

Table B. 1: Summary of programs required to process raw data files, presented in the order of operation. Page indicates the starting page of the copied MATLAB code.

Period of Application	**Program**	**Summary of Function**	**Page**
Entering/Processing New Data Files	*CreateInputFile()*	Create an Excel file for user to enter in the required information for newly collected data files that are not recorded on the file list	99
	NewRawFiles()	Add the information recorded on the Excel file generated by *CreateInputFile()* to a .mat table that is less prone to accidental editing. Tracks which files have been processed, and the required information for processing the data file. Can also be used to track changes to the data set done using *AddAlterData()*	100
	ProcessNewFiles()	Processes the raw data files that are noted on the file list to have not yet been processed	105
Entering/Processing New Data Files or Information Management & Analysis	*AddAlterData()*	Enables the user to edit the raw file list to correct any errors from initial entry, make changes to initial parameters for different conditions, as well as add notes to the data base regarding specific deployments/data files	136

Table B. 2: Summary of Information management and analysis programs. Page indicates the starting page of the copied MATLAB code.

Period of Application	**Program**	**Summary of Function**	**Page**
Information Management & Analysis	*GetFileList()*	Get a copy of the raw data file list	155
	GetDatabase()	Get a copy of the database structure	159
	GetObSummary()	Generate a copy of the observation summaries of deployments. Can print a copy to Excel	159
	GetScreenedPeriods()	Get a copy of the summary and time series of the events that have been manually screened. The summary include which deployment they were a part of and the reason for them being screened	165
	GetTable()	Get a copy of the summary of the recorded supercooling events. Can be filtered using inputs to get specific events (event IDs), or events that meet a specified criterion.	166
	GetTimeSeries()	Get a copy of the time series of recorded supercooling events and deployments. Can be filtered using inputs to get specific time series (deployment/event IDs), or events that meet a specified criterion.	177
	ClearDatabase()	Clear the database. Gives the user the option to keep the records of the manually screened events (avoid re-screening events).	181
	SCPlot()	Plotting tool that enables consistent formatting and faster plotting of specific time series. Integrated with *GetTimeSeries ()* such that multiple figures can be generated with a single command (ex. all deployments period)	183
	StatTable()	Outputs a statical summary of the inputted table. Can be filtered by submitting specific table headings. Can be used with any table with quantitative data.	207

B-3.2 Program code

```
function [] = CreateInputFile()
%CreateInput: Creates an input file for NewRawFiles function. This enables
%cosistant formatting of inputs so there are minimal errors in formatting.
%Note that this method requires that the user double check spelling of
%rivers and sites, as any mis-spelled errors would be a understood as a new
%site/season. River, Site, and Deployment ID typos will not cause errors,
%but considering their usefulness in data filtering, consistent spelling is
%ideal. the AddAlterData can be used to correct any mistakes detected
%afterwards.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Author: Sean R. Boyd January 19th, 2022
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Get unlogged files
f = GetFileList(4);
%Generate Table
T = table('Size',[length(f) 26],'VariableType',...

{'double','string','double','double','double','double','string','string','string','string',...
    'double','double','double','double','double','double','double','double','double',...
    'double','double','double','double','double','datetime','datetime'},...

'VariableNames',{'File_Number','File_Name','Sensor_Accuracy','Sensor_Offset','Negative_Threshold'
,...

'Minimum_Event_Duration_Minutes','River','River_ID','Site','Site_ID','Deployment_Start_Year',...

'Deployment_End_Year','Month_Fall_Start','Day_Fall_Start','Hour_Fall_Start','Minute_Fall_Start',.
..
    'Month_Winter_Start','Day_Winter_Start','Hour_Winter_Start','Minute_Winter_Start',...

'Month_Spring_Start','Day_Spring_Start','Hour_Spring_Start','Minute_Spring_Start','Freeze_Up_End_
Date','Break_Up_Start_Date'});
%Add unlogged RSK File Names and default values
T.File_Name =f';T.File_Number = (1:height(T))';
T.Sensor_Accuracy(:)=0.002;T.Negative_Threshold(:) = -0.2; %Default negative threshold
T.Minimum_Event_Duration_Minutes(:)=10; %Default minimum duration
T.Month_Fall_Start(:)=9;T.Day_Fall_Start(:) = 1;T.Hour_Fall_Start(:) = 0;T.Minute_Fall_Start(:) =
0; %Default September 1st of the start year of the season
T.Month_Winter_Start(:)=12;T.Day_Winter_Start(:) = 1;T.Hour_Winter_Start(:) =
0;T.Minute_Winter_Start(:) = 0; %Default December 1st of the start year of the season
T.Month_Spring_Start(:)=3;T.Day_Spring_Start(:) = 1;T.Hour_Spring_Start(:) =
0;T.Minute_Spring_Start(:) = 0; %Default March 1st of the end year of the season
T.Freeze_Up_End_Date(:)="NaT";T.Break_Up_Start_Date(:)="NaT";
%Create xlsx file in directory (easy to input back to NewRSKFiles)
tabName =sprintf('Raw File Input %s.xlsx',datestr(now,'dd-mm-yyyy-HHMMSS'));
writetable(T,tabName)
```

```
%Print statement
fprintf('An Excel file has been created in the main directory.\n')
fprintf('Once you have completed the file, inputting the filename\n')
fprintf('as the input for NewRawFiles will import the file to MATLAB\n')
fprintf('and log the raw data files for processing.\n')
fprintf('\nNote that any spelling error in River and Site Names will\n')
fprintf('be interpreted as unique rivers and sites by the program.\n\n')
fprintf('Suggested date formmatting for importing from Excel to MATLAB.\n')
fprintf("is yyyy-mm-dd. In addition, set the column value to 'Date'\n\n")
fprintf('For ease of file conversion, any deployment notes will need\n')
fprintf('to be added after the files are logged and/or processed with\n')
fprintf('AddAlterData function\n')
end
```

Published with MATLAB® R2021a

```
function [err]=NewRawFiles(InputFilename)
%NewRawFiles: Generates/updates CurrentRawFileList for raw data files in
%Raw Data Files folder. The program requests the required information to
%complete the table. If there is a conflict at any stage, the program
%outputs a summary of the incomplete table so that the user can use that
%to determine the error and termininates

%Once the program has updated CurrentRawFileList, the program then
%asks if the user wants to process the new data files, processing and
%updating the database if the user wishes to do so.
%
%INPUT:
%   InputFilename: An excel file that contains the requested information.
%   If this is not submitted, the program will prompt the user to generate
%   one using CreateInputFile function
%OUTPUT
%   err: If a conflict causes the program to terminate early (or the user
%   terminates the program early), the program outputs the incomplete list
%   so that the user can determine where the error occured.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Author: Sean R. Boyd January 19th, 2022
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

narginchk(0,1);
clc;err=[]; %Assume that the program runs successfully (err will be cleared in the case that the
program runs without any issues)

%2) Set up directory lists
home = pwd; lists = sprintf('%s/Raw Data Files',home);
%3) If RawFileList.mat exists, load it.  Otherwise create it
cd(lists)
if exist('CurrentRawFileList.mat','file') == 2
    load('CurrentRawFileList.mat','rawfileList')
    rawfileList.File_Number(:) = (1:height(rawfileList));
    oldList = rawfileList;
```

```
else
    rawfileList = table('Size',[1 20],'VariableType',...

{'double','string','double','double','double','double','string','string','string','string',...

'string','string','datetime','datetime','datetime','datetime','datetime','string','string','strin
g'},'VariableNames',...

{'File_Number','File_Name','Sensor_Accuracy','Sensor_Offset','Negative_Threshold','Minimum_Event_
Duration','River','River_ID',...

'Site','Site_ID','Deployment_Period','Deployment_ID','Fall_Start','Winter_Start','Spring_Start','
Freeze_Up_End_Date',...
        'Break_Up_Start_Date','Deployment_Notes','Review_Notes','Processed'});
    oldList=[];
end

if nargin==0 %No input file to use to log files
    cd(home)
    error("Please generate an input file using 'CreateInputFile' in order to log any new raw data
files.")
else %There is an input file
    %Check that InputFile is an excel file
    cd(home)
    if exist(InputFilename,'file')
        InptFile = readtable(InputFilename);
    else
        error('%s cannot be found in the current directory.',InputFilename)
    end
    %Create a table that will store the new data prior to adding it to the
    %larger table
    newRawFiles = table('Size',[height(InptFile),20],'VariableType',...

{'double','string','double','double','double','double','string','string','string','string',...

'string','string','datetime','datetime','datetime','datetime','datetime','string','string','strin
g'},'VariableNames',...

{'File_Number','File_Name','Sensor_Accuracy','Sensor_Offset','Negative_Threshold','Minimum_Event_
Duration_Minutes','River','River_ID',...

'Site','Site_ID','Deployment_Period','Deployment_ID','Fall_Start','Winter_Start','Spring_Start','
Freeze_Up_End_Date',...
        'Break_Up_Start_Date','Deployment_Notes','Review_Notes','Processed'});
```

```
%Add the columns that are identical from the file list to the
    %newRawFiles table
    newRawFiles.File_Number = InptFile.File_Number; newRawFiles.File_Name =
string(InptFile.File_Name);

newRawFiles.Sensor_Accuracy = InptFile.Sensor_Accuracy;newRawFiles.Sensor_Offset =
InptFile.Sensor_Offset;
    newRawFiles.Negative_Threshold =
InptFile.Negative_Threshold;newRawFiles.Minimum_Event_Duration_Minutes =
InptFile.Minimum_Event_Duration_Minutes;
    newRawFiles.River = string(InptFile.River); newRawFiles.River_ID = string(InptFile.River_ID);
    newRawFiles.Site = string(InptFile.Site); newRawFiles.Site_ID = string(InptFile.Site_ID);
    %Add in placeholders for the notes and processing status
    newRawFiles{:,[18,19]} = "N/A";newRawFiles{:,20} = "No";
    %For the height of the table, generate the deployment period,
    %check that the date-times convert
    %properly, and generate the Deployment ID
    for k = 1:height(newRawFiles)
        %Check deployment period
        y1 = InptFile.Deployment_Start_Year(k);y2 = InptFile.Deployment_End_Year(k);
        if ~isnumeric(y1) || ~isnumeric(y2)
            error('Deployment Start and End Years must be entered as a numeric value.')
        end
        if (y2-y1) == 1
            deployPeriod=string(sprintf('%s-%s',string(y1),string(y2)));
            %Add the seasons to the table
            %Fall
            [newRawFiles] = addSeason(newRawFiles,InptFile,"Fall_Start",k);
            %Winter
            [newRawFiles] = addSeason(newRawFiles,InptFile,"Winter_Start",k);
            %Spring
            [newRawFiles] = addSeason(newRawFiles,InptFile,"Spring_Start",k);
        else
            error('Issue with Deployment Period. Year 2 must only be 1 year bigger than Year 1.')
        end
        newRawFiles.Deployment_Period(k) = deployPeriod;
        newRawFiles.Deployment_ID(k) =
sprintf('%s%s%s',newRawFiles.Site_ID(k),deployPeriod{:}(3:4),deployPeriod{:}(8:9));
        %Convert Freeze-Up and Break-Up dates into datetime
        try
            newRawFiles.Freeze_Up_End_Date(k) =
datetime(datestr(string(InptFile.Freeze_Up_End_Date(k))));
        catch
            if strcmpi(string(InptFile.Freeze_Up_End_Date(k)),"NaT")
                newRawFiles.Freeze_Up_End_Date(k) = NaT;
            else
                error('NewRawFiles cannot convert %s to a date-time or NaT value for
%s.',string(InptFile.Freeze_Up_End_Date(k)),newRawFiles.File_Name(k))
            end
        end
```

```
        try
            warning('off','MATLAB:datenum:EmptyDate') %Supress warning that this code is checking
for
            newRawFiles.Break_Up_Start_Date(k) =
datetime(datestr(string(InptFile.Break_Up_Start_Date(k))));
        catch
            if strcmpi(string(InptFile.Break_Up_Start_Date(k)),"NaT")
                newRawFiles.Break_Up_Start_Date(k) = NaT;
            else
                error('NewRawFiles cannot convert %s to a date-time or NaT value for
%s.',string(InptFile.Break_Up_Start_Date(k)),newRawFiles.File_Name(k))
            end
        end
        %Check that years of freeze-up and break up are equal to either y1
        %or y2
        if~isnat(newRawFiles.Freeze_Up_End_Date(k))
            if year(newRawFiles.Freeze_Up_End_Date(k)) ~= y1 &&
year(newRawFiles.Freeze_Up_End_Date(k)) ~= y2
                error('Recorded Freeze-up End Date  for %s does not happen in Deployment Period
%s',newRawFiles.File_Name(k),deploymentPeriod)
            end
        end
        if~isnat(newRawFiles.Break_Up_Start_Date(k))
            if year(newRawFiles.Break_Up_Start_Date(k)) ~= y1 &&
year(newRawFiles.Break_Up_Start_Date(k)) ~= y2
                error('Recorded Break-up Start Date  for %s does not happen in Deployment Period
%s',newRawFiles.File_Name(k),deploymentPeriod)
            end
        end
    end
    warning('on','MATLAB:datenum:EmptyDate') %Turn supressed warning back on
    %Determine if there are any duplicate Deployment IDs
    if isempty(oldList)
        if ~isequal(sort(newRawFiles.Deployment_ID),unique(newRawFiles.Deployment_ID))
            error('Duplicate Deployment IDs have been found in the submitted new files. Check
River/Site Names and Deployment Start/End Years.')
        end
    else
        allDID = [oldList.Deployment_ID;newRawFiles.Deployment_ID];
        if ~isequal(sort(allDID),unique(allDID))
            error('Duplicate Deployment IDs have been found in the submitted new files with
either themselves or the older files. Check River/Site Names and Deployment Start/End Years.')
        end
    end
    %Determine any potentially missing values in the table
    chk = ismissing(newRawFiles);chkIndex = find(chk == 1);
    for k = 1:length(chkIndex)
        [row,col] = ind2sub(size(newRawFiles),chkIndex(k));
        if col~=16 && col~=17 %These are the date-time columns, which may have NaT values
            error('There appears to be be missing values in %s',InputFilename)
```

```
elseif col == 16 || col == 17
            if  ~isnat(newRawFiles{row,col})
                error('There appears to be be missing (not NaT) value for a date-time in
%s',InputFilename)
            end
        end
    end
    %Print out a table in the command window for the verify the input table
    clc;disp(newRawFiles);chk=1;
    while chk
        fprintf('\n Are there any issues with the current input table for the new raw data
files?: (Y/N)')
        inpt=input(' ','s');
        if strcmpi(inpt,'Y')
            fprintf('\n\n Review %s for errors. Re-run this program after errors have been
corrected',InputFilename);chk=0;
        elseif strcmpi(inpt,'n')
            %If nothing appears to be missing, add the new files to rawfilelist
            if ~isempty(oldList)
                startrow = height(rawfileList)+1;endrow = height(newRawFiles)+startrow-1;
                rawfileList(startrow:endrow,:)=newRawFiles;
            else
                rawfileList =  newRawFiles;
            end
            clc;disp('Logging New raw data files Completed');cd(home);clearvars
err;pause(1.5);clc;close()
            %Check if the user wishes to delete the input file
            inpt=input(sprintf('With the new raw data files logged, do you wish to delete %s?
[Y/N]: ',InputFilename),'s');
            if strcmpi(inpt,'Y')
                delete(InputFilename)
                fprintf('%s has been deleted.\n',InputFilename)
            elseif strcmpi(inpt,'N')
                fprintf('%s is not deleted.\n',InputFilename)
            else
                fprintf('Invalid entry. %s is not deleted.\n',InputFilename)
            end
            %Save the new file
            cd(lists)
            rawfileList = sortrows(rawfileList,[6,8,10]); %Orders by River, then by site, then by
deployment period
            rawfileList.File_Number = (1:height(rawfileList))';
            save('CurrentRawFileList.mat','rawfileList')
            %Determine if the user wishes to Process the new files
            chk1=1;
            while chk1
                inpt = input('Do you wish to process the new raw data files? (Y/N)?: ','s');
                if strcmpi(inpt,'y') %Process new files
                    chk1=0;clc;cd(home);ProcessNewFiles;
                elseif strcmpi(inpt,'n') %Do not process new files
                    chk1=0;clc
                else %Invalid entry
                    clc;fprintf('\nInvalid entry. Please type (Y)es/(N)o\n')
```

```
                end
            end
        else
            fprintf('\nInvalid entry\n')
        end
    end
end
end
%SUBFUNCTION

function [newRawFiles] = addSeason(newRawFiles,InptFile,colName,row)
%Determine Reference columns in InptFile for specified season season
season = colName{1}(1:(find(colName{1} == '_',1))-1); %First part of column name is the season
%Extract the month and day from newFiles
m = InptFile.(strcat("Month_",season,"_Start"))(row);d =
InptFile.(strcat("Day_",season,"_Start"))(row);
h = InptFile.(strcat("Hour_",season,"_Start"))(row);M =
InptFile.(strcat("Minute_",season,"_Start"))(row);
%Determine the correct year
if m>6 %If the month happens after June this is the start of a deployment period
    y = InptFile.Deployment_Start_Year(row);
else %end of a deployment period
    y = InptFile.Deployment_End_Year(row);
end
%Generate datetime and add to newRawFiles
newRawFiles.(colName)(row) = datetime(datestr(sprintf('%d-%d-%d %d:%d',y,m,d,h,M)));
end
```

Published with MATLAB® R2021a

```
function ProcessNewFiles
%ProcessNewFiles: Loads CurrentRawFileList, processes the files listed
%within it, and compiles the supercooling events ino a single database.
%After processing these events, it records which of the files have been
%processed. The program flags certain events to be manually screened by
%the user. See NOTES ON MANUAL SCREENING OF EVENTS for more detail.
%This program prints a summary of a data file to the command window
%during processing. The time the text stays on the screen is controled by
%the variable pausedur set by the user at Line 95 of this function.
%Larger values means longer processing time.
%
%The Database is a structure containing three fields:
%   i) Observation Summaries - This is a sub structure that stores the
%   summary of the number of events identified, catalogued, and discarded
%   in each Deployment_Period, as well as other key parameters of the
%   Deployment_Period such as those submitted when logging the raw file
%   (NewRawFiles) as well other parameters of the overall time series such
%   as average time step. This structure can be extracted an more easily
%   analysed useing GetObSummary.
%
```

```
%   ii) Event Table - This is a summary table of all events observed in
%   the processing of the raw files. This can be extracted using
%   GetTable function.
%
%   iii) Time Series - This sub-structure contains both the full Deployment
%   Period time-series (both the filtered time series between end of
%   freeze-up and start of break-up and raw data) as well as the time
%   series for all catalogued supercooling events.
%   The Deployment Period or event time-series can be extracted using
%   GetTimeSeries.
%
%DEFINITIONS USED FOR CATALOGING AN EVENT
%   Definition of Supercooling Event: A supercooling event is defined as a
%   continuous measurement of water temperature below 0C.  The event
%   starts/ends at the interpolated 0C time. Any event that cannot be
%   assigned a definitive start and/or end time is not considered (ex.
%   events that run into the freeze-up end date). See steps (i) to (vii) in
%   subfunction SupercoolingEvents (or search for START OF DEFINITION and
%   END OF DEFINITION) for the lines of code that define an event.
%
%   Once defined, supercooling events are screened using the following
%   definitions:
%   1) Micro Events: Events that are shorter than the minimum duration in
%   minutes (set when logging the raw data files)
%   2) Mild Events: Events that never record water temperatures less than
%   the sensor accuracy of the sensor (set when logging the raw data files)
%   3) Extreneous Temperature Events: Events that have a Peak Supercooling
%   Temperature below the negative threshold (defined in rawfilelist)
%   4) negTDMF Events: Events that are removed due to having most of the
%   temperature above 0 C (may become relevant if definition of a
%   supercooling event is changed)
%   5) Manually Screened: Events that fall between -0.1 C and the negative
%   threshold are manually screened by the user for anomalies (this process
%   can be skipped). It should be noted that if this step is skipped, the
%   flagged events are added to the database and not checked again.
%
%I) NOTES ON ADDING ADDITONAL PARAMETERS TO THE SUPERCOOLING EVENTS TABLE
%The Event Table for the Deployment Period is generated in the
%SupercoolingEvents aubfucntion. Search for (I) for the section of code
%where the table generation sub-function outputs to the program and for the
%sub-function in order to make your desired changes. Make sure to test that
%your table formatting and unit assignment is correct.
%
%II) NOTES ON ANALYSIS FROM CALCULATED PARAMETERS
%After the Events are added to the table, additional filtering/analysis
%can be done based on calculated parameters. Search for (II) for the
%section of code where the analysis sub-function outputs to the
%program and for the sub-function in order to make your desired changes.
%In the interest of tracking information, it is good practice to add the
%variable to the Observation Summary Table. Follow the same procedure as
%the other discarded event trackers, by making the variable an output of
%SupercoolingEvents. Remember that you will likely have to adjust
```

```
%formatting of the Observation Summary Table (search for ****START OF
%FORMATTING OBSERVATION SUMMARY and ****END OF FORMATTING OBSERVATION
%SUMMARY for the relavent sections of code.
%
%III) NOTES ON MANUAL SCREENING OF EVENTS
%During the processing of each file, the program will flag any events that
%fall under the criterion to be manually screened search for (III) for the
%section of code where the analysis sub-function outputs to the
%program and for the sub-function in order to make your desired changes.
%This will add a temporary column to the Supercooling Event Table that will
%be used to denote which events need to be manually reviewed by the user.
%Once the review is completed, the column is removed and the Database is
%reorganized.
%
% If additional criteria for manual screening are to be added, and thus
% the possibility of multiple flags on an event, review the auotomated part
% of the flagging process to improve efficency in perserving the details as
% to why an event was flagged.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Author: Sean R. Boyd January 19th, 2022
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc;
pausedur = 0;%Pause length for readability of printouts in command window
%window (Set for personal preference)
%%%%%%%%
fprintf('Loading Required Files...')
%Determine current folder (direct) then generate directories
direct=pwd;filesLists=strcat(pwd,'\Raw Data Files');DB=strcat(pwd,'\Databases');
tools=strcat(direct,'/RSKTools');
%Make sure RSKTools is part of your directory path
addpath(genpath(tools));
savepath(strcat(tools,'/pathdef.m'))
%Load required files
try %Load New Site Data Summary Table
    cd (filesLists)
    load('CurrentRawFileList.mat','rawfileList');
catch
    cd (direct)
    error(sprintf("'CurrentRawFileList.mat' cannot be found in current directory.\nPlease create
file and try again."))
end
%Get the list of files to be processed
NewRawFiles = rawfileList(rawfileList.Processed == "No",:);
```

```
%Load Databases
cd (DB)
if exist('CurrentSupercoolingDatabase.mat','file')~=0 %Load SupercooligDatabase
    load('CurrentSupercoolingDatabase.mat','SupercoolingDatabase');
else
    SupercoolingDatabase =
struct('Observation_Summaries',struct('Rivers',{},'Sites',struct('Site',{},'Deployments',table())
),...
'Event_Table',table(),'Time_Series',struct('Deployments',struct('Deployment_ID',{},'Deployment_Da
taTable',timetable()),...

'Supercooling_Events',struct('Event_ID',{},'Event_DataTable',timetable())),'Screened_Periods',str
uct('Summary',table(),...

'TimeSeries',struct("TimeSeries_ID",[],"TimeSeries",[])),'Archive_Notes',struct('Previous_Databas
e','unknown','Change_List',table()));
end
%Get Screened Periods records
screenedPeriods = SupercoolingDatabase.Screened_Periods;
%Determine if the deployment IDs need to be updated for the screened events
if ~isempty(screenedPeriods.Summary)
    for k = 1:height(screenedPeriods.Summary)
        %Get the file name and deployment id from this row
       f =  screenedPeriods.Summary.File_Name(k);
       d = screenedPeriods.Summary.Deployment_ID(k);
       ts = screenedPeriods.Summary.TimeSeries_ID(k);
       des = ts{:}(strfind(ts,'_')+1:end);
       %Find the file in the file list and deployment ID
       r = find(rawfileList.File_Name == f,1);depID = rawfileList.Deployment_ID(r);
       %If it differs from the Screened Event,update the Screened Event Records
       if ~isequal(d,depID)
          screenedPeriods.Summary.Deployment_ID(k) =  depID;
          screenedPeriods.Summary.TimeSeries_ID(k) = sprintf('%s_%s',depID,des);
          %Find and update the TIme Series ID
          TS = string({screenedPeriods.TimeSeries.TimeSeries_ID});
          r1 = find(TS == ts,1);screenedPeriods.TimeSeries(r1).TimeSeries_ID =
sprintf('%s_%s',depID,des);
       end
    end
end
%Create a storage array in case files are skipped in the processing due to
%duplication or other issues
skipList=zeros(height(NewRawFiles),1);skipCount=0;cd(direct)
%Process files and add them to the database
if ~isempty(NewRawFiles)
    for k = 1:height(NewRawFiles)
        clc;fprintf('File %d of %d\n\n',k,height(NewRawFiles))
        fileName=NewRawFiles.File_Name{k};
        %Generate and correct the formatting on date-times
        summary=NewRawFiles(k,:);summary.Properties.RowNames=NewRawFiles.Deployment_ID(k);
```

```
        %Create array of seasons and get minimum duration from rawfileList
         Fall= NewRawFiles.Fall_Start(k);
         Winter= NewRawFiles.Winter_Start(k);
         Spring= NewRawFiles.Spring_Start(k);
         seasonStarts = [Fall,Winter,Spring];
         minDurationMinutesEvents = NewRawFiles.Minimum_Event_Duration_Minutes(k);
         %Generate AnalysisSummary structure for the file
         cd (filesLists)

[AnalysisSummary]=SuperCoolingDataProcessing(fileName,summary,minDurationMinutesEvents,seasonStar
ts);
         cd (direct)
         %Add AnalysisSummary to Database
         fprintf('\nAdding supercooling events to Database...')

[SupercoolingDatabase,exitCue]=UpdateSupercoolingDatabase(AnalysisSummary,SupercoolingDatabase);
         if exitCue == 1
             skipCount=skipCount+1;skipList(skipCount,1) = k;fprintf('Skipped');pause(pausedur)
         else
             num = find(rawfileList.File_Number == NewRawFiles.File_Number(k),1);
             rawfileList.Processed(num) = "Yes";fprintf('Complete');pause(pausedur)
         end
     end
     clc;skipList(skipList==0)=[];
     %If Files were skipped, print list of files that were not processed
     if isempty(skipList) == 0
         disp('The following files were not processed due to duplication errors during
processing:')
         disp(NewRawFiles(skipList,:))
         fprintf('Check the raw file list for errors/duplications. If you wish to overwrite
data\ncurrently causing conflict in the database, use AddAlterData to overwrite the required data
and reprocess the files\n')
         cont = input('hit Enter to continue:','s');
     else
         cont=[];
     end
     if isempty(cont)
         cd (direct);clc;fprintf('Processsing of Files Completed & Database
Updated\n');pause(pausedur);clc;
     end
     %Manually check all flagged events
     if ~isequal(skipList',(1:height(NewRawFiles))) %Not all files were skipped
         fprintf('Searching Database for supercooling events to be manually screened...')
         T=SupercoolingDatabase.Event_Table; %Get the full data table
         if ~isempty(find(string(T.Properties.VariableNames) == "Manual_Check",1))
             flags = T.Manual_Check;
             flagchk = flags;flagchk(flagchk~=0)=1; %change flagchk to logic array
         else %No manual checks were flagged
             flagchk=0;
         end
         if sum(flagchk)>0%There are events flagged
             fprintf('\nFlagged events found...')
```

```
        if ~isempty(screenedPeriods.Summary)
                fprintf('\nComparing flagged events to records of screened time periods...')
            end
            %Get all time series in the database
            ts = SupercoolingDatabase.Time_Series.Supercooling_Events;
            %Create an index for every event, and remove all events that were
            %not flagged
            id = (1:length(ts))';
            id=id.*flagchk;id(id==0)=[];
            ts=ts(1,id);
            %Create Event Table summary
            T1 = T(id,[6,9,10]);
            %Get the file name for these ids
            for r = 1:height(T1)
               T1.File_Name(r) =  rawfileList.File_Name(find(rawfileList.Deployment_ID ==
T1.Deployment_ID(r),1));
            end
            T1 = T1(:,[4,1:3]);
            %Compare table to the previous records
            if ~isempty(screenedPeriods.Summary)
                fileName = screenedPeriods.Summary.File_Name;addIndex = zeros([height(T1),1]);
%List of deployments already screened and the storage array for the index of events to add
                for r = 1:height(T1)
                    rows = find(fileName==T1.File_Name(r));
                    if ~isempty(rows) %Is this period potentially already registered
                        sTab = screenedPeriods.Summary(rows,:);
                        s = sTab.Start_Time;e = sTab.End_Time; %Start and End Times of logged
period
                        schk = find(s == T1.Start_Time(r));echk = find(e == T1.End_Time(r),1);
                        if ~isempty(schk) && ~isempty(echk)
                            for r1 = 1:min(length(schk),length(echk))
                                if isequal(schk(r1),echk(r1)) %The start and end times of the T1
pair match an previously screened start and end time pair
                                    %Check if this event was discarded
                                    %previously
                                    if isequal(sTab.Decision(schk),"Keep")
                                        continue
                                    else%If it was discarded, automatically flag it for removal
(no query)
                                        addIndex(r) = -rows(schk(r1));
                                    end
                                else
                                    addIndex(r)=r;
                                end
                            end
                        else %This is a different period. Add it to the screened  periods summary
                            addIndex(r)=r;
                        end
                    else
                        addIndex(r)=r;
                    end
                end
                %Delete all zero add indicies
                addIndex(addIndex==0)=[];
```

```
                %Determine if there were previously discarded events
                r = find(addIndex<0);
                if~isempty(r)
                    del = -addIndex(r);%Get teh actual index for ts
                    [T,SupercoolingDatabase]=removeDiscardedEvts(del,ts,T,SupercoolingDatabase);
                end
                %Delete all negative add indicies
                addIndex(addIndex<0)=[];
                if ~isempty(addIndex) %New events were found
                    summary = table('Size',[length(addIndex) width(screenedPeriods.Summary)],...

'VariableType',["string","string","datetime","datetime","string","string","string"],'VariableName
s',...

["File_Name","Deployment_ID","Start_Time","End_Time","Reason_Flagged","Decision","TimeSeries_ID"]
);
                    summary=[screenedPeriods.Summary;summary];rsum =
height(screenedPeriods.Summary)+1;
                    if ~isempty(addIndex)
                        for r = 1:length(addIndex)
                            addTab = T1(addIndex(r),:);addTab.Decision="TBD";
                            addTab.TimeSeries_ID=strcat(addTab.Deployment_ID,"_TBD");
                            %Determine the reason flagged (STORE REASON
                            %FLAGGED). Since there is currently only one reason
                            %to flagg events, there are no cases to check.
                            addTab.Reason_Flagged = "Peak Temperature <-0.1 deg C";
                            %Add the new time period to the table
                            summary(rsum,:)=addTab;
                            %Add Time series to structure
                            screenedPeriods.TimeSeries(end+1).TimeSeries_ID =
addTab.TimeSeries_ID;

screenedPeriods.TimeSeries(rsum).TimeSeries=ts(addIndex(r)).Event_DataTable;
                            rsum=rsum+1;
                        end
                        T1 = summary; %Create a single variable so that the two cases use the
same varaibles during later assignments
                        T2=T1(T1.Decision == "TBD",:); dispIndex = find(T1.Decision == "TBD");
%Generate display table and the indicies of the events being screened
                    else
                        fprintf('\nAll events flagged were screened previously.\n');T2=[];
                    end
                else
                    fprintf('\nAll events flagged were screened previously.\n');T2=[];
                end
            else %No records to compare to. Use T1 as the starting table
                T1.Reason_Flagged(:)="TBD";T1.Decision(:)="TBD";
                %Determine reason flagged
                for r = 1:length(id)
                    switch flags(id(3))
                        case 1
                             T1.Reason_Flagged(r) = "Peak temperature < -0.1 deg C";
                    end
                end
```

```
                for r = 1:height(T1)
                    T1.TimeSeries_ID(r)=strcat(T1.Deployment_ID(r),"_TBD");
                end
                T2=T1; %Single table variable to display later on
                dispIndex = (1:height(T1)); %Index of displayed table is the same as the record
table
                addIndex = dispIndex; %addIndex is the same as row index (since the whole table
is added to the record)
                for r = 1:length(ts) %Add all the time series to the record
                    screenedPeriods.TimeSeries(r).TimeSeries_ID = T1.TimeSeries_ID(r);
                    screenedPeriods.TimeSeries(r).TimeSeries=ts(r).Event_DataTable;
                end
            end
            %If new records were found, add their time series to the
            %screening record

            %Print out summary table of 'TBD' time periods
            str = sprintf('Do you wish to review events that have been flagged for manual
screening?\nNote that any flagged events that are not reviewed will be flagged\nin the future.
(Y/N)?: ');
            if isempty(T2) %No events to be screened
                fprintf('\nNo supercooling events to be manually screened\n\n')
                 T.Manual_Check=[];SupercoolingDatabase.Event_Table = T;
            else %Print out each event to make it easy to manually screen
                clc;format short;disp(T2);fprintf('\n');
                inpt = input(str,'s');chk=1;chk1=1;
                while chk
                    if strcmpi(inpt,'y') %User wishes to screen events
                        chk=0;
                    elseif strcmpi(inpt,'n') %User does not wish to screen events
                        chk=0;chk1=0;
                    else %incorrect input; re-request
                        fprintf('Incorrect Input\n')
                        inpt = input(str,'s');
                    end
                end
                if chk1 %Manually screen events
                    del = zeros(length(ts),1);delcount=0;close all %Stores the indicies of events
to be cleared
                    for k = 1:height(T2) %Go down list. The table will be in the same order as
the addIndex reference, which can be used when selecting the time series
                        clc;disp(T2);r = addIndex(k);
                        fprintf('%d of %d: %s\n',k,length(addIndex),ts(r).Event_ID)
                        tt = ts(r).Event_DataTable;
                        plot(tt.Time,tt.Water_Temperature);grid on;xlabel('Date-
Time');ylabel('Water Temperature ({o}^C)');title(strrep(ts(k).Event_ID,'_'," "))
                        inpt=input('Keep Supercooling Event (Y/N)?: ','s');chk=1;
                        while chk
                            if strcmpi(inpt,'y') %User wishes to keep the event
                                chk=0;T2.Decision(k) = "Keep";
                            elseif strcmpi(inpt,'n') %User does not wish to keep the event
                                inpt1=input('Confirm Clearing Supercooling Event (Y/N)?:
','s');chk3=1;
                                while chk3
```

```
                                    if strcmpi(inpt1,'y') %User wishes to delete event
                                        chk3=0;chk=0;T2.Decision(k) = "Discard";
                                        delcount=delcount+1;del(delcount,1)=k;
                                    elseif strcmpi(inpt1,'n') %User does not wish to delete event
                                        chk3=0;chk=0;
                                    else %incorrect input; rerequest
                                        fprintf('Incorrect Input\n')
                                        inpt1 = input('Confirm Clearing Supercooling Event
(Y/N)?: ','s');
                                    end
                                end
                            else %incorrect input; rerequest
                                fprintf('Incorrect Input\n')
                                inpt = input('Keep Supercooling Event (Y/N)?: ','s');
                            end
                        end
                        %Update display table and records
                        T2.TimeSeries_ID(k) = strrep(T2.TimeSeries_ID(k),"TBD", T2.Decision(k));
                        T1.TimeSeries_ID(dispIndex(k)) = T2.TimeSeries_ID(k);
T1.Decision(dispIndex(k)) = T2.Decision(k);
                        screenedPeriods.TimeSeries(dispIndex(k)).TimeSeries_ID =
T2.TimeSeries_ID(k);
                        close all
                    end
                    %Store T1 in screened Periods
                    screenedPeriods.Summary = T1;
                    %Clear out all zero values in del & keep
                    del(del == 0,:)=[];
                    %If there are events to be deleted (indicated by del), remove the
                    %events from the database and relabel the required
                    %events
                    if ~isempty(del)

[T,SupercoolingDatabase]=removeDiscardedEvts(del,ts,T,SupercoolingDatabase);
                        %Remove Manual Check Column from the Table
                        T.Manual_Check=[];
                        %Add table back to database
                        SupercoolingDatabase.Event_Table = T;
                        %After all event labels are adjusted, relabel the event time
                        %series
                        for k = 1:height(T)
                            SupercoolingDatabase.Time_Series.Supercooling_Events(k).Event_ID =
T.Event_ID(k);
                        end
                    end
                end
                %Update Screened Periods records in the database
                SupercoolingDatabase.Screened_Periods = screenedPeriods;
                clc;fprintf('Screening of Events Completed');pause(pausedur);clc;
            end
        else
            %Remove Manual Check Column from the Table
            T.Manual_Check=[];
```

```
        %Add table back to database
            SupercoolingDatabase.Event_Table = T;
            clc;fprintf('No supercooling events to be manually screened\n')
        end
        %Organise and save SupercoolingDatabase to file
        fprintf('Organising Database Structure')
        fprintf('\n    >River Site Summaries...')
        [~,index] = sortrows({SupercoolingDatabase.Observation_Summaries.Rivers}.');
        SupercoolingDatabase.Observation_Summaries =
SupercoolingDatabase.Observation_Summaries(index);
        clear index;
        %       Sort by Site (Go through each river)
        for river = 1:length(SupercoolingDatabase.Observation_Summaries)
            numsites = length(SupercoolingDatabase.Observation_Summaries(river).Sites);
            if numsites>1 %Sort the sites
                [~,index] =
sortrows({SupercoolingDatabase.Observation_Summaries(river).Sites.Site}.');
                SupercoolingDatabase.Observation_Summaries(river).Sites =
SupercoolingDatabase.Observation_Summaries(river).Sites(index);clear index;
            end
            %Sort by Deployment_Period
            for site = 1:numsites
                if
height(SupercoolingDatabase.Observation_Summaries(river).Sites(site).Deployments)>1
                    SupercoolingDatabase.Observation_Summaries(river).Sites(site).Deployments =
sortrows(SupercoolingDatabase.Observation_Summaries(river).Sites(site).Deployments,'RowNames','as
cend');
                end
            end
        end
        fprintf('Complete')
        %   Super_Cooling_Catalogue (Alphabetical by River, Alphabetical by Site
        %   Within River, Chronological within Site)
        fprintf('\n    >Super Cooling Event Summary...')
        SupercoolingDatabase.Event_Table =
sortrows(SupercoolingDatabase.Event_Table,{'River','Site','Start_Time'});
        fprintf('Complete')
        %   Super_Cooling_Time_Series (same order as Catalogue; organised by sorting
        %   using the sorted catalogue Event_ID column)
        fprintf('\n    >Super Cooling Time Series')
        fprintf('\n       >Deployments...')
        rankedOrder =
sort(string({SupercoolingDatabase.Time_Series.Deployments.Deployment_ID}));rawOrder={Supercooling
Database.Time_Series.Deployments.Deployment_ID};
        for row = 1:length(rawOrder)
            chk=1;chkrow=1;current=rawOrder{row};
            while chk && chkrow<=length(rankedOrder)
                if  isequal(current,rankedOrder{chkrow})
                    chk = 0;SupercoolingDatabase.Time_Series.Deployments(row).Rank =  chkrow;
                else
                    chkrow=chkrow+1;
                end
            end
        end
```

```
        [~,index] = sortrows([SupercoolingDatabase.Time_Series.Deployments.Rank].');
        SupercoolingDatabase.Time_Series.Deployments =
SupercoolingDatabase.Time_Series.Deployments(index);clear index
        SupercoolingDatabase.Time_Series.Deployments =
rmfield(SupercoolingDatabase.Time_Series.Deployments,'Rank');
        fprintf('Complete\n')
        fprintf('       >Supercooling Events...')
        rankedOrder =
SupercoolingDatabase.Event_Table.Event_ID;rawOrder={SupercoolingDatabase.Time_Series.Supercooling
_Events.Event_ID};
        for row = 1:length(rawOrder)
            chk=1;chkrow=1;current=rawOrder{row};
            while chk && chkrow<=length(rankedOrder)
                if  isequal(current,rankedOrder{chkrow})
                    chk = 0;SupercoolingDatabase.Time_Series.Supercooling_Events(row).Rank =
chkrow;
                else
                    chkrow=chkrow+1;
                end
            end
        end
        [~,index] = sortrows([SupercoolingDatabase.Time_Series.Supercooling_Events.Rank].');
        SupercoolingDatabase.Time_Series.Supercooling_Events =
SupercoolingDatabase.Time_Series.Supercooling_Events(index);clear index
        SupercoolingDatabase.Time_Series.Supercooling_Events =
rmfield(SupercoolingDatabase.Time_Series.Supercooling_Events,'Rank');
        fprintf('Complete\n')
        fprintf('Organisation Complete\nSaving Database...');cd (DB)
        save ('CurrentSupercoolingDatabase.mat','SupercoolingDatabase');fprintf('Complete\n')
        fprintf('\nSaving Summary of Processed Files...')
        %Save List of Rawfiles
        cd (filesLists);
        save('CurrentRawFileList.mat','rawfileList')
        fprintf('Complete\n');
        fprintf('Processing and Screening of new files Complete!');cd(direct);pause(pausedur);clc
    end
else
    clc;fprintf('All Files in CurrentRawFileList.mat have already been processed.')
    cd(direct);pause(pausedur+3);clc
end
end

%SUB FUNCTIONS
%I)     Sub-Functions For ProcessNewRawFiles
%1) SuperCoolingDataProcessing
function[AnalysisSummary]=SuperCoolingDataProcessing(file,obsTab,minDurationMinutes,seasonStarts)
%SuperCoolingDataProcessingTables: Extracts and processes water temperature
%data from a raw file.  Required parameters are stored in the summary, and are used to
% process data as required.
%Greater detail of steps in the process can be found in the description of the relevant sub-
function
```

```
        %NOTE ABOUT ASSUMPTIONS:
%   This script assumes that one has RSKTools folder as part of the MATLAB
%   path
INPUTS
%   file: raw data file (requests if not supplied)
%   obsTab: Summary of observation summmary.  Uses the formatting generated
%   in NewRawFiles
%   minDurationMinutes: The minimum duration of events. This is hardcoded
%   as part of the program (see write-up)
%   seasonStarts - Dayes of the year that are estimated to be the start of a
%   particular Deployment_Period
%OUTPUTS
%   AnalysisSummary: Structure summarising the analysis of the raw file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Written by: Sean Boyd Last Updated October 08, 2021
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all;
format short e
AnalysisSummary=struct();%Initial value until properly assigned.
%Extract variables from site summary table
offset = obsTab.Sensor_Offset;sensorAccuracy=obsTab.Sensor_Accuracy; %Offset of sensor used at
site
negativeThreshold=obsTab.Negative_Threshold;deployID=obsTab.Deployment_ID; %Negative Threshold
used as cut-off for extraneous readings

%Extract Freeze Up Completion and Break Up start from site summary table

% Extract Freeze-Up and Break-Up dates
endFreezeUp = obsTab.Freeze_Up_End_Date;
startBreakUp = obsTab.Break_Up_Start_Date;

%Try to extract file first to determine if file can be accessed (EXT)
ext = find(file(:) =='.',1,'last');ext = file(ext+1:end);
switch ext
    case 'rsk' %.rsk file
try
    [DateTimes,dateNumbers,waterTemp]=rskdataextract(file,offset);
catch
    if exist(file,'file') == 0
        err = 1; %File cannot be found
    else
        err = 2;
    end
    if err == 1
        error('%s cannot be found in Raw Data Files\nPlease change the folder and try
again\n',file)
    elseif err == 2
        error('Cannot access RSKTools. Make sure the folder & subfolders are part of the
directory path.')
    end
end
```

```
    case 'csv' %.rsk file
        try
           tt = readtimetable(file);
           DateTimes = tt.Time;waterTemp = tt.Water_Temperature-offset;
           dateNumbers = datenum(DateTimes);
        catch
            if exist(file,'file') == 0
                err = 1; %File cannot be found
            else
                err = 2;
            end
            if err == 1
                error('%s cannot be found in Raw Data Files\nPlease change the folder and try
again\n',file)
            else
                error('Cannot correctly read file to MATLAB. Make sure %s presents data as a
Time/ Water_Temperature time table\n',file)
            end
        end
    otherwise %Error
        error('.%s extension is not currently readable by this program.\nSearch for EXT for the
section of code in which\nadjustments can be made to extract data from .%s files',ext,ext)
end
%Determine average time-step
avgTimeStp = mean(minutes(DateTimes(2:end)-DateTimes(1:end-1)));
%Print out Site Summary
fprintf('==================================================\n')
fprintf('SUMMARY FOR %s',obsTab.Row{:})
fprintf('\n   >River Name: %s',obsTab.River{1})
fprintf('\n   >Location Name: %s',obsTab.Site{1})
fprintf('\n   >Deployment Period: %s',obsTab.Deployment_Period{1})
%Print out the dates for the beginning and end of  seasons
%based on weather data
fprintf('\n     -> Fall: %s - %s',seasonStarts(1),seasonStarts(2)-hours(24))
fprintf('\n     -> Winter: %s - %s',seasonStarts(2),seasonStarts(3)-hours(24))
fprintf('\n     -> Start of Spring: %s',seasonStarts(3))
fprintf('\n   >Freeze-Up End Date: %s',endFreezeUp)
fprintf('\n   >Break-Up Start Date: %s',startBreakUp)
if ~isinf(negativeThreshold) %State what the negative threshold for the analysis discarding
events
    fprintf('\n   >Negative Temperature Threshold: %.3f C',negativeThreshold)
else
    fprintf('\n   >No Negative Temperatue Threshold')
end
fprintf('\n==================================================\n')
fprintf('Data extracted from %s',file)
fprintf('\nInitial Data Processing...')
%Tabulate water temperature data
rawDataTable = RawDataTable(DateTimes,dateNumbers,waterTemp);
%Filter out any data between the end of Freeze up and the start of break up
[filteredDataTable,timestep]=RemoveMidWinter(rawDataTable,endFreezeUp,startBreakUp);
```

```
%Generate Supercooling Events (I),(II)
[filteredDataTable,EventTab,RawEvents,MicroEvents,SmallTempEvents,ExtTemps,negCDMSEvents] = ...
SupercoolingEvents(filteredDataTable,sensorAccuracy,timestep,negativeThreshold,minDurationMinutes
,endFreezeUp,startBreakUp,seasonStarts);
fprintf('Complete\n')
%Extract the start and end of the supercooling Deployment_Period
if ~isnat(EventTab.Start_Time(1))
     count = height(EventTab);
    startSeason = EventTab.Start_Time(1);
    endSeason = EventTab.End_Time(end);
    fprintf('Start of Supercooling Observations: %s\n',startSeason)
    fprintf('End of Supercooling Observations: %s\n',endSeason)
    if ~isnat(startBreakUp)
        fprintf('Period of Intact Ice cover: %s - %s\n',endFreezeUp,startBreakUp)
        if endSeason>startBreakUp %There are break-up events. Remove the period of intact ice
cover
            Seasdur = days((endSeason-startSeason)-(startBreakUp-endFreezeUp));
        else
            Seasdur = days((endSeason-startSeason));
        end
    else
        fprintf('Period of Intact Ice cover: N/A\n')
        Seasdur = days((endSeason-startSeason));
    end
    fprintf('Effective Supercooling Period Duration: %.1f days\n',Seasdur)
else
    count = 0;
    startSeason = NaT;endSeason = NaT;Seasdur = NaN;
    fprintf('No supercooling events met the filtering protocals.\n')
    fprintf("Placeholder supercooling event marked as 'SC0'.\n")
    fprintf('Time Series and Observation Summary are stored for review.\n')
end
fprintf('%.0f Events Identified.\n',RawEvents)
fprintf('     ->%.0f Events Discarded as Micro Events (duration less than %.0f
minutes)\n',MicroEvents,minDurationMinutes)
fprintf('     ->%.0f Events Discarded as Mild Events (Minimum temperature greater than %.3f
degree C)\n',SmallTempEvents,-sensorAccuracy)
fprintf('     ->%.0f Events Discarded due to Extraneous Temperatures (Minimum Temperature below
%.3f degree C)\n',ExtTemps,negativeThreshold)
fprintf('     ->%.0f Events Discarded due to a negative Total Degree Minutes of Freezing (Water
Temperature is mostly above 0 degree C)\n',negCDMSEvents)
fprintf('%.0f Events Catalogued and Classified.\n',count)
fprintf('=================================================\n')
fprintf('Compiling Deployment Period Analysis...')
%****START OF FORMATTING OBSERVATION SUMMARY
%Add the analysis summary to Event summary table
obsTab.Observations_Start = startSeason;obsTab.Observations_End = endSeason;
obsTab.Effective_Observation_Duration = Seasdur;
obsTab.Number_Events_Identified = RawEvents;
obsTab.Number_Events_Catalogued = count;
obsTab.Number_Micro_Events = MicroEvents;
obsTab.Number_Mild_Events = SmallTempEvents;
obsTab.Number_Extraneous_Temperature_Events = ExtTemps;
obsTab.Number_Negative_CDMS_Events = negCDMSEvents;
```

```
obsTab.Average_Time_Step_Minutes = avgTimeStp;
obsTab.Number_Manually_Screened_Events = 0;%Default value

%Rearrange for readability. Note that the first 6 columns are for
%information required to store events in the database, and cannot be moved
%out of order.
obsTab = obsTab(:,[7:12,2:6,30,16,17,21:29,31,18,19]);
%Add units
obsTab.Properties.VariableUnits = {'','','','','','','','degree C','degree C','degree
C','minutes','minutes','date-time','date-time',...
'date-time','date-
time','days','events','events','events','events','events','events','events','',''};
%****END OF FORMATTING OBSERVATION SUMMARY
%Full deployment and Event summary and time series for the sensor
Time_Series=struct('Deployments',struct('Deployment_ID',deployID),'Supercooling_Events',struct('E
vent_ID',[])); %Structure to hold all time series
if count>0 %There are events catalogued
    for k = 1:height(EventTab)+1
        if k == 1 %Full Deployment_Period
            Time_Series(1).Deployments(1).Deployment_DataTable=filteredDataTable;
        else %Supercooling Event
            Time_Series(1).Supercooling_Events(k-1).Event_ID =
strcat(deployID{:},'_',EventTab.Properties.RowNames{k-1});
            Time_Series(1).Supercooling_Events(k-
1).Event_DataTable=filteredDataTable(EventTab.Start_Index(k-1):EventTab.End_Index(k-1),:);
        end
    end
else %No events catalogued. Store the full deployment and a placeholder for the time series
(empty)
    Time_Series(1).Deployments(1).Deployment_DataTable=filteredDataTable;
    Time_Series(1).Supercooling_Events(1).Event_ID =
strcat(deployID{:},'_',EventTab.Properties.RowNames{1});
    Time_Series(1).Supercooling_Events(1).Event_DataTable=timetable();
end
%Remove Indicies from Tables (not used in database)
EventTab.Start_Index=[];EventTab.End_Index=[];
%Store analysis in 1 structure
AnalysisSummary.Observation_Summary=obsTab;AnalysisSummary.Event_Table=EventTab;AnalysisSummary.T
ime_Series=Time_Series;fprintf('Complete');
end

%2)UpdateSupercoolingDatabase;
function
[SupercoolingDatabase,exitCue]=UpdateSupercoolingDatabase(AnalysisSummary,SupercoolingDatabase)
%Set exitCue to 0 to allow for database to update
exitCue=0;
%Extract Summary Table from Analysis summary
siteTable =  AnalysisSummary.Observation_Summary; %Condense variable name down for readability
summary2=AnalysisSummary.Event_Table;summary2.Properties.RowNames={}; %Extract Event Parameters
from Deployment
summary1=table('Size',[height(summary2),8],'VariableType',...
    {'string','string','string','string','string','string','string','string'},...
'VariableName',{'River','River_ID','Site','Site_ID','Deployment_Period','Deployment_ID','Event','
Event_ID'}); %Build a table for all the ID Tags for the Site/Deployment Period
```

```
summary1.River(:)=siteTable.River{:}; %Assign River
summary1.River_ID(:)=siteTable.River_ID{:}; %Assign River ID Tag
summary1.Site(:)=siteTable.Site{:}; %Assign Site
summary1.Site_ID(:)=siteTable.Site_ID{:}; %Assign Site ID Tag
summary1.Deployment_Period(:)=siteTable.Deployment_Period{:}; %Assign Year
summary1.Deployment_ID(:)=siteTable.Row{1}; %Assign Site/Deployment_Period ID Tag
if ~isnat(summary2.Start_Time(1))
for row = 1:height(summary1)
    summary1.Event(row)=strcat('SC',num2str(row)); %Assign Event Number for the Deployment_Period
    summary1.Event_ID(row)=strcat(summary1.Deployment_ID(row),'_SC',num2str(row)); %Assign Event
ID Tag
end
else %Placeholder 'null event'
   summary1.Event(1)=strcat('SC','0'); %Assign Event Number for the Deployment_Period
    summary1.Event_ID(1)=strcat(summary1.Deployment_ID(1),'_SC0'); %Assign Event ID Tag
end
SCsummaryTable=[summary1,summary2]; %Consolidate Table
%Time-Series
superCoolingTimeSeries=AnalysisSummary.Time_Series; %Extract Time Series
%Determine if this river is already a part of the database, and add
%it if required
currentRivers = {SupercoolingDatabase.Observation_Summaries.Rivers};k=1;chk=1;
while chk && k<=length(currentRivers) %Find where a River occurs in the current listing of
Rivers, or adds a new river
    if strcmp(currentRivers{k},siteTable.River{1})
        chk=0; %Stop loop; k = River Index
    else
        k = k+1;
    end
end
if chk %If the current river is a new river, add the river to the database list
    SupercoolingDatabase.Observation_Summaries(k).Rivers=siteTable.River{1};
    SupercoolingDatabase.Observation_Summaries(k).Sites(1).Site=siteTable.Site{1}; %Adds a new
site (new river)
    SupercoolingDatabase.Observation_Summaries(k).Sites(1).Deployments=siteTable(:,7:end); %Adds
Deployment_Period to Site Table Listing
else % k is the location of the current river in the structure.  Either this is a new site, or a
new Deployment_Period at the same site
    currentSite=siteTable.Site{1}; %Current site being considered
    %Determine if the current site is already in the database
    k1=1;chk=1;allsites={SupercoolingDatabase.Observation_Summaries(k).Sites.Site}; %Run through
the list of all sites on the river to determine if this site already exists
    while k1<=length(allsites) && chk
        if  strcmp(allsites{k1},currentSite)
            chk=0; %Stop loop; k1 = site index
        else
            k1=k1+1;
        end
    end
    if chk == 0 %the current site is already in the database
        entryNum=height(SupercoolingDatabase.Observation_Summaries(k).Sites(k1).Deployments)+1;
```

```
%The row that will be added to the site Deployment_Period summary table
        try
Rows=[SupercoolingDatabase.Observation_Summaries(k).Sites(k1).Deployments.Properties.RowNames;sit
eTable.Row{:}];

SupercoolingDatabase.Observation_Summaries(k).Sites(k1).Deployments(entryNum,:)=siteTable(:,7:end
);

SupercoolingDatabase.Observation_Summaries(k).Sites(k1).Deployments.Properties.RowNames =Rows;
        catch
            SupercoolingDatabase.Observation_Summaries(k).Sites(k1).Deployments(entryNum,:)=[];
            exitCue = 1;return
        end
    else %The current site is a new site to be added to the database
        SupercoolingDatabase.Observation_Summaries(k).Sites(k1).Site=siteTable.Site{1}; %Add Site
Listing
        SupercoolingDatabase.Observation_Summaries(k).Sites(k1).Deployments=siteTable(:,7:end);
%Add Deployment_Period to the Site
    end
end
%Add the Super-Cooling Summary Table to the Super-Cooling Catalogue
if ~isempty(SCsummaryTable) %There are events to add
    if isempty(SupercoolingDatabase.Event_Table) %First Table added to an empty Database
        SupercoolingDatabase.Event_Table=SCsummaryTable;
    else
        %Add a column for manual flagging if it does not exist (set to 0 since they would all
        %have been manually processed previously)
        if isempty(find(string(SupercoolingDatabase.Event_Table.Properties.VariableNames) ==
"Manual_Check",1))
            SupercoolingDatabase.Event_Table.Manual_Check =
zeros([height(SupercoolingDatabase.Event_Table),1]);
        end
        %Determine column sizes for the parameters with variable width
        [rowDB,colDB]=size(SupercoolingDatabase.Event_Table.Time_of_Secondary_Peak_Supercooling);
%Rows and columns in the Minimum Temperature (and related Paramters) of Database
        [rowSC,colSC]=size(SCsummaryTable.Time_of_Secondary_Peak_Supercooling); %Rows and columns
in the Minimum Temperature (and related Paramters) of Deployment_Period being added
        if colSC<colDB %New entery does not have the correct size for Time of Minimum
Temperature. Adjust size by adding sub-columns
            %Build Placeholders for Time of Minimum Temp and related Paramaters.  Same number of
rows as the new entry, but number
            %columns equal to the database
            allNaT = NaT(rowSC,colDB);
            %Add the new entry's data to the Placeholder.  Now in a matrix of
            %consistant size to the Database
            allNaT(:,1:colSC) = SCsummaryTable.Time_of_Secondary_Peak_Supercooling(:,:);
            %Clear the old columns and add the new placeholders to the new
            %entry

SCsummaryTable.Time_of_Secondary_Peak_Supercooling=[];SCsummaryTable.Time_of_Secondary_Peak_Super
cooling=allNaT;
```

```
        elseif colSC>colDB %Current Database does not have the correct size for
Time_of_Secondary_Peak_Supercooling. Adjust size by adding sub-columns
            %Build Placeholders for Time of Minimum Temp and related Paramaters.  Same number of
rows as the Database, but number
            %columns equal to the new entry
            allNaT = NaT(rowDB,colSC);
            %Add the new entry's data to the Placeholder.  Now in a matrix of
            %consistant size to the New Entry
            allNaT(:,1:colDB) =
SupercoolingDatabase.Event_Table.Time_of_Secondary_Peak_Supercooling(:,:);
            %Clear the old columns and add in the new columns.  Reshape matrix
            %to be the same order as before
            SupercoolingDatabase.Event_Table.Time_of_Secondary_Peak_Supercooling=[];
            SupercoolingDatabase.Event_Table.Time_of_Secondary_Peak_Supercooling=allNaT;
            SupercoolingDatabase.Event_Table=SupercoolingDatabase.Event_Table(:,[1:13,22,14:21]);
%Reshapes the summary table to desied column order
        end
        %Combine the two tables
        SupercoolingDatabase.Event_Table=[SupercoolingDatabase.Event_Table;SCsummaryTable];
    end
end
%Add the Time Series to the Database
%All entries will have the same format as the first, and need to have the data added
Fieldnames = fieldnames(SupercoolingDatabase.Time_Series); %Fieldnames of the overall structure
for f1 = 1:length(Fieldnames) %Enter each sub structure to enter in data
    Fieldnames1 = fieldnames(SupercoolingDatabase.Time_Series.(Fieldnames{f1})); %Fieldnames of
the current sub structure
    startChk = isempty(SupercoolingDatabase.Time_Series.(Fieldnames{f1}));%First set of time
series to be added to the field
    if startChk
        startRow =1;
    else
        startRow=1+length( SupercoolingDatabase.Time_Series.(Fieldnames{f1})); %Start Row for
adding the next set of time series
    end
    for k1 = startRow:startRow+length(superCoolingTimeSeries.(Fieldnames{f1}))-1
        for f2 = 1:length(Fieldnames1) %add data from analysis summary to database structure in
the current row
            try
                if~isempty(superCoolingTimeSeries.(Fieldnames{f1})(k1-
startRow+1).(Fieldnames1{f2}))
                SupercoolingDatabase.Time_Series.(Fieldnames{f1})(k1).(Fieldnames1{f2}) =...
                    superCoolingTimeSeries.(Fieldnames{f1})(k1-startRow+1).(Fieldnames1{f2});
                else %Do not add to the database structure
                    continue
                end
            catch %Field does not exist. Do not add to the database structure
                continue
            end
        end
    end
end
end
```

```
        %II) Sub-Functions For SuperCoolingDataProcessing
%1).rsk file Extraction
function [DateTimes,dateNumbers,waterTemp]= rskdataextract(file,offset)
narginchk(0,2);
if nargin ==0
    file=input('Enter of name for data to be extracted (or enter x to close function): ');
    offset = 0;
elseif nargin == 1
    offset = 0;
end
checkSize=size(file);
switch checkSize(2)
    case checkSize(2)<=1 %A file is entered (a valid .rsk file name is always greater than 1
char)
        if file=='x' || file=='X'
            return
        else
            disp('Invalid file name.')
            return
        end
    otherwise
        if strcmp(file(length(file)-3:length(file)),'.rsk') == 0
            file=strcat(file,'.rsk');
        end
        try %Try opening files to MATLAB
        rsk=RSKopen(file);
        rsk=RSKreaddata(rsk);
        data=rsk.data;
        dateNumbers=data.tstamp;
        DateTimes = datetime(dateNumbers,'ConvertFrom','datenum','Format','dd-MMM-yyyy HH:mm');
        waterTemp=data.values-offset;
        catch
            error('Cannot access RSKTools')
        end
end
end
%2)Raw Data Processing
function [rawdatatable] = RawDataTable(Time,dateNumber,waterTemp)
%RawDataProcessing Part of the SuperCoolingDataProcessing series, this
%fuction takes the data extracted from a .rsk file and generates a
%timetable of the timeseries, along with the slope of the temperature curve
%NOTE ON ASSUMPTIONS:
% To correct for NaT points in the datetimes, this function uses the
% submitted dateNumber value, or assumes an even time spacing an sets the
% time at the midpoint between the two known time values
%INPUT
%   Date_Time: The array of the date times for the water temperature series
%   dateNumber: the array of the serial date number for the water
%   temperature series
%   waterTemp: the array of water temperature fom the .rsk file
%OUTPUT
%   rawdatatable: timetable of the raw data from the .rsk file, along with
%   the foreword moving slope of the temperature curve
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
rawdatatable=timetable(Time,(1:length(Time))',dateNumber,waterTemp,...
    'VariableNames',{'Row_Index','Serial_Date_Number','Water_Temperature'});
%Calculated data
%   Slope of cooling curve (measure foreward)
changeTemp = rawdatatable.Water_Temperature(2:height(rawdatatable)) -
rawdatatable.Water_Temperature(1:height(rawdatatable)-1);
durationArray=minutes(rawdatatable.Time(2:height(rawdatatable)) -
rawdatatable.Time(1:height(rawdatatable)-1));
rawdatatable.Water_Temperature_Slope=[changeTemp./durationArray;NaN];
%Check for NaT appearing in data
natArray = find(isnat(rawdatatable.Time') == 1); %Find where NaT occurs in data
if isempty(natArray) == 0
    for k = 1:length(natArray)
        prev = rawdatatable.Time(natArray(k)-1,1);next=rawdatatable.Time(natArray(k)+1,1);
%Datapoints on either side of the NaT point
        datenumerr = 0; %Assume that the serial date numbers are correct
        if ~isnan(dateNumber(natArray(k))) %There is a known datenumber for the time
            date=datetime(datestr(rawdatatable.Serial_Date_Number(k,1)));
            %Check that the date number falls between the previous and next
            %datetimes.  If it does not, set datenumerr to 1
            if date>=next || date<=prev
                datenumerr=1;
            end
        end
        if isnan(dateNumber(natArray(k))) || datenumerr %If there is no known datenumber, or it
gives an erroneous time
            rawdatatable.Time(k,1)=prev+0.5*(next-prev); %The NaT point is midway between the two
known points
            rawdatatable.Serial_Date_Number(k)=datenum(rawdatatable.Time(k,1));
        end
    end
end
%Add units to raw data table
rawdatatable.Properties.VariableUnits = ["","","degree C","degree C/minute"];
end
%3) Remove Mid Winter
function [filtereDataTable,timestep]=RemoveMidWinter(rawdatatable,endFreezeUp,startBreakUp)
%RemoveMidWinter: Compares the date time of the raw water temperature data,
%and sets all data between the end of freeze-up and the start of break up
%to NaT and Nan
%INPUTS
%  rawdatatable: timetable of the extracted data
%OUTPUTS
%   filtereddatatable; timetable with all data between Freeze Up and Break Up removed
%   timestep: assumed regular timestep for the filtered time table (ignore
%   the skip in the data)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Set the filtered datatable equal to the raw datatable
filtereDataTable=rawdatatable;
filtereDataTable.Raw_Water_Temperature = rawdatatable.Water_Temperature; %Create a column for the
unfiltered water temperature data.
filtereDataTable.Raw_Water_Temperature_Slope = filtereDataTable.Water_Temperature_Slope; %Create
a column for the unfiltered water temperature slope.
```

```
        if ~isnat(endFreezeUp) %Both period boundaries are datetimes
    index=filtereDataTable.Row_Index; %Index of all rows

pastFreezeUp=filtereDataTable.Time>endFreezeUp;beforeBreak=filtereDataTable.Time<startBreakUp;%Lo
gic arrays of the datatable times for times past freeze up and before breakup
    index=index.*pastFreezeUp.*beforeBreak;index(index==0)=[];%Index of datatable rows in
MidWinter
   if ~isempty(index) %There is a Midwinter timeskip in the data

filtereDataTable.Water_Temperature(index)=NaN;filtereDataTable.Water_Temperature_Slope(index)=NaN
;
        %Determine Time Step of the datatable
        if index(end)+2<=length(rawdatatable.Time) %If the index of Midwinter Events does not
reach the end of the time-series
            timesteps=[rawdatatable.Time(2:index(1)-1)-rawdatatable.Time(1:index(1)-
2);rawdatatable.Time(index(end)+2:end)-rawdatatable.Time(index(end)+1:end-1)];
        else %The timeseries ends before Break-up starts at the site, resulting in index
including the end of the time series
            timesteps=rawdatatable.Time(2:index(1)-1)-rawdatatable.Time(1:index(1)-2);
        end
    else %There is no Midwinter Timeskip in the data
        %Determine Time Step of the datatable
        timesteps = rawdatatable.Time(2:end)-rawdatatable.Time(1:end-1);
    end
else %Freeze-up never ends
    %Determine Time Step of the datatable
    timesteps = rawdatatable.Time(2:end)-rawdatatable.Time(1:end-1);
end

%Determine Unique Timesteps.  If there are multiple unique timesteps,
%all time steps will be used in classifying events
timestep=unique(timesteps);
if isempty(timestep)
   error('There is an issue with either time indexing of the raw data file or the freeze-
up/break-up dates as no time steps can be calculated.')
end
end

%4)Super-Cooling Events
function [dataTable,EventTab,RawEvents,MicroEvents,SmallTempEvents,ExtTempsEvents,negCDMSEvents]
=
SupercoolingEvents(dataTable,sensorAccuracy,timestep,negativeThreshold,minDurationMinutes,endFree
zeUp,startBreakUp,seasonStarts)
%SupercoolingEvents Part of the SuperCoolingDataProcessing series, this
%fuction takes the time series water temperature data supplied, and divides
%it into Super-Cooling Events.  These events are catelogued as tables for further analysis

%CURRENT DEFINITION OF AN EVENT: An Event starts at 0C as the temperature
%drops below 0 degree C, and ends asthe temperature rises above 0 degree C.
%Thus all datapoints measured will be below 0 C.  Events are
%then filtered based on end of Freeze-Up, start of Break - Up, Minimum
%Required Duration, sensor accuracy, and the absolute negative threshold of
%expected supercooling temperatures
%
```

```
%INPUT
%   dataTable: the timeseries of water temperature data to be analysed
%   sensoAccuracy: The accuracy of the sensor used.  Events that do not
%   record temperatures below this threshold are discarded.
%   timestep: Array of the timesteps in the dataTable (used to account for
%   clock drift)
%   negativeThreshold: If the minimum temperature of an event gets below
%   this value, it is discarded, along with any events within an hour of
%   the event to minimise any effect of the influence that caused the
%   extraneous reading.
%   minDurationMinutes: If the duration of an event in minutes is less than
%   this, it is discarded for being too small
%   endFreezeUp: Date-time of the end of freeze-up
%   startBreak: Date-time of the start of break-up
%   seasonStarts: Date-time array of the start of seasons (fall, winter,
%   spring) to determine what seasons events occur in
%OUTPUT
%   EventTab: Summary of all Super-Cooling Events in the given
%   data set
%   MicroEvents: Number of Events Eliminated from being too short
%   SmallTempEvents: Number of Events Eliminated for having a minimum
%   temperature above the sensor accuracy
%   ExtTemps: Number of Events Eliminated from having minimum temperatures below
%   negative temperature threshold, as well as events that are adjacent to
%   these event by 1 hour or less
%   negCDMSEvents: Number of Events Eliminated for having a Total Degree
%   Minutes of Freezing (cumulative temperature over duration of the event)
%   greater than 0.  A legacy counter from previous definitions, but if
%   events are defined as periods of temperature strictly below 0C, this
%   should not register any values
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%START OF DEFINITION
%i) Find the times of all temperatues below 0 degree C
TOI=dataTable.Time(dataTable.Water_Temperature<=0);
TOIIndex=dataTable.Row_Index(dataTable.Water_Temperature<=0);
eventEnds=cell([0,8]); %Stores start and end rows of all periods of sub zero temperature, the
start and end temperatures, start and end slope for interpolating the 0 crossing, and the
duration and minimum temperature,

%ii) Determine if there are multiple time steps to check due to clock drift
if length(timestep)>1 %Set timechk value to trigger a run through all time steps when required
    timechk=1;
else
    timechk=0;
end

%iii) There is at least one event if TOI is not empty
if ~isempty(TOI)
    if TOI(1) == dataTable.Time(1) %Dataset starts below 0 degree C.  Since we do not know when
this event starts, we remove it from the data set
        del=zeros(0,1);delCount=0; %Store array for the index of POI to be deleted
        index=1;chk=1; %Start at first point of TOI and continue until chk =0
```

```
        while index<=length(TOI) && chk
            delCount=delCount+1;del(delCount,1)=index; %Within this loop, all TOI considered are
part of the partial event
            if timechk %Multiple timesteps to consider
                chk1=1;step=1;partofEvent=0; %Set up WHILE Loop and assume that the TOI is not
part of the event by default
                while chk1 && step<=length(timestep) %Find the correct time step for this
comparison
                    if TOI(index) == TOI(index+1)-timestep(step)% If the current TOI is equal to
the next TOI - a timestep, the next TOI is part of the event
                        partofEvent=1;chk1=0;
                    else
                        step=step+1;
                    end
                end
            else %Only one timestep to check
                partofEvent=TOI(index) == TOI(index+1)-timestep;
            end
            if partofEvent %If the current TOI under consideration is adj to the following TOI,
it is part of the first event
                index=index+1; %Move to the next TOI
            else %The current TOI is the end of the starting partial Event
                chk=0; %Stop the loop
            end
        end
        TOI(del)=[]; %Delete partial event.  All remaining events are assumed to be full events
until proven later in the analysis
        TOIIndex(del)=[];
    end
else %There are no supercooling events in this data series
    return
end

%iv) Determine if any TOI are either just before or after the midwinter
%filter, and remove series of concecutive supercooling temperatures before
%or after the period (partial events to be removed)

%   Set Index of datetimes of mid-winter
midwinterIndex=dataTable.Row_Index(isnan(dataTable.Water_Temperature));
if ~isempty(midwinterIndex)
    startMidWin = midwinterIndex(1);endMidWin = midwinterIndex(end);
else
    startMidWin=0;endMidWin=0; %There is no index for a mid winter season in the time series
end
if ~isempty(midwinterIndex)
    %TOI just before Start of Midwinter
    critTOIrow = find(TOIIndex == startMidWin-1,1);
    if ~isempty(critTOIrow) %There exists a TOI just before midwinter; flag the indicies of all
directly preceding TOIs (partial event)
        prevTOI = TOIIndex(1:critTOIrow);prev1 = prevTOI(1:end-1);prev2=prevTOI(2:end); %Take all
the TOI preceding midwinter, and divide into two sets shifted by 1 row
        TOIcmp = prev2-1 == prev1; %Compare the second set to the first to determine which pairs
are adjacent (part of the same event)
        eventGap = find(TOIcmp == 0); %Determine all cases where the two sets are not adjacent;
```

```
this is a gap between two events
        del1 = ((eventGap(end)+1):critTOIrow); %The index following the event gap up to
critTOIrow is a partial event adjacent to the start of midwinter
    else
        del1=[]; %Empty place holder
    end
    critTOIrow = find(TOIIndex == endMidWin+1,1);
    if ~isempty(critTOIrow) %There exists a TOI just after midwinter; flag the indicies of all
directly following TOIs (partial event)
        nextTOI = TOIIndex(critTOIrow:end);next1 = nextTOI(1:end-1);next2=nextTOI(2:end); %Take
all the TOI following midwinter, and divide into two sets shifted by 1 row
        TOIcmp = next2 == next1+1; %Compare the second set to the first to determine which pairs
are adjacent (part of the same event)
        eventGap = find(TOIcmp == 0); %Determine all cases where the two sets are not adjacent;
this is a gap between two events
        del2 = (critTOIrow:critTOIrow+(eventGap(1)-1)); %The index following the event gap up to
critTOIrow is a partial event adjacent to the start of midwinter
    else
        del2=[]; %Empty place holder
    end
    del = [del1,del2]; %Combine all indicies to be cleared
    if ~isempty(del)
        TOIIndex(del)=[];TOI(del)=[]; %Delete partial events
    end
end

%v) Find series of concecutive numbers (events) from remaining TOIs

%   Set up arrays
TOI1 = TOI(1:end-1);TOI2=TOI(2:end); %The list of TOI and the list of TOI shifted by 1 time step
startTime = zeros([length(TOI), length(timestep)+1]);
endTime = zeros([length(TOI), length(timestep)+1]);

%   Determine Logic values for start and end for all TOIs
for k = 1:length(timestep) %For every value in timestep
    %Start of Event
    chk1 = TOI2 - timestep(k) ~= TOI1; %For those that this statement is true, it will be the
start of a supercooling event with this time step
    startTime(:,k) = [NaN;chk1];
    %End of Event
    chk2 = TOI1~=TOI2-timestep(k); %For those that this statement is true, it will be the end of
a supercooling event with this time step
    endTime(:,k) = [chk2;NaN];
end

%   Event Start
startTime(:,k+1) = floor(sum(startTime(:,1:k),2)/k); %If the statement is false for any timestep,
set to 0.
startTime(1,k+1) = 1; %The first TOI is always the start of an event
%   Convert logic values to TOIIndex values (and then TOI)
index = (1:length(TOI2)+1);index = index'.*startTime(:,end);index(index==0)=[];
startTime = dataTable.Time(TOIIndex(index)); %Start Time
```

```
startTemp = dataTable.Water_Temperature(TOIIndex(index)); %Start Temperature
startSlope = dataTable.Water_Temperature_Slope(TOIIndex(index)-1); %Start Slope (previous slope
due to foreward facing slope calculation)

%   Event End
endTime(:,k+1) = floor(sum(endTime(:,1:k),2)/k); %If the statement is false for any timestep, set
to 0.
endTime(end,k+1) = 1; %The last TOI is always the end of an event
%   Convert logic values to TOIIndex values (and then TOI)
index = (1:length(TOI1)+1);index = index'.*endTime(:,end);index(index==0)=[];
endTime = dataTable.Time(TOIIndex(index)); %End Time
endTemp = dataTable.Water_Temperature(TOIIndex(index)); %Start Temperature
endSlope = dataTable.Water_Temperature_Slope(TOIIndex(index)); %Start Slope (previous slope due
to foreward facing slope calculation)

%vi) Add start and ends to eventEnds
eventEnds((1:length(startTime)),(1:6))=num2cell([datenum(startTime),startTemp,startSlope,datenum(
endTime),endTemp,endSlope]);

%vii) Combine events on either side of a  zero duration
%period (one data point, or end of one event equals the start of the other)
ends = [eventEnds{1:end-1,4}];starts = [eventEnds{2:end,1}];%Ends and Start dataTable time of the
events that are not the end and start of the entire Deployment_Period
roughadjEvents1 = find(ends == starts); %Find end time that are equal to the start indicies of
the next row
roughadjEvents2=roughadjEvents1+1; %The rows with start indicies that match with end indicies
noted in roughadjEvents1
adjEvents=sort([roughadjEvents1;roughadjEvents2]); %List of event rows which are adjacent to each
other (run of concecutive numbers are multiple rows that are adj.)
repeat1=find(adjEvents(1:end-1) == adjEvents(2:end));%Find rows that repeat themselves in
adjEvents;these events are bordered on both sides by minimal duration gaps
repeat2=repeat1+1;repeats=sort([repeat1;repeat2]); %Compile a list of all the repeating indicies
in adjEvents [Index of the repeating events]
adjEvents(repeats)=[];adjEventsStorage=reshape(zeros(size(adjEvents)),[],2); %Delete the events
that are found with repeats; The remaining indices are paired to represent the old rows of
Eventends
adjEventsStorage(:,1)=adjEvents(1:2:end-
1);adjEventsStorage(:,2)=adjEvents(2:2:end);%adjEventsStorage(a,b) states that the data table row
indicies for a distinct supercooling event is [eventEnds(a,1),eventEnds(b,2)]
adjEvents=adjEventsStorage;dif = diff(adjEvents,1,2);removeCount=sum(dif);
removeRows=zeros(removeCount,1); %The difference between any adjacent Events row pair is the
number of rows to be deleted from eventsEnds between those two indicies
currentRow=1; %the current row in the delete index
for k = 1:size(adjEvents,1)
    eventEnds(adjEvents(k,1),2) = eventEnds(adjEvents(k,2),2); %Move to the row of eventEnds
specified by adjEvents(k,1) [start of an event], and change the index in the second column to
    %the value of in the 2nd column of eventEnds in row = adjEvents(k,2) (the end of the last
roughly adjacent event)
```

```
%Add the index for which rows are to be deleted
    if k == 1 %For the first group of rows, it adds to the start of the index counter
        removeRows(1:dif(k)) = (adjEvents(k,1)+1:adjEvents(k,2));
    else %after the first group, the starting point for the next group of rows (current row)
varies with the changing length
        removeRows(currentRow:currentRow+dif(k)-1) = (adjEvents(k,1)+1:adjEvents(k,2));
    end
    currentRow=currentRow+dif(k);
end
eventEnds(removeRows,:)=[]; %Clear out all the events that have been merged

RawEvents=size(eventEnds,1); %Number of Events before filtering
%END OF DEFINITION

%Determine Duration and Minimum Temperature of rough events
for k = 1:RawEvents
    %Duration Hours
    %Interpolate the start and end time the temperature is at 0.
    %Start Time adjustment
    T = eventEnds{k,2};s = eventEnds{k,3}; %The start temperature and slope between the start
temperature and 0
    dt = T/s;eventEnds{k,1} = datetime(datestr(eventEnds{k,1}))-minutes(dt); %Approximate Date-
time of the 0 crossing
    %End Time adjustment
    T = eventEnds{k,5};s = eventEnds{k,6}; %The end temperature and slope between the end
temperature and 0
    dt = -T/s;eventEnds{k,4} = datetime(datestr(eventEnds{k,4}))+minutes(dt); %Date-time of the 0
crossing
    %Calculate duration
    eventEnds{k,7} =hours(eventEnds{k,4} - eventEnds{k,1});
    %Minimum Temperature
    % Row of the data table in both start and end case is found as the
    % smallest row index of date times grater or equal to the date time in
    % question.  This will be "close enough" since the new start and end
    % points will be above the old ones (at 0C)
    startRows =
dataTable.Time>=eventEnds{k,1};index=dataTable.Row_Index(startRows);startRow=min(index);
%Determines which row of the data table is closest to the event start
    endRows =
dataTable.Time>=eventEnds{k,4};index=dataTable.Row_Index(endRows);endRow=min(index); %Determines
which row of the data table is closest to the event end
    temps=dataTable.Water_Temperature(startRow:endRow);eventEnds{k,8} =min(temps); %Determines
minimum temperature
end

%Remove events that could not have an interpolated start or end time (NaT
%value indicate an error with the data)
del1 = find(isnat([eventEnds{1:end,1}]));del2 = find(isnat([eventEnds{1:end,4}]));
eventEnds([del1;del2],:)=[]; %Clear out all the events that have NaT start or end times

%Add the event ends to the data table
Time  = [eventEnds{:,4},eventEnds{:,1}]';z=zeros(size(Time));
tt = timetable(Time,z,z,z,z,z,z,'VariableNames',dataTable.Properties.VariableNames); %Create
storage timetable for the new datapoints
```

```
dataTable.Time = datetime(dataTable.Time,'Format','dd-MMM-yyyy hh:mm:ss'); %Adjust timetable to
make the difference between interpolated points clear

%Add values to table, then adjust the required rows
dataTable = sortrows([dataTable;tt]);row = find(dataTable.Row_Index == 0);
for k = 1:length(row)
    r = row(k);
    dataTable.Row_Index(r) = r;
    dataTable.Serial_Date_Number(r) = datenum(dataTable.Time(r));
    dataTable.Water_Temperature_Slope(r) = dataTable.Water_Temperature_Slope(r-1);
end
%Eliminate events based on duration and too warm peak supercooling
%temperature

%Eliminate Events with a duration less than minDurationMinutes
del=zeros(0,1);delCount=0;
for k = 1:size(eventEnds,1)
    if 60*eventEnds{k,7}<minDurationMinutes %Mark all events that are less than
minDurationMinutes
        delCount=delCount+1;del(delCount)=k;
    end
end
MicroEvents=length(del); %Number of events cleared because they are too short
eventEnds(del,:)=[]; %Delete events that are either too short

%Eliminate Events with a Minimum temperature above the sensor accuracy
del=zeros(0,1);delCount=0;
for k = 1:size(eventEnds,1)
    if eventEnds{k,8}>=-sensorAccuracy %Mark all events with a Minimum temperature above the
sensor accuracy
        delCount=delCount+1;del(delCount)=k;
    end
end
SmallTempEvents=length(del); %Number of events cleared because of a too small minimum temperature
eventEnds(del,:)=[]; %Delete events that have too small of a minimum temperature

%Create Event Table from the remaining events (I)
[EventTab,tab] = createEventTable(dataTable,eventEnds,endFreezeUp,startBreakUp,seasonStarts);

%Additional Analysis and Filterig from Calculated Parameters (II)
[ExtTempsEvents,negCDMSEvents,EventTab] = analyzeCalParams(EventTab,negativeThreshold);

%Flag Events for Manual Screening (III)
[EventTab] = ManualScreenFlag(EventTab);
```

```
%Add Row Names to Event Table after all events have been removed
rownames=cell([1,height(EventTab)]);
if ~isempty(EventTab) %There are events to be labeled
    for k = 1:length(rownames)
        rownames{k}=strcat('SC',num2str(k));
    end
else %No events from this deployment, create a placeholder SC0 (add manual check column to enable
entry to the database
    rownames{1}=strcat('SC','0');EventTab = tab;EventTab.Manual_Check = 0;
    %Set all the numbers to NaN
    EventTab.Duration=NaN;EventTab.Peak_Supercooling=NaN;EventTab.Hours_Between_Events=NaN;

EventTab.Principal_Supercooling_Average_Cooling_Rate=NaN;EventTab.Principal_Supercooling_Duration
=NaN;

EventTab.Principal_Supercooling_Percent_of_Duration=NaN;EventTab.Cumulative_Degree_Minutes_Superc
ooling=NaN;
end
EventTab.Properties.RowNames = rownames;
end
% Create Event Table (I)
function[SuperCoolingTable,tab]=createEventTable(dataTable,eventEnds,endFreezeUp,startBreakUp,met
Seasons)
%Create Table for SupercoolingEvents and an empty table (tab) for use to
%generate a placeholder row if no events are recorded for this deployment

SuperCoolingTable=table('Size',[size(eventEnds,1)
15],'VariableTypes',{'double','double','datetime','datetime','double','double','datetime','dateti
me','double','double',...

'double','double','double','string','string'},'VariableNames',{'Start_Index','End_Index','Start_T
ime','End_Time','Duration','Peak_Supercooling',...

'First_Time_of_Peak_Supercooling','Time_of_Secondary_Peak_Supercooling','Hours_Between_Events','P
rincipal_Supercooling_Average_Cooling_Rate',...

'Principal_Supercooling_Duration','Principal_Supercooling_Percent_of_Duration','Cumulative_Degree
_Minutes_Supercooling','Season','River_Ice_Process'});
SuperCoolingTable.Properties.VariableUnits = {'','','date-time','date-time','hours','-degree
C','date-time','date-time','hours','degree C/minute',...
    'hours','percent','-degree C*minutes','',''};tab =
SuperCoolingTable;tab((2:height(tab)),:)=[];
%Add Start and End Times to table
SuperCoolingTable.Start_Time = [eventEnds{1:end,1}]';SuperCoolingTable.End_Time =
[eventEnds{1:end,4}]';
%Add Peak Supercooling
SuperCoolingTable.Peak_Supercooling=[eventEnds{1:end,8}]';




%Process all parameters that need to be considered by every event
```

```
for k = 1:height(SuperCoolingTable)
    startTime=SuperCoolingTable.Start_Time(k);endTime=SuperCoolingTable.End_Time(k);
    %Index of start, end, and peak supercooling
    startindex = find(dataTable.Time==startTime,1);
    endindex = find(dataTable.Time==endTime,1);
    SuperCoolingTable{k,(1:2)} = [startindex,endindex]; %Add indicies for the anlysis summary
processing (columns are removed before the summary table is added to the database)
    %CDMS
    Timeminutes = minutes(dataTable.Time(startindex:endindex)-dataTable.Time(startindex));
    SuperCoolingTable.Cumulative_Degree_Minutes_Supercooling(k)=-
(trapz(Timeminutes,dataTable.Water_Temperature(startindex:endindex)));
    %Time(s) of Peak Supercooling
    minMatch=find(dataTable.Water_Temperature(startindex:endindex) ==...
        SuperCoolingTable.Peak_Supercooling(k))'; %Determine the Index for the Occurance(s) of
the Minimum Temperature
    for k1 =1:length(minMatch) %Enter Index and Times to array to develop a consistent column
size
        if k1 == 1
            peakindex = minMatch(k1)+startindex-1;
            SuperCoolingTable.First_Time_of_Peak_Supercooling(k)=dataTable.Time(peakindex);
        else
            SuperCoolingTable.Time_of_Secondary_Peak_Supercooling(k,k1-
1)=dataTable.Time(minMatch(k1)+startindex-1);
        end
    end
    %Calculate Hours between events
    if k>1 %Events have occured in the Deployment_Period before this one
        SuperCoolingTable.Hours_Between_Events(k) = hours(startTime -
SuperCoolingTable.End_Time(k-1));
    else %1st event of the Deployment_Period always passes if the preceding cooling rate can be
captured
        SuperCoolingTable.Hours_Between_Events(k) = NaN; %First event of the time series
    end
    %Classify Events by Season
    if startTime>=metSeasons(1) && startTime<metSeasons(2) %Fall Event
        SuperCoolingTable.Season(k) = "Fall";
    elseif startTime>=metSeasons(2) && startTime<metSeasons(3) %Winter Event
        SuperCoolingTable.Season(k) = "Winter";
    else %Spring Event
        SuperCoolingTable.Season(k) = "Spring";
    end
    %Classify Events by Period
    if isnat(endFreezeUp)
        SuperCoolingTable.River_Ice_Process(k) = "Freeze-Up";
    else %Both Periods exist
        if startTime<= endFreezeUp
            SuperCoolingTable.River_Ice_Process(k) = "Freeze-Up";
        elseif startTime>= startBreakUp
            SuperCoolingTable.River_Ice_Process(k) = "Break-Up";
```

```
        else %Event recorded during the period of consolidated/mostly consolidated ice cover
(will only happen if the data is not filtered between freeze-up and break-up
            SuperCoolingTable.River_Ice_Process(k) = "Consolidated Cover";
        end
    end
end

%Duration of Event, Duration of Principal Supercooling, and Principal
%Supercooling Percent of Total Duration
SuperCoolingTable.Duration=hours(SuperCoolingTable.End_Time - SuperCoolingTable.Start_Time);
SuperCoolingTable.Principal_Supercooling_Duration=hours(SuperCoolingTable.First_Time_of_Peak_Supe
rcooling - SuperCoolingTable.Start_Time);
SuperCoolingTable.Principal_Supercooling_Percent_of_Duration=100*SuperCoolingTable.Principal_Supe
rcooling_Duration./SuperCoolingTable.Duration;
SuperCoolingTable.Principal_Supercooling_Average_Cooling_Rate=
SuperCoolingTable.Peak_Supercooling./(60*SuperCoolingTable.Principal_Supercooling_Duration);
%Avg. Cooling Rate is defined as degree C/min
end

%Analysis from Calculated Parameter (II)
function [ExtTempsEvents,negCDMSEvents,SuperCoolingTable] =
analyzeCalParams(SuperCoolingTable,negativeThreshold)
%Eliminate Events with a minimum temperature below the threshold
ExtTempsEvents = 0; %Counter for extreneous events
%Eliminate Events that have a extreneous minimum temperature
extIndex = find(SuperCoolingTable.Peak_Supercooling<negativeThreshold);

%If any events fall within 1 hour of these events, remove them as well
if ~isempty(extIndex)
    extIndex1 = cell([length(extIndex),1]);%placeholder for any adjacent events
    for k1 = 1:length(extIndex)
        row = extIndex(k1); %Row being checked
        if extIndex(k1) == 1 %First event; only check after this event
            extchk =
find(SuperCoolingTable.Start_Time(row+1:end)<=SuperCoolingTable.End_Time(row)+hours(1));%Find any
events directly after an event
        else %Any other event in the Deployment_Period
            extchk1 =
find(SuperCoolingTable.Start_Time(row+1:end)<=SuperCoolingTable.End_Time(row)+hours(1))+row;%Find
any events directly after an event
            etchk2 = find(SuperCoolingTable.End_Time(1:row-1)>=SuperCoolingTable.Start_Time(row)-
hours(1));%Find any events directly before an event
            extchk = [extchk1;etchk2]; %Combine all indicies
        end
        extIndex1{k1} = extchk; %Row indicies of all adjacent events for this index
    end
    %Remove all selected events from table
    rows = unique([cell2mat(extIndex1);extIndex]); %All rows to be removes
    ExtTempsEvents = length(rows); %Updated number of exxtreneous events
    SuperCoolingTable(rows,:)=[];
end
```

```
        %Eliminate Events with a Negative CDMSc value (more time above 0 C than below)
negCDMScindex =
(SuperCoolingTable.Cumulative_Degree_Minutes_Supercooling<=0).*(1:height(SuperCoolingTable))';
negCDMScindex(negCDMScindex==0)=[]; %Create an index of all events with negative CDMSc
negCDMSEvents = length(negCDMScindex); %Number of Negative CDMSc Events
if ~isempty(negCDMScindex)
    SuperCoolingTable(negCDMScindex,:)=[];%Delete Rows
end
end

%Flag Events for Manual Screening (III)
function [SuperCoolingTable] = ManualScreenFlag(SuperCoolingTable)
%Add a column to the table for all flags (Zero is the default for not
%flagged)
SuperCoolingTable.Manual_Check = zeros([height(SuperCoolingTable),1]);
%For each manual check filter, only check the events that have not yet been
%flagged
rows = find(SuperCoolingTable.Manual_Check == 0);
%Manual Check 1: Peak SUpercooling is between -0.1C and Negative Threshold
%(note that all events with TP<negative threshold have already been
%removed)
rows1 = SuperCoolingTable.Peak_Supercooling(rows)<=-0.1;
%If rows1 is not empty, flag the resulting rows
if ~isempty(rows1)
    SuperCoolingTable.Manual_Check(rows1)=1;
end
end
%Remove discarded events
function [T,SupercoolingDatabase]=removeDiscardedEvts(del,ts,T,SupercoolingDatabase)
deploy = strings(size(del)); %Storage of event strings for relabeling
for K = 1:length(del)
    event = ts(del(K)).Event_ID;
    %Find the event in the Event Summary Table and remove it
    row = find(T.Event_ID == event,1);deploy(K) = T.Deployment_ID(row);riv =
T.River(row);site=T.Site(row);
    T(row,:)=[]; %Remove the affected row
    %Find the event in the Time Series structure
    eventList = string({SupercoolingDatabase.Time_Series.Supercooling_Events.Event_ID});
    row = find(eventList == event,1);
    SupercoolingDatabase.Time_Series.Supercooling_Events(row)=[];%Delete time series
    %Deduct the event count from the river summary for the
    %Deployment_Period
    rivs = string({SupercoolingDatabase.Observation_Summaries.Rivers});r = find(rivs==riv,1);
    sites =
string({SupercoolingDatabase.Observation_Summaries(r).Sites.Site});s=find(sites==site,1);
    deploymnt =
string(SupercoolingDatabase.Observation_Summaries(r).Sites(s).Deployments.Properties.RowNames);ro
w = find(deploymnt==deploy{K},1);

SupercoolingDatabase.Observation_Summaries(r).Sites(s).Deployments.Number_Events_Catalogued(row)
=
SupercoolingDatabase.Observation_Summaries(r).Sites(s).Deployments.Number_Events_Catalogued(row)-
1;
```

```
SupercoolingDatabase.Observation_Summaries(r).Sites(s).Deployments.Number_Manually_Screened_Event
s(row) =
SupercoolingDatabase.Observation_Summaries(r).Sites(s).Deployments.Number_Manually_Screened_Event
s(row)+1;
    %Relabel all affected events
    for k = 1:length(deploy)
        rows = find(T.Deployment_ID == deploy(k));
        if ~isempty(rows) %Not all events were cleared out of a Deployment_Period.
            for k1 = 1:length(rows)
                T.Event(rows(k1)) = strcat('SC',num2str(k1));
                T.Event_ID(rows(k1)) =
sprintf('%s_%s',T.Deployment_ID(rows(k1)),T.Event(rows(k1)));
            end
        end
    end
end
end
```

Published with MATLAB® R2021a

```
function [] = AddAlterData()
%AddAlterData Function allows the user to  alter any of the parameters for
%an raw data file that has been logged in CurrentRawFiles.mat.
%If any of the parameters other than the notes are changed for any files,
%the program will reprocess the raw data files with an option to archive the
%current fileList and database before re-processing files.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Author: Sean R. Boyd
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%1) Set Directories
home = pwd;dbFold = strcat(home,'\Databases');lists=strcat(home,'\Raw Data Files');
%2) Load Database and RawFileList
clc;fprintf("Loading Required Files...\n")
SupercoolingDatabase=GetDatabase;rawfileList=GetFileList(1,2);
%3) Create a copy of the 'original' files to archive
oldrawfileList = rawfileList;oldSuperCoolingDatabase=SupercoolingDatabase;
%4) Create a change list (document any changes made that required
%reprocessing the data
changeList = table('Size',[1 4],'VariableType',["string","string","string","string"],...
    'VariableNames',["File_Name","Variable_Changed","Old_Value","New_Value"]);
%5) Create UI Window for RawFileList and prepare for the main menu
fprintf("Generating UI window of currently logged files...\n")
cols = string(rawfileList.Properties.VariableNames);
f=uifigure("Name","Currently Logged Raw Files","IntegerHandle","off");
uit=uitable('Parent',f,'Data',rawfileList,'ColumnName',cols,...
    'RowName',rawfileList.Properties.RowNames, 'Units', 'Normalized','ColumnWidth','fit');clc
    changerows = zeros(height(rawfileList),1); %Storage arrays for the files to be reset and
reprocessed
```

```
%Generate options
o1 = "Change Sensor Accuracy";o2="Change Sensor Offset";o3="Change Negative Threshold";
o4 = "Change Minimum Event Duration";o5 = "Change River Name";o6="Change River ID";o7="Change
Site Name";
o8 = "Change Site ID";o9="Change Deployment Period";o10="Change Fall Start Date";o11="Change
Winter Start Date";
o12="Change Spring Start Date";o13="Change Freeze-Up End Date";o14="Change Break-Up Start Date";
oN="Add/Append Deployment or Review Notes";oT="Hit any other key/Enter to Terminate Program";
%Alter column names so index lines up with inputs
cols([1,2,20])=[];
%Print main option menu
chk=1;
%Assume that the changes required will not need to reset the database. Set to 1 if this is not
the case
reproc = 0;
while chk
    %4) Print Prompt screen
    clc;fprintf("MAIN MENU:\n\nWhich of the following options would you like to do?:\n")
    fprintf("\nChanges that will require re-processing all files:\n   (01)%s\n   (02)%s",o1,o2)
    fprintf("\n   (03)%s\n   (04)%s\n   (05)%s\n   (06)%s\n   (07)%s\n   (08)%s\n
(09)%s",o3,o4,o5,o6,o7,o8,o9)
    fprintf("\n   (10)%s\n   (11)%s\n   (12)%s\n   (13)%s\n   (14)%s\n\n",o10,o11,o12,o13,o14)
    fprintf("Changes that will not require re-processing all files:\n   (N)%s\n\n\n",oN)
    fprintf("(S) to save current changes and end program\n%s\n\n\n",oT)
    inpt = input('Selection:  ','s'); %Collect input as a string
    % Switch case depending on input
    chk1 = 1; %Put in a loop in case the user wants to make related changes (River Name & River
ID for example)

    try
        if ~strcmpi(inpt,'N') && ~strcmpi(inpt,'S')
            id = str2double(inpt);
        elseif strcmpi(inpt,'N')
            id = 0;
        elseif strcmpi(inpt,'S')
            id = -1;
        end
    catch
        id = length(cols)+1;
    end
    while chk1
        if id>=10
           id1=id+1; %Correct for the omission of Deplyment ID corrections
        else
            id1=id;
        end
        switch id
            case 1 %Change Sensor Accuracy
                reproc =
1;[uit,rawfileList,changeList,chk1,changerows]=numdataCorrect(uit,rawfileList,cols,id1,changeList
,changerows);
```

```
            case 2 %Change Sensor Offset
                reproc =
1;[uit,rawfileList,changeList,chk1,changerows]=numdataCorrect(uit,rawfileList,cols,id1,changeList
,changerows);
            case 3 %Change a Negative Threshold
                reproc =
1;[uit,rawfileList,changeList,chk1,changerows]=numdataCorrect(uit,rawfileList,cols,id1,changeList
,changerows);
            case 4 %Change Minimum Event Duration
                reproc =
1;[uit,rawfileList,changeList,chk1,changerows]=numdataCorrect(uit,rawfileList,cols,id1,changeList
,changerows);
            case 5 %Change River Name
                reproc =
1;[uit,rawfileList,changeList,chk1,changerows]=strdataCorrect(uit,rawfileList,cols,id1,changeList
,changerows);
            case 6 %Change River ID
                reproc =
1;[uit,rawfileList,changeList,chk1,changerows]=strdataCorrect(uit,rawfileList,cols,id1,changeList
,changerows);
            case 7 %Change Site Name
                reproc =
1;[uit,rawfileList,changeList,chk1,changerows]=strdataCorrect(uit,rawfileList,cols,id1,changeList
,changerows);
            case 8 %Change Site ID
                reproc =
1;[uit,rawfileList,changeList,chk1,changerows]=strdataCorrect(uit,rawfileList,cols,id1,changeList
,changerows);
            case 9 %Change Deployment Period
                reproc =
1;[uit,rawfileList,changeList,chk1,changerows]=strdataCorrect(uit,rawfileList,cols,id1,changeList
,changerows);
            case 10 %Change Fall Start Date
                reproc =
1;[uit,rawfileList,changeList,chk1,changerows]=datdataCorrect(uit,rawfileList,cols,id1,changeList
,changerows);
            case 11 %Change Winter Start Date
                reproc =
1;[uit,rawfileList,changeList,chk1,changerows]=datdataCorrect(uit,rawfileList,cols,id1,changeList
,changerows);
            case 12 %Change Spring Start Date
                reproc =
1;[uit,rawfileList,changeList,chk1,changerows]=datdataCorrect(uit,rawfileList,cols,id1,changeList
,changerows);
            case 13 %Change End of Freeze-Up
                reproc =
1;[uit,rawfileList,changeList,chk1,changerows]=datdataCorrect(uit,rawfileList,cols,id1,changeList
,changerows);
            case 14 %Change Start of Break-Up
                reproc =
1;[uit,rawfileList,changeList,chk1,changerows]=datdataCorrect(uit,rawfileList,cols,id1,changeList
,changerows);
```

```
            case 0 %Add/Append Deployment or Review Notes

[uit,rawfileList,SupercoolingDatabase,chk1]=notesCorrect(uit,rawfileList,SupercoolingDatabase);
            case -1 %Save changes to database
                chk1=0;chk=0;endChk=0;
            otherwise %end program without saving
                chk1=0;chk=0;endChk=1;
        end
    end

end
%6) Review if the files need to be reprocessed
changerows(changerows==0)=[]; %Clear all zero values from storage array
if reproc && ~endChk %remove specified files from database, and reprocess the files.
    %Before clearing the database, ask if the user wishes to archive the
    %old rawfileList and Database
    clc;str=sprintf('Do you wish to archive the old RawFileList and Database\nprior to re-
processing the files? (Y/N): ');
    chk=1;
    while chk
        inpt=input(str,'s');
        if strcmpi(inpt,'y')
            %RawFileList
            cd(lists)
            %Determine how many other .mat files are in the database
            d=dir('*.mat');files = string({d.name});%This will include number of archived lists +
currentlist, meaning archiving the oldrawfilelist makes the number of archived lists the same
length
            %Generate a string to save the file list as (these are auto
            %generated to keep the file lists and databases 'matched'.
            listName = sprintf("RawFileList - V%d.mat",length(files));
            save(listName,'oldrawfileList')
            %Database
            cd(dbFold)
            %Determine how many other .mat files are in the database
            d=dir('*.mat');files = string({d.name});files =
files(contains(files,'SupercoolingDatabase'));
            %Generate a string to save the file list as (these are auto
            %generated to keep the file lists and databases 'matched'.
            dbName = sprintf("SuperCoolingDatabase - V%d.mat",length(files)); %Same reasoning as
with the rawfilelists
            %Update the Archive Notes
            oldSuperCoolingDatabase.Archive_Notes.Change_List = changeList;
            save(dbName,'oldSuperCoolingDatabase')
        elseif strcmpi(inpt,'n')
            chk=0;
        else
            fprintf('\nInvalid Entry. Please enter (Y)es or (N)o.\n')
        end
    end
```

```
        %Remove the events & time series for the files that are removed (all
    %files registered in changerows
    rawfileList.Processed(changerows) = "No";
    for k = 1:length(changerows)
        fTab = oldrawfileList(changerows(k),:); %Get the corresponding row from before changes
were made
        %Extract the river, site name and deployment ID from the row
        riv = fTab.River;site=fTab.Site;deploy=fTab.Deployment_ID;
        %Go through database and remove the results from this file analysis
        %from the database
        %   1) Observation Summaries
        %Find where the file registered
        if~isempty(SupercoolingDatabase.Observation_Summaries) %If this is empty, there are no
time series processed in the database
            rivers = string({SupercoolingDatabase.Observation_Summaries.Rivers});r = find(rivers
== riv,1); %Find river
            if ~isempty(r) %If the river is not found, the file is not in the data base
                sites = string({SupercoolingDatabase.Observation_Summaries(r).Sites.Site});s =
find(sites == site,1); %Find site
                deployment =
string(SupercoolingDatabase.Observation_Summaries(r).Sites(s).Deployments.Properties.RowNames);d
= find(deployment == deploy,1); %Find deployment
                %Delete deployment from table
                SupercoolingDatabase.Observation_Summaries(r).Sites(s).Deployments(d,:)=[];
                %If this was the only deployment at the site, remove site from site
                %list
                if isempty(SupercoolingDatabase.Observation_Summaries(r).Sites(s).Deployments)
                    SupercoolingDatabase.Observation_Summaries(r).Sites(:,s)=[];
                end
                %If this was the only site on the river, remove river from list
                if isempty(SupercoolingDatabase.Observation_Summaries(r).Sites)
                    SupercoolingDatabase.Observation_Summaries(:,r)=[];
                end
                %   2) Event Table
                %Get event IDs
                evtrows = find(SupercoolingDatabase.Event_Table.Deployment_ID == deploy);
                %Remove events
                SupercoolingDatabase.Event_Table(evtrows,:)=[];
                %   3) Time Series
                %Deployment Timeseries
                deployments =
string({SupercoolingDatabase.Time_Series.Deployments.Deployment_ID});
                r = find(deployments ==
deploy,1);SupercoolingDatabase.Time_Series.Deployments(:,r)=[];
                %Event Time Series (organized like the table)
                SupercoolingDatabase.Time_Series.Supercooling_Events(:,evtrows)=[];
            end
        end
    end
    %Reorder list by River, then by site, then by deployment period
    rawfileList = sortrows(rawfileList,[6,8,10]);
    rawfileList.File_Number = (1:height(rawfileList))';

%Save data base and list changes
```

```
        clc;fprintf('Saving Changes to File...')
    cd(lists)
    save('CurrentRawFileList.mat','rawfileList')
    cd(dbFold)
    save('CurrentSupercoolingDatabase.mat','SupercoolingDatabase')
    cd(home)
elseif ~endChk %Just notes added. If notes and additional changes are made, the new notes will be
added to the database as part of processing.
    %Compare the old rawfilelist to the new rawflieList notes to determine
    %which rows had changes and where.
    oldDepNotes = oldrawfileList.Deployment_Notes;oldRevNotes = oldrawfileList.Review_Notes; %old
(pre changes) notes
    newDepNotes = rawfileList.Deployment_Notes;newRevNotes = rawfileList.Review_Notes; %new notes
    %Determine where changes were made
    depChange = find(oldDepNotes~=newDepNotes);revChange = find(oldRevNotes~=newRevNotes);
    %Distribute any new notes to the appropriate observation summary
    if~isempty(depChange)
        fprintf('\nUpdating Deployment Notes...\n')
        for k = 1:length(depChange)
            %Determine the river site and deployment id of the deployment note
            %to update
            riv =
rawfileList.River(depChange(k));site=rawfileList.Site(depChange(k));deploy=rawfileList.Deployment
_ID(depChange(k));
            %Find the deployment in the observation summary
            allrivs = string({SupercoolingDatabase.Observation_Summaries.Rivers});rivind =
find(allrivs == riv,1);
            allsites =
string({SupercoolingDatabase.Observation_Summaries(rivind).Sites.Site});siteid = find(allsites ==
site,1);
            rows =
string(SupercoolingDatabase.Observation_Summaries(rivind).Sites(siteid).Deployments.Properties.Ro
wNames);row = find(rows == deploy,1);
            %Update the notes

SupercoolingDatabase.Observation_Summaries(rivind).Sites(siteid).Deployments.Deployment_Notes(row
) = newDepNotes(k);
        end
    end
    if~isempty(revChange)
        fprintf('\nUpdating Review Notes...\n')
        for k = 1:length(revChange)
            %Determine the river site and deployment id of the review note
            %to update
            riv =
rawfileList.River(revChange(k));site=rawfileList.Site(revChange(k));deploy=rawfileList.Deployment
_ID(revChange(k));
            %Find the deployment in the observation summary
            allrivs = string({SupercoolingDatabase.Observation_Summaries.Rivers});rivind =
find(allrivs == riv,1);
            allsites =
string({SupercoolingDatabase.Observation_Summaries(rivind).Sites.Site});siteid = find(allsites ==
site,1);
            rows =
```

```
string(SupercoolingDatabase.Observation_Summaries(rivind).Sites(siteid).Deployments.Properties.Ro
wNames);row = find(rows == deploy,1);
            %Update the notes

SupercoolingDatabase.Observation_Summaries(rivind).Sites(siteid).Deployments.Review_Notes(row) =
newRevNotes(k);
        end
    end
    clc;fprintf('Saving Notes to File...')
    cd(lists)
    save('CurrentRawFileList.mat','rawfileList')
    cd(dbFold)
    save('CurrentSupercoolingDatabase.mat','SupercoolingDatabase')
    cd(home)
end
%End program
clc;close(f)
   %Process all files that need to be processed
   if reproc && ~endChk
       fprintf('Starting the reprocessing of updated files...\n\n');pause(3)
    ProcessNewFiles
   end
end

%SUBFUNCTIONS
function
[uit,rawfileList,changeList,chk1,changerows]=numdataCorrect(uit,rawfileList,cols,id,changeList,ch
angerows)
col = cols(id);vars = strrep(col,"_"," ");
clc;fprintf("Changing %s\n\n",vars) %Print out column title
chk=1;
while chk
    str2 = sprintf('Enter the file(s) you wish to add/edit %s for: ',vars);
    rows = input(str2);
    %Check that the rows are valid
    if (~isnumeric(rows)||~isequal(floor(rows),rows)) && ~isempty(find(rows<=0,1)) &&
max(rows)<=max(rawfileList.File_Number) && chk %Not an integer value or is greater than the
number of files
        disp('Invalid entry');
    else
        chk =0;
    end
end
for k = 1:length(rows)
    chk = 1;
    oldChk = 0; %Use as a check to get the original value from the table
    while chk %Request input for the notes until the user hits enter (auto check writing without
a specific confirmation request)
        fTab = rawfileList(rows(k),:); %row on the overall table (shortens variable reference)
        if ~oldChk %Get the original value prior to any alterations (this loop is potentially
repeating
            old =  fTab.(col);oldChk = 1;
        end
        clc;fprintf('File %d of %d: %s\n',k,length(rows),fTab.File_Name)
```

```
        %Print out the River/Site/Season Data
        %current text associated with the file

        fprintf('   Year: %s\n   River: %s (%s)\n   Site: %s (%s)\n   Deployment ID:
%s\n\n',fTab.Deployment_Period,fTab.River,fTab.River_ID,fTab.Site,fTab.Site_ID,fTab.Deployment_ID
)
        fprintf('Current %s: %g\n\n',vars,fTab.(col))

        %Request the new value (with enter/no value used to continue
        %and keep the old value
        s1=sprintf("Enter the new %s (hit Enter to keep current %s): ",vars,vars);
        %Verify the validity of the new value for the given column
        chk2 = 1;
        while chk2
            try
                new = input(s1);
            catch
                fprintf('\nThere was an issue with that input. Please try again\n')
            end
            switch col
                case 'Sensor_Accuracy'
                    if new<=0
                        fprintf('\nThe sensor accuracy must be a positive value. Please try
again\n')
                    else
                        chk2=0;
                    end
                case 'Sensor_Offset'
                case 'Negative_Threshold'
                    if new>=0
                        fprintf('\nThe negative threshold must be a negative value. Please try
again\n')
                    else
                        chk2=0;
                    end
                case 'Minimum_Event_Duration_Minutes'
                    if new<=0
                        fprintf('\nThe minimum event duration must be a positive value. Please
try again\n')
                    else
                        chk2=0;
                    end
            end
        end
        if ~isempty(new)
            fTab.(col)= new;
            uit.Data(rows(k),:) = fTab;
            rawfileList(rows(k),:) = fTab;
```

```
        %Update the changeList
            if height(changeList) == 1 %Newly generated list, no other values entered
                changeList.File_Name(1) = fTab.File_Name;
                changeList.Variable_Changed(1) = vars;
                changeList.Old_Value(1) = string(old);
                changeList.New_Value(1) = string(new);
            else %Add a new row
                changeList.File_Name(end+1) = fTab.File_Name;
                changeList.Variable_Changed(end+1) = vars;
                changeList.Old_Value(end+1) = string(old);
                changeList.New_Value(end+1) = string(new);
            end
        else
            chk=0;
        end
    end
    %Add row to the changelist
    changerows(rows(k))=rows(k);
end
%After all the files are processed, enable the program to go back to the
%main menu
chk1=0;
end

function[uit,rawfileList,changeList,chk1,changerows]=strdataCorrect(uit,rawfileList,cols,id,chang
eList,changerows)
%After the loop is ended go back to main list
chk1=0;
%Set up subloop
col = cols(id);chk=1;updateType=0;
while chk
    vars = strrep(col,"_"," ");skip=0;
    clc;fprintf("Changing %s strings\n\n",vars) %Print out column title

    %Determine if the user is wanting to change all occurances of a string,
    %or a specific file (only check the first time if a multiple changes
    %are required)
    if updateType == 0
        s0 = sprintf('Select Update Method:\n   (1) Update all occurances of a specified string\n
(2) Update specified file(s)\nany other key/Enter to return to Main Menu\n\n>>');
        updateType=input(s0,'s');
        try
            updateType = str2double(updateType); %Convert string to double
            if isnan(updateType)
                updateType=0;
            end
        catch
            updateType=0; %Assume 0 to exit the loop
        end
    else %Rows are already assigned
        skip = 1;
    end
```

```
        switch updateType
        case 1 %Block change - Gett rows from finding the string to change
            clc;
            if ~skip
                s1 = sprintf('Enter the current %s string: %s\n\n',vars);
                oldchk=1;
                while oldchk %Get the value needed to be replaced
                    old = input(s1,'s');
                    rows = find(rawfileList.(col) == old);
                    if isempty(rows)
                        fprintf('Invalid entry. Cannot find %s under %s\n',old,vars)
                    else
                        oldchk=0;
                    end
                end
            end
        case 2 %Individual changes - Adjust specified rows
            clc;
            if ~skip
                chk2=1;
                while chk2
                    str2 = sprintf('Enter the file(s) you wish to edit %s string: ',vars);
                    rows = input(str2);
                    %Check that the rows are valid
                    if (~isnumeric(rows)||~isequal(floor(rows),rows)) &&
~isempty(find(rows<=0,1)) && max(rows)<=max(rawfileList.File_Number) && chk %Not an integer value
or is greater than the number of files
                        disp('Invalid entry');
                    else
                        chk2 =0;
                    end
                end
                %Determine what the old values are
                old = rawfileList.(col)(rows);
            end
        otherwise %Exit subroutine
            chk=0;
    end
    %Request the new value to be inputted
    if chk
    s1=sprintf("Enter the new %s string (hit Enter to keep current %s): ",vars,vars);
    %Verify the validity of the new value for the given column. If
    %there are additional corrections to be made due to changes in
    %strings, make those changes
    try
        new = input(s1,'s');
    catch
        fprintf('\nThere was an issue with that input. Please try again\n')
    end
```

```
    switch col
        case 'River'
            [new,uit,rawfileList,changeList,chk,changerows] =
strUpdate(new,old,uit,rawfileList,col,vars,changeList,rows,changerows,updateType);
            %Check if the user wishes to update River ID
            %(immediately dependent string category)
            inpt = input('Do you wish to also update River ID for these rows? (Y/N): ','s');
            chk3=1;
            while chk3
                if strcmpi(inpt,'Y')
                    col = 'River_ID';chk3=0;chk=1; %Keep in the loop
                elseif strcmpi(inpt,'N')
                    chk3=0;
                else
                    fprintf('Invalid entry.\n\n')
                end
            end
        case 'River_ID'
            [uit,rawfileList,changeList,chk,changerows] =
strUpdate(new,old,uit,rawfileList,col,vars,changeList,rows,changerows,updateType);
            %Check if the user wishes to update River ID
            %(immediately dependent string category)
            inpt = input('Do you wish to also update Site ID & Deployment ID? (Y/N): ','s');
            chk3=1;
            while chk3
                if strcmpi(inpt,'Y')
                    col = 'Site_ID';chk3=0;chk=1; %Keep in the loop
                elseif strcmpi(inpt,'N')
                    chk3=0;
                else
                    fprintf('Invalid entry.\n\n')
                end
            end
        case 'Site'
            [uit,rawfileList,changeList,chk,changerows] =
strUpdate(new,old,uit,rawfileList,col,vars,changeList,rows,changerows,updateType);
            %Check if the user wishes to update River ID
            %(immediately dependent string category)
            inpt = input('Do you wish to also update Site ID & Deployment ID? (Y/N): ','s');
            chk3=1;
            while chk3
                if strcmpi(inpt,'Y')
                    col = 'Site_ID';chk3=0;chk=1; %Keep in the loop
                elseif strcmpi(inpt,'N')
                    chk3=0;
                else
                    fprintf('Invalid entry.\n\n')
                end
            end
```

```
        case 'Site_ID'
            [uit,rawfileList,changeList,chk,changerows] =
strUpdate(new,old,uit,rawfileList,col,vars,changeList,rows,changerows,updateType);
            %Auto-update Deployment ID (the string is synthesized
            %from the Site ID and Deployment Period)
            [uit,rawfileList,changeList] = updateDepID(uit,rawfileList,rows,changeList);
        case 'Deployment_Period'
            [uit,rawfileList,changeList,chk,changerows] =
strUpdate(new,old,uit,rawfileList,col,vars,changeList,rows,changerows,updateType);
            %Auto-update Deployment ID (the string is synthesized
            %from the Site ID and Deployment Period)
            [uit,rawfileList,changeList] = updateDepID(uit,rawfileList,rows,changeList);
        case 'Deployment_ID'
            fprintf('This parameter is auto-generated from Site ID and Deployment Period.\nPlease
check those two parameters (and the other string paramters)\nto determine the source of your
error.\n')
            pause(5);chk=0;
    end
    end
end
end

function
[uit,rawfileList,changeList,chk1,changerows]=datdataCorrect(uit,rawfileList,cols,id,changeList,ch
angerows)
col = cols(id);vars = strrep(col,"_"," ");
clc;fprintf("Changing %s\n\n",vars) %Print out column title
chk=1;
while chk
    str2 = sprintf('Enter the file(s) you wish to add/edit %s for: ',vars);
    rows = input(str2);
    %Check that the rows are valid
    if (~isnumeric(rows)||~isequal(floor(rows),rows)) && ~isempty(find(rows<=0,1)) &&
max(rows)<=max(rawfileList.File_Number) && chk %Not an integer value or is greater than the
number of files
        disp('Invalid entry');
    else
        chk =0;
    end
end
for k = 1:length(rows)
    chk = 1;
    oldChk = 0; %Use as a check to get the original value from the table
    while chk %Request input for the dates until the user hits enter
        fTab = rawfileList(rows(k),:); %row on the overall table (shortens variable reference)
        if ~oldChk %Get the original value prior to any alterations (this loop is potentially
repeating
            old =  fTab.(col);oldChk = 1;
        end
        clc;fprintf('File %d of %d: %s\n',k,length(rows),fTab.File_Name)
```

```
         %Print out the River/Site/Season Data
        %current text associated with the file

        fprintf('   Year: %s\n   River: %s (%s)\n   Site: %s (%s)\n   Deployment ID:
%s\n\n',fTab.Deployment_Period,fTab.River,fTab.River_ID,fTab.Site,fTab.Site_ID,fTab.Deployment_ID
)
        fprintf('Current %s: %s\n\n',vars,datestr(fTab.(col),'mmm dd yyyy HH:MM:ss'))

        %Request the new value (with enter/no value used to continue
        %and keep the old value
        s1=sprintf("Enter the new %s (hit Enter to keep current %s): ",vars,vars);
        %Verify the validity of the new value for the given column
        chk2 = 1;
        while chk2
            try
                new = input(s1,'s');
                if ~isempty(new)
                    new = datetime(datestr(new));
                end
                chk2=0;
            catch
                fprintf('\nThere was an issue with that input. Please try again\n')
            end
        end
        if ~isempty(new)
            fTab.(col)= new;
            uit.Data(rows(k),:) = fTab;
            rawfileList(rows(k),:) = fTab;
        elseif ~isempty(new) && ~isequal(old,new) %The loop is broken and there are changes made.
Update change List
            if height(changeList) == 1 %Newly generated list, no other values entered
                changeList.File_Name(1) = fTab.File_Name;
                changeList.Variable_Changed(1) = vars;
                changeList.Old_Value(1) = string(old);
                changeList.New_Value(1) = string(new);
            else %Add a new row
                changeList.File_Name(end+1) = fTab.File_Name;
                changeList.Variable_Changed(end+1) = vars;
                changeList.Old_Value(end+1) = string(old);
                changeList.New_Value(end+1) = string(new);
            end
        else
            chk=0;
        end
    end
    %Add row to the changelist
    changerows(rows(k))=rows(k);
end
%After all the files are processed, enable the program to go back to the
%main menu
chk1=0;
end

function
```

```
[uit,rawfileList,SupercoolingDatabase,chk1]=notesCorrect(uit,rawfileList,SupercoolingDatabase)
CHK=1;
while CHK
    clc;fprintf("Adding/Appending Notes\n") %Print out column title
    fprintf("\n\nWhich of the following options would you like to do?:\n")
    fprintf("   (01)Add/Append Deployment Notes\n   (02)Add/Append Review Notes\nHit any other
key/Enter to return to the Main Menu\n\n")
    inpt = input('>>','s');
    try
        inpt=str2double(inpt);
    catch
        inpt=0; %Will automatically return to the main menu
    end

    switch inpt %Use to select the note column
        case 1
            noteType = "deployment";col = "Deployment_Notes";colnum = 18;
        case 2
            noteType = "review";col = "Review_Notes";colnum = 19;
        otherwise
            CHK=0;chk1=0;
    end
    %General note writing
    if CHK
        notechk=1;
    else
        notechk=0;
    end
    while notechk
        str1 = sprintf('Do you wish to add/edit any %s notes for any of the files (Y/N)\n[Hit
Enter to return to the previous menu]: ',noteType);
        inpt1 = input(str1,'s');
        if isempty(inpt1) ||strcmpi(inpt1,'n')
            notechk=0;
        elseif strcmpi(inpt1,'y')
            %Request the rows/files that the user wishes to add
            %deployment notes to
            chk=1;
            while chk
                str2 = sprintf('Enter the files you wish to add/edit %s notes for: ',noteType);
                rows = input(str2);
                %Check that the rows are valid
                if (~isnumeric(rows)||~isequal(floor(rows),rows)) && ~isempty(find(rows<=0,1)) &&
max(rows)<=max(rawfileList.File_Number) && chk %Not an integer value or is greater than the
number of files
                    disp('Invalid entry');
                else
                    chk =0;
                end
                %if Rows are valid check that the user selected all the
                %correct rows
                tab = rawfileList(rows,[1,2,7,9,11,colnum]);
                disp(tab)
                inpt = input("Are these the correct files? (Y/y to confirm, any other key/Enter
```

```
to correct): ','s');
                if strcmpi(inpt,'y')
                   chk = 0;
                end
            end
            %Run through the array.
            for k = 1:length(rows)
                chk = 1;
                while chk %Request input for the notes until the user hits enter (auto check
writing without a specific confirmation request)
                    fTab = rawfileList(rows(k),:); %row on the overall table (shortens variable
reference)
                    clc;fprintf('File %d of %d: %s\n',k,length(rows),fTab.File_Name)
                    %Print out the River/Site/Season Data
                    %current text associated with the file
                    str = fTab.(col){:};
                    if ~strcmpi(str,'N/A')
                        notestrold = strings((length(str))/32,1);
                        for ns = 1:length(notestrold)
                            nstr = str((ns-1)*32+1:ns*32); %Line of notes
                            notestrold(ns) = nstr;
                        end
                    else
                        notestrold = "N/A";
                    end
                    notestrold = sprintf('%s\n',notestrold);
                    fprintf('   Year: %s\n   River: %s (%s)\n   Site: %s (%s)\n   Deployment ID:
%s\n\n',fTab.Deployment_Period,fTab.River,fTab.River_ID,fTab.Site,fTab.Site_ID,fTab.Deployment_ID
)
                    fprintf('Current %s Notes:\n\n',noteType);disp(notestrold);fprintf('\n')

                    %Check if the user wishes to overwrite or append
                    fprintf("What do you wish to do with the current %s notes?\n
(01)Overwrite\n   (02)Append\n\nHit any other key/Enter to keep the current %s
notes\n\n",noteType,noteType)
                    inpt = input('>>','s');
                    %Conver input to double
                    try
                        inpt=str2double(inpt);
                    catch
                        inpt=0;
                    end

                    switch inpt
                        case 1
                            notestr='>>';skip=0;apndow = "overwrite";
                        case 2
                            notestr=sprintf('%s\n>>',notestrold);skip=0;apndow = "append";
                        otherwise
                            skip=1;notes=[];
                    end
```

```
        if ~skip
                        %Request the new text (with enter/no text used to continue
                        %and keep the old text
                        s1=sprintf("Enter the new notes you wish to %s the current %s
notes.",apndow,noteType);
                        s2="The text is fitted to a field width of 32 characters.";s3="If you
wish to add a newline, add a ";s4="(newline character).";
                        s5=sprintf("[hit Enter to keep the current %s notes]",noteType);s6 =
"Note that special symbols (such as";
                        s7=char(8320);s8="are complicated to render.";s9="For readability, use
shorthand such as 'deg','min',&'sec' for degree, minutes,";
                        s0="& seconds, respectively, for latitude and longitude.";n='\n';
                        fprintf('%s\n%s\n%s %s
%s\n\n%s\n\n%s)%s\n%s\n%s\n%s\n\n\n',s1,s2,s3,n,s4,s5,s6,s7,s8,s9,s0)
                        notes = input(notestr,'s');
                        if inpt == 2 %Append new notes to old notes
                            notes = strcat(fTab.(col),notes);
                        end
                    end
                    if ~isempty(notes)
                        %Process notes to fit to a 31 width with new lines

                        %1)Find all newline characters in the text and replace
                        %with spaces to finish the fit width line
                        %Determine if newline characters are present
                        nid = strfind(notes,'\n');
                        if ~isempty(nid)
                            nid=nid(1); %Find a newline character
                        end
                        while ~isempty(nid) %There is a newline character in notes (rechecks at
the end of the loop)
                            numspaces = 31-mod(nid,31)+1; %Number of spaces to covert the line of
text the newline character ends into a full width line of text
                            spacestr = strings(1,numspaces);spacestr(1,1:end)=' '; %spaces vector
                            notes1 = notes(1:nid-1);notes2 = notes(nid+2:end);
                            notes=sprintf('%s%s%s',notes1,spacestr,notes2);
                            %Check for the next newline character
                            nid = strfind(notes,'\n');
                            if ~isempty(nid)
                                nid=nid(1); %Find a newline character
                            end
                        end
                        %Add the spaces to the lst line to 'round' the line width
                        numspaces = 31-mod(length(notes),31); %Number of spaces to 'fill' the
line
                        spacestr = strings(1,numspaces);spacestr(1,1:end)=' '; %spaces vector
                        notes=sprintf('%s%s',notes,spacestr);
                        %3)Adjust text for readability. If a word is split between the end of the
first
                        %line and the start of the next, move the word to
                        %the next line
```

```
        %Determine spacing
                        chk1 = 1;linchk=0; %Start loop
                        while chk1
                            spaceid=zeros(size(notes)); %Storage array of when to add spaces
                            skiplinchk=0; %Enable line counter. Stop counting when partial word
is found
                            c = (linchk*31+1);
                            while c <=length(notes) && ~skiplinchk %Start at the line after the
line that has been checked (andy adjustments will not affect previous lines)
                                nstr1 = notes(c:min(c+30,length(notes))); %Get the current line
of text
                                nstr2 = notes(min(c+31,length(notes)):min(c+61,length(notes)));
%Get the next line of text
                                rchk = isequal(nstr1(end),' ') || isequal(nstr2(1),' '); %There
is not a partial word across two lines
                                if ~rchk %Partial word that requires reformatting (will impact
all future lines)
                                    s1 = find(nstr1 == ' ',1,'last');chars1 = nstr1(s1+1:end);
%Find the partial word at the end of current line
                                    spaceid(c+s1:c+s1+length(chars1)-1) =
(c+s1:c+s1+length(chars1)-1);
                                    skiplinchk=1;
                                elseif ~skiplinchk %Signify that the current line has been
checked (meets standards)
                                    linchk=linchk+1;c = c+31;
                                end
                            end
                            %Determine if all lines have been checked
                            numlines = floor(length(notes)/31)+ceil(mod(length(notes),31)/31);
                            if linchk == numlines
                                chk1=0;
                            else %Adjust notes
                                spaceid(spaceid==0)=[]; %Clear out zeroes
                                if ~isempty(spaceid)
                                    %For each space id add a space in the notes. Adjust
                                    %following indicies by one every time
                                    %Determine sets
                                    B = unique([spaceid(1:end-1).' spaceid(2:end).'], 'rows');
                                    S = zeros(size(B)); %Storage array of unique sets
                                    setcount = 1;
                                    for sid = 1:size(B,1) %Go through all pairs
                                        if B(sid,1)~=B(sid,2)-1
                                            S(setcount,1:2) = B(sid,1);%Set of 1 point
                                            setcount=setcount+1; %Move to next set
                                        else %Concecutive set
                                            if isequal(S(setcount,1:2),[0 0]) %Empty set
                                                S(setcount,1:2) = B(sid,1:2); %Add concecutive
set
                                            elseif S(setcount,2) == B(sid,1) %Current concecutive
set continues with the current pair in B;update the end of the run
                                                S(setcount,2) = B(sid,2);
                                            else %Completely different set
                                                setcount=setcount+1;
                                                S(setcount,1:2) = B(sid,1);
```

```
                                        end
                                    end
                                end
                                %Remove all zeros from S. Determine the number of
                                %spaces added
                                S(S(:,1) == 0,:)=[];S(:,3) = S(:,2) - S(:,1) +1;
                                %%For each sets add the required spaces for each set.
                                %%Adjust any following sets but the added number of
                                %%spaces
                                for sid = 1:size(S,1)
                                    spacestr = strings(1,S(sid,3));spacestr(1,1:end)=' ';
%Number of spaces
                                    notes = sprintf('%s%s%s',notes(1:S(sid,1)-
1),spacestr,notes(S(sid,1):end));%Splice in spaces
                                    if sid<size(S,1) %Additional sets
                                        S(sid+1:end,1:2) = S(sid+1:end,1:2)+length(spacestr);
%Adjust numbers by number of spaces
                                    end
                                end
                            end
                        end
                    end
                    %Add spaces to end of notes to have a full 31 width
                    numspaces = 31-mod(length(notes),31); %Number of spaces to 'fill' the
line
                    spacestr = strings(1,numspaces);spacestr(1,1:end)=' '; %spaces vector
                    notes=sprintf('%s%s',notes,spacestr);
                    %Create string vector to store formatted notes
                    notestr = strings(length(notes)/31,1);
                    for ns = 1:length(notestr)
                        nstr = notes((ns-1)*31+1:ns*31); %Line of notes
                        %Add a space to the end of line (makes a 32
                        %width line)
                        nstr(32)=" ";
                        if isequal(nstr(1)," ")%Check if there are left leading spaces to
remove
                            char1 = find(nstr ~=' ',1); %Find the first character in the
string
                            if ~isempty(char1)
                                nstr(1:char1-1)=[];nstr(end+1:end+char1-1) = ' ';%Remove
space and add to the end of the line
                            else %Only spaces - empty line
                                nstr="";
                            end
                        end

                        notestr(ns) = nstr;
                    end
                    %Delete any empty rows and recompile into a single
                    %string
                    notestr(notestr=="")=[];
                    notes=strjoin (reshape(notestr,1,[]),''); %Single string
```

```
        %Add row to table
                        fTab.(col)= notes;
                        uit.Data(rows(k),:) = fTab;
                        rawfileList(rows(k),:) = fTab;
                    else
                        chk=0;
                    end
                end
            end
        else
            disp('Invalid entry. Please enter (Y)es/(N)o or hit enter')
        end
    end
end
end

function [uit,rawfileList,changeList,chk,changerows] =
strUpdate(new,old,uit,rawfileList,col,vars,changeList,rows,changerows,updateType)
if ~isempty(new)
    %Update the rawlist and uit
    uit.Data.(col)(rows) = new;
    rawfileList.(col)(rows) = new;
    chk=0;
    %Update the changeList
    switch updateType
        case 1 %Group change - Only note one row change
            if ismissing(changeList.File_Name(1)) %Newly generated list, no other values entered
                changeList.File_Name(1) = 'Group Change';
                changeList.Variable_Changed(1) = vars;
                changeList.Old_Value(1) = string(old);
                changeList.New_Value(1) = string(new);
            else %Add a new row
                r=height(changeList)+1;
                changeList{r,:} = {'Group Change',vars,string(old),string(new)};
            end
        case 2 %Individual change
        for k = 1:length(rows)
            if ismissing(changeList.File_Name(1)) %Newly generated list, no other values entered
                changeList.File_Name(1) = rawfileList.File_Name(rows(k));
                changeList.Variable_Changed(1) = vars;
                changeList.Old_Value(1) = string(old(k));
                changeList.New_Value(1) = string(new);
            else %Add a new row
                r=height(changeList)+1;
                changeList{r,:} =
{rawfileList.File_Name(rows(k)),vars,string(old(k)),string(new)};
            end
        end
    end
    %Update change rows
    for k = 1:length(rows)
        changerows(rows(k)) = rows(k);
    end
```

```
else
    chk=0;
end
end

function [uit,rawfileList,changeList] = updateDepID(uit,rawfileList,rows,changeList)
for k = 1:length(rows)
    fTab = rawfileList(rows(k),:);
    id = fTab.Site_ID;deploy =
sprintf('%s%s',fTab.Deployment_Period{:}(3:4),fTab.Deployment_Period{:}(8:9));
    old =  fTab.Deployment_ID;new=strcat(id,deploy);
    fTab.Deployment_ID =new;
    uit.Data(rows(k),:) = fTab;
    rawfileList(rows(k),:) = fTab;
    %Update the changeList
    if height(changeList) == 1 %Newly generated list, no other values entered
        changeList.File_Name(1) = fTab.File_Name;
        changeList.Variable_Changed(1) = 'Deployment_ID';
        changeList.Old_Value(1) = string(old);
        changeList.New_Value(1) = string(new);
    else %Add a new row
        r = height(changeList);
        changeList{r+1,:} = {fTab.File_Name,'Deployment_ID',string(old),string(new)};
    end
end
end
```

Published with MATLAB® R2021a

```
function [T]=GetFileList(inpt1,inpt2)
%GetRSKFiles: Loads RSKFileList (list of all rsk files logged in RSKFileList for view.
%INPUTS
% inpt1 - Number for 1 of 4 options
%   1 - All rsk files logged in RSKFileList.mat
%   2 - All unprocessed, but logged rsk files
%   3 - All processed rsk files
%   4 - All un logged rsk files (not in RSKFileList.mat)
%   5 - Output an archived rskfile List (TBD)
% inpt2 - Number 1 or 2, for cases 1-3 for inpt1
%   1 - Output just the file names.
%   2 - Output the full rows of RSKFileList for the relevant files
%
%Note that for inpt1 = 4, only the rsk. file names can be outputted.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Author: Sean R. Boyd January 19th, 2022
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Create storage variable
T=[];
```

```
%Set directories
direct=pwd;fold=strcat(direct,'\Raw Data Files');
cd (fold);clc
%Check number of inputs
narginchk(0,2);
switch nargin
    case 0
        inpt1=[];inpt2=[];
    case 1
        inpt2=[];
end

try
    load('CurrentRawFileList.mat','rawfileList');
    T=rawfileList; %Most likely output
    %Check if user wants all files,new files, or processed files
    fprintf('Select an option to output:\n   (1) All raw files logged\n   (2) All unprocessed raw
files\n   (3) All processed raw files\n   (4) All unlogged raw files\n')
    if isempty(inpt1)
        inpt1=input('[hit Enter terminate the program]: ');fprintf('\n\n')
    else
        if ~isnumeric(inpt1)
            try
                inpt1 = str2double(inpt1);
            catch
                fprintf('\n%s is an invalid value for inpt1',inpt1)
                inpt1=0; %Will end the program
            end
        end
    end
    nofile=0;
catch
    fprintf('RSKFileList.mat cannot be found. All the outputted files are unlogged files.\n')
    inpt1=4;nofile=1;
end

switch inpt1
    case 1 %output all logged files (ask if the user wants just file names or full summary)
        fprintf('Select an option to output:\n   (1) Just raw file names\n   (2) raw file name &
submitted information\n')
        if isempty(inpt2)
            inpt2=input('[hit Enter/any other key terminate the program]: ');
        else
            if ~isnumeric(inpt2)
                try
                    inpt2 = str2double(inpt2);
                catch
                    fprintf('\n%s is an invalid value for inpt2',inpt2)
                    inpt2=0; %Will end the program
                end
            end
        end
```

```
        switch inpt2
            case 1
                T = T.File_Name;
            case 2 %Nothing, default option
            otherwise
                clearvars T; %clear the output variable.
                disp('Program Terminated.');pause(1);clc
        end

        if isempty(T)
            clc;disp('There are no logged files in the current Raw Data folder');pause(1);clc
        end
    case 2 %output all unprocessed files (ask if the user wants just file names or full summary)
        T=T(T.Processed == "No",:);
        fprintf('Select an option to output:\n   (1) Just raw file names\n   (2) raw file name &
submitted information\n')
        if isempty(inpt2)
            inpt2=input('[hit Enter/any other key terminate the program]: ');
        else
            if ~isnumeric(inpt2)
                try
                    inpt2 = str2double(inpt2);
                catch
                    fprintf('\n%s is an invalid value for inpt2',inpt2)
                    inpt2=0; %Will end the program
                end
            end
        end
        switch inpt2
            case 1
                T = T.File_Name;
            case 2 %Nothing, default option
            otherwise
                T=[];
                disp('Program Terminated.');pause(1);clc
        end
        if isempty(T)
            clc;disp('There are no unprocessed files in the current Raw Data
folder');pause(1);clc
        end
    case 3 %output all processed files (ask if the user wants just file names or full summary)
        T=T(T.Processed == "Yes",:);
        fprintf('Select an option to output:\n   (1) Just raw file names\n   (2) raw file name &
submitted information\n')
        if isempty(inpt2)
            inpt2=input('[hit Enter/any other key terminate the program]: ');
        else
            if ~isnumeric(inpt2)
                try
                    inpt2 = str2double(inpt2);
                catch
                    fprintf('\n%s is an invalid value for inpt2',inpt2)
                    inpt2=0; %Will end the program
```

```
                end
            end
        end
        switch inpt2
            case 1
                T = T.File_Name;
            case 2 %Nothing, default option
            otherwise
                T=[];
                disp('Program Terminated.');pause(1);clc
        end
        if isempty(T)
            T=[];
            clc;disp('There are no processed files in the current Raw Data Folder
folder');pause(1);clc
        end
  case 4 %output all unlogged files (can only be file names)
        if ~nofile
            log = T.File_Name;
            d=[dir('*.rsk');dir('*.csv')];all = string({d.name});del=zeros(size(all));
            for k = 1:length(all)
                if find(log == all(k),1)
                    del(k)=k;
                end
            end
            del(del==0)=[];all(del)=[];%Delete all raw files from directory name list that are
not logged
        else
            d=[dir('*.rsk');dir('*.csv')];all = string({d.name});
        end
        if ~isempty(all)
            T=all;
        else
            T=[];
            clc;disp('There are no unlogged raw files in the current RBR raw Files & List
folder.');pause(1);clc
        end
    otherwise
        T=[];
        disp('Program Terminated.');pause(1);clc
end
clc;cd (direct)
end
```

Published with MATLAB® R2021a

```
function [DB]=GetDatabase()
%GetDB: Loads CurrentSupercoolingdatabase from the Database folder
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Author: Sean R. Boyd January 19th, 2022
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
direct=pwd;dbfold=strcat(direct,'\Databases');
cd (dbfold)
load('CurrentSuperCoolingDatabase.mat','SupercoolingDatabase');
  DB=SupercoolingDatabase;
cd (direct)
end
```

Published with MATLAB® R2021a

```
function [summary] = GetObSummary(varargin)
%GetRiverSummaries Function extracts a structure of observation summaries
%from the database based on submitted inputs. There are two methods this
%function uses to extract the timeseries data from the database.
%
% i) Enter the string "IDList" followed by a cell or string of
% Deployment IDs.If a cell/string array
% of potential timeseries IDs is submitted, the function verifies that the
% IDs are correct and then output a structure containing these time series.
%
% ii) The user requests timeseries that fit a certain set of parameters
% using name-value pairs following the same procedure as for the GetTable()
% function, and the final Deployment IDs are used
%
% This function also has the "Print" followed by either 'Y'/'y' or 'N'/'n'
% to indicates if you wish to print the summary to Excel.
% If not submitted, the program will prompt the
%question
%OUTPUT
%summary = Structure of the summary tables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Author: Sean R. Boyd January 19th, 2022
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Determine if print is submitted
summary=[];chk = 1;k = 1;printXSLX=[]; %start loop and placeholder
while chk && k < length(varargin) %Last input myst be y or n. Otherwise no input given
    try
        if strcmpi(varargin{k},'Print')
            printXSLX = string(varargin{k+1});
            if ~strcmpi(printXSLX,'n') && ~strcmpi(printXSLX,'y')
                disp('Invalid entry')
            else %printXLSX is either y/n
                chk=0;
            end
        end
    catch %issues with string compare (invalid input)
        continue
    end
```

```
    %Increase count
    if chk
        k = k+1;
    end
end
if isempty(printXSLX)
    chk = 1;
else
    chk=0;
end
%No print submitted, requyest input
    while chk
        printXSLX = input('Do you wish to save the summary to an Excel file? (Y/N): ','s');
        if ~strcmpi(printXSLX,'n') && ~strcmpi(printXSLX,'y')
            disp('Invalid entry')
        else %printXLSX is either y/n
           chk=0;
        end
    end

%Get file name if printXSLX is 'y'
if strcmpi(printXSLX,'y')
    %Get filename for spreadsheet
    chk1 = 1;
    while chk1
        filename = input(sprintf("Enter filename. It will be saved as a string.\n(default: 'River
Site Summaries')[press Enter for default]: "),'s');
        if isempty(filename)
            filename = 'River Site Summaries.xlsx';
        else
            filename=strcat(filename,'.xlsx');
        end
        fprintf('Filename: %s\n',filename);
        inpt=input('Is this the correct filename?(Y/N): ','s');
        if strcmpi(inpt,'y')
            %end loop
            chk1=0;
        end
    end
    printchk=1;%print to excel at the end of the program
else
    printchk=0; %do not print
end
%Print the loading statement
clc;fprintf('Processing Request...')
%Remove 'Print' and value from submitted varargin prior to submission to
%GetTable
vars = varargin;
if ~isempty(vars)
    vars(k:k+1)=[];
end
```

```
%Get Event Table and extract deployment ids
if isempty(vars)
    T = GetTable();
else
    T = GetTable(vars{:});
end
if ~isempty(T)
    IDs = unique(T.Deployment_ID);%All unique submitted IDs
else
    fprintf('No Deployments have been found that contain events that match the submitted
deployments\n');return
end

%If no issues arise from getting the IDs, proceed with the rest of the
%function

%Get directories
home=pwd;results = strcat(pwd,'/Generated Results');
%1) Get the Database and the full event catalogue
db = GetDatabase();
%2) Default is that the summary is the initial structure
summary = db.Observation_Summaries;
%If there are submitted deployment IDs, verify the input and then modify summary
%to only contain the submitted deployment IDs
if ~isempty(IDs)
    %Check that the list of IDs submitted is valid
    allDep = string({db.Time_Series.Deployments.Deployment_ID}); %All deployment ID
    allEvts = string({db.Time_Series.Supercooling_Events.Event_ID}); %All Event ID
    allIDs = [allDep,allEvts];
    IDs=unique(IDs);
    evts = db.Event_Table; %Full event table for reference
    err = strings(size(IDs));del = zeros(size(IDs));
    for k = 1:length(IDs)
        if isempty(find(allIDs == IDs(k),1))
            err(k) = IDs(k);del(k)=k;
        end
    end
    err(err=="")=[]; %Clear all empty elements
    del(del==0)=[];IDs(del)=[]; %Clear all invalid elements
    if ~isempty(err)
        if isequal(err,IDs)
            error('None of the deploymentIDs submitted were valid.')
        else
            warning('Some of the deploymentIDs submitted were invalid.')
        end
    end
    %Create a reference table of the river and sites for each deployment ID
    T = table('Size',[length(IDs),
3],'VariableTypes',["string","string","string"],'VariableNames',["River","Site","Deployment_ID"])
;
```

```
    for k = 1:length(IDs)
        T.Deployment_ID(k) = IDs(k);
        r=find(evts.Deployment_ID == IDs(k),1); %Get a row for reference
        if isempty(r) %ID woud be an event ID
            r=find(evts.Event_ID == IDs(k),1);
            T.Deployment_ID(k) = evts.Deployment_ID(r);
        end
        T.River(k) = evts.River(r);T.Site(k) = evts.Site(r);
    end
    %Remove the rivers that are not in the reference table from the summary
    %I) River level - clear out any rivers not in T
    allrivs = string({summary.Rivers});del = zeros(size(allrivs));
    for k = 1:length(allrivs)
        if isempty(find(T.River == allrivs(k),1))
            del(k)=k;
        end
    end
del(del==0)=[];summary(:,del)=[];
    %II) Site Level for each remaining river(s)
    for k = 1:length(string({summary.Rivers}))
        allsites = string({summary(k).Sites.Site});del = zeros(size(allsites));
        for k1 = 1:length(allsites)
            if isempty(find(T.Site == allsites(k1),1))
                del(k1)=k1;
            end
        end
        del(del==0)=[];summary(k).Sites(:,del)=[];
    end
    %III) Deployment Level for all remaining sites
    for k = 1:length(string({summary.Rivers}))
        for k1 = 1:length(string({summary(k).Sites.Site}))
            alldeploys = string(summary(k).Sites(k1).Deployments.Properties.RowNames);
            del = zeros(size(alldeploys));
            for k2 = 1:length(alldeploys)
                if isempty(find(T.Deployment_ID == alldeploys(k2),1))
                    del(k2)=k2;
                end
            end
            del(del==0)=[];summary(k).Sites(k1).Deployments(del,:)=[];
        end
    end
end
%Determine the relevant totals for summing events
%Turn off warning to so that MATLAB adds default values without setting off
%a warning
warning('off','all')
rivs = string({summary.Rivers});
rivTabs = cell(1,length(rivs)); %storage array of river summary tables
for r = 1:length(rivs)
    sites = string({summary(r).Sites.Site});
    for s = 1:length(sites)
        sumT = summary(r).Sites(s).Deployments;
        tot = cell([1,width(sumT)]); %Storage array
```

```
          for c = 1:length(tot)
             if c>=12 && c<=18 %In the columns that need to be totalled
                 if height(sumT)>1 %There are multiple deployments at the site, create a site
subtable
                     tot{c} = sum(sumT{:,c}); %Array of totals
                 else
                     tot{c} = sumT{:,c};
                 end
             else %Not applicible, put a place holder marker
                 unit =sumT{:,c};
                 if isequal(class(unit),'cell')
                     unit=unit{:};
                 end
                 if isequal(class(unit),'string') || isequal(class(unit),'char')
                     tot{c} = "N/A";
                 elseif isequal(class(unit),'double')
                     tot{c} = NaN();
                 else %date-time
                     tot{c} = NaT();
                 end
             end
         end
         if height(sumT)>1
             h = height(sumT)+1;
             %Add a row to the table named total
             for c = 1:length(tot)
                 sumT{h,c} = tot{c};
             end
             sumT.Properties.RowNames(end) = {'EVENT_TOTALS'};
         end
         %Put current summary Tabe back into the structure
         summary(r).Sites(s).Deployments = sumT;

         %If this is the first site in the river, set r1 to this site last
         %row's totals. Else, add the last row (either the totals row or the
         %sole season's numbers) to the ongoing table
         if s == 1
             r1 = [tot{12:18}];
         else
             r1 = r1+[tot{12:18}];
         end
         %If there are multiple sites, collect the river totals at the end of this section
         if length(sites)>1
             cols=string([sumT.Properties.VariableNames]);
             rivTab =
table('Size',[1,7],'VariableTypes',["double","double","double","double","double","double","double
"],'VariableNames',cols(12:18));
             rivTab{:,:}=r1;rivTabs{r}=rivTab;%Add the event totals to the table,then put it in
the storage array
         end
     end
end
```

```
%If there are multiple rivers, summarize the full set
if length(rivs)>1
    if isempty(IDs) || isequal(allDep,sort(IDs)) %All Deployments considered
        dbLab = 'Database Overview';
    else
        dbLab = 'Multi-River Set';
    end
    dbTab =
table('Size',[1,7],'VariableTypes',["double","double","double","double","double","double","double
"],'VariableNames',cols(12:18));
    vals = rivTabs{1};
    vals = vals{:,:};
    for k= 2:length(rivTabs)
        vals1 = rivTabs{k};
        vals1 = vals1{:,:};
        vals = vals+vals1;
    end
    dbTab{:,:}=vals;
end

%Request if the user wishes to print summary to an excel file
if printchk
    %Print summary to spreadsheet
    cd(results)
    if length(rivs)>1
        %Write Database summary
        dblabelstartCell='A1';dbtabstartCell='B2';
        writecell({dbLab},filename,'Sheet','Database','Range',dblabelstartCell);

writetable(dbTab,filename,'Sheet','Database','Range',dbtabstartCell,'WriteVariableNames',true,'Wr
iteRowNames',true);
        dbrow = 5;dbtabStartCell = strcat('B',num2str(dbrow));dblabelstartCell =
strcat('A',num2str(dbrow-1));
    end
    %Write each river's summary
    for r = 1:length(rivs)
        sites = string({summary(r).Sites.Site});labelstartCell='A1';tabstartCell
='B2';row=2;%Starting cell and row for the first table
        for s = 1:length(sites)
            writecell({sites(s)},filename,'Sheet',rivs(r),'Range',labelstartCell)
            sumT = summary(r).Sites(s).Deployments;

writetable(sumT,filename,'Sheet',rivs(r),'Range',tabstartCell,'WriteVariableNames',true,'WriteRow
Names',true)
            %Determine the starting cell of the next table
            row = row+height(sumT)+3;tabstartCell = strcat('B',num2str(row));labelstartCell =
strcat('A',num2str(row-1));
        end
```

```
%Add the river summary at the end
        if length(sites)>1
            writecell({sprintf('%s
Summary',rivs(r))},filename,'Sheet',rivs(r),'Range',labelstartCell)
writetable(rivTabs{r},filename,'Sheet',rivs(r),'Range',tabstartCell,'WriteVariableNames',true,'Wr
iteRowNames',true)
        end
        %Add river summary to the database summary page
        if length(rivs)>1
            writecell({sprintf('%s
Summary',rivs(r))},filename,'Sheet','Database','Range',dblabelstartCell);

writetable(rivTabs{r},filename,'Sheet','Database','Range',dbtabStartCell,'WriteVariableNames',tru
e,'WriteRowNames',true)
            dbrow = dbrow+3;dbtabStartCell = strcat('B',num2str(dbrow));dblabelstartCell =
strcat('A',num2str(dbrow-1));
        end
    end
end
%Turn warning back on
cd(home)
warning('on','all')
%Complete loading statement
fprintf('Complete\n')
if printchk
    fprintf('%s is saved in Generated Results\n\n',filename)
else
    fprintf('\n\n')
end
end
```

Published with MATLAB® R2021a

```
function [s] = GetScreenedPeriods(outputType)
%GetScreenedPeriods Loads the records of manually screened events
%INPUT
%   outputType = Enter 'Summary' or 'TimeSeries' if you only wish the
%   summary table or Time Series outputted. Otherwise the function outputs
%   the whole structure.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Author: Sean R. Boyd January 19th, 2022
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
narginchk(0,1)
home = pwd;
%Get Database
db = GetDatabase();
screenedPeriods = db.Screened_Periods;
```

```
%Use input to decide on output
if nargin == 0 %Output everything
    outputType = "both";
elseif nargin == 1 && (~isequal(class(outputType),'string') &&
~isequal(class(outputType),'char')) %The input cannot be processed as either 'Summary' or
'TimeSeries'
    outputType = "both";
end
switch outputType
    case 'Summary'
        s = screenedPeriods.Summary;
    case 'TimeSeries'
        s = screenedPeriods.TimeSeries;
    otherwise
        s = screenedPeriods;
end
cd(home)
end
```

Published with MATLAB® R2021a

```
function [T] = GetTable(varargin)
%GetTable: Retrieves the current SupercoolingDatabase Event Table and then
%   filters for only events that match the submitted argument.
%
%NOTES ON POSSIBLE FILTERS
%
%   I) Qualatative Filter - Qualatative data is data stored as string
%   arrays that can only be filtered as a direct match.
%
%   Ex. T = GetTable("River_ID","KR") would output all the events with the
%   River ID of 'KR', while T = GetTable("River_ID",["KR","NSR"]) will
%   output all the events that have either of these River IDs.
%
%   II) Quantatative Filter - Quantatative data is data stored as double or
%   date-time data types can be filtered through various logical arguments.
%   In both cases, the logical argument filter is the same.
%
%       i) Double: The valid variable name for a double type variable is
%       followed by a string containing a logical operation and then the
%       threshold value.
%
%       Ex. T = GetTable("Duration","<",10) will output all the events with
%       a duration less than 10 hours (keep the units of the variable in
%       mind)
%
%       ii) Date-time: The valid variable name for a datetime  type
%       variable is either followed by (a) a logical operation and a
%       specific datetime, or (b) a string array indicating the component
%       of the datetime object under consideration, the logical operation,
%       and the threshold value.
%
```

```
%       (a) Ex. T = GetTable("Start_Time","<",Dec 01, 2015 10:00:00)
%       will get all events that started prior to 10AM on December 1st,
%       2015.
%
%       (b) Ex. T = GetTable("Start_Time","month,"<=","mar")
%       will get all events that started in January-March (inclusive) of
%       any given year. List of valid string arrays are:
%
%       'year' - year of datetime
%       'month' - month of datetime
%       'doy' - day of the year of datetime
%       'hour' - hour of day of datetime
%
%LIST OF VALID LOGICAL ARGUMENT INPUTS
%For all quantifiable data types, the following character strings indicate
%the noted logical operation.
%
% '=='       Equal To.
%
% '~='       Not Equal To.
%
% '<'        Strictly Less Than.
%
% '>'        Strictly Greater Than.
%
% '<='       Less Than or Equal To.
%
% '>='       Greater Than or Equal To.
%
% '><'       Strictly Greater Than the first value,
%            Strictly Smaller Than the second value.
%
% '>=<'      Greater Than Or Equal to first value,
%            Strictly Smaller Than the second value.
%
% '><='      Strictly Greater Than first value,
%            Smaller Than Or Equal to the second value.
%
% '>=<='     Greater Than Or Equal to the first value,
%            and Smaller Than Or Equal to the second value.
%
% Note that the last 4 options enable comparison between two values. Thus
% the threshold needs to be a double array containing two values
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Author: Sean R. Boyd January 19th, 2022
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Create array of valid logical arguments
validLogicInpt = ["==","~=","<",">","<=",">=","><",">=<","><=",">=<="];
%Get the  SupercoolingDatabase Event Catalogue
Database = GetDatabase();
T=Database.Event_Table;
```

```
if ~isempty(varargin) %The table needs to be filtered. varargin is a cell array

    %Determine the cases each variable name falls into (avoid needing to update
    %this code with changes in variables
    vars = string(T.Properties.VariableNames);
    vartype = strings(size(vars));
    for k = 1:length(vars)
        vartype(k) = class(T.(vars(k)));
    end
    %Search through varargin and determine which variables are going to be used
    %as a filter
    headers = varargin; %Fist store all inputs as possible headers

    %If "IDList" is entered, replace with both "Deployment_ID" and
    %"Event_ID" and duplicate lists to check
    chk=1;k=1;i=0;
    while chk && k<=length(headers)
        if isequal(size(headers{k}),[1,1]) && isa(headers{k},'string')%They can be compared
            i = strcmpi(headers{k},"IDList");chk=0;
        else
            k = k+1;
        end
    end
    if i %IDList is an entry, copy the following list and add two copies of it to the header
line.
        if k<length(varargin)
            list = headers{k+1};
            elist=list(contains(list,'_SC'));%List of Event ID
            dlist = list(~contains(list,'_SC'));%List of Deployment
            %Add "Deployment_ID" and "Event_ID" to header followed by the
            %list
            headers(end+1:end+4) = {"Deployment_ID",dlist,"Event_ID",elist};
            varargin(end+1:end+4) = {"Deployment_ID",dlist,"Event_ID",elist};
        else
            error('IDList must be followed by a list of IDs')
        end
    end
    del = 1:length(headers); %Array for the elements to be removed
    index = 1:length(headers); %Index of elements in varargin
    type = cell(size(headers)); %Storage array of data type
    varsindex = zeros(size(headers)); %Storage array of index of valid headers in vars
    for k = 1:length(headers)
        if isequal(size(headers{k}),[1,1]) && isa(headers{k},'string')%They can be compared
            i = find(vars == headers{k},1);
        else
            i=[];
        end
        if ~isempty(i)
            del(k)=0;type{k} = vartype(i);varsindex(k)=i;
        end
    end
    %Remove zero values and all invalid headers
  del(del==0)=[];index(index==0)=[];type(del)=[];index(del)=[];headers(del)=[];varsindex(del)=[];
```

```
%Create a storage cell array for the valid headers, the data type,logical
    %argument(s), and threshold values
    c = cell([length(headers),6]);
    c(:,1)=num2cell(varsindex);c(:,2) = headers';c(:,3)=type';
    %Going through each row in the cell, determine by type how much of the
    %following varargins need to be considered. Index can be used as an
    %additional limiter
    for k = 1:length(type)
        switch type{k}
            case "string"
                c(k,(4:5)) = {'N/A','N/A'}; %No logical arguments needed
                c(k,6) = varargin(index(k)+1); %Threshold values
            case "double"
                c(k,4) = varargin(index(k)+1); %One logical argument needed
                c(k,5) = {'N/A'};
                c(k,6) = varargin(index(k)+2);%Threshold values
            case "datetime"
                c(k,4) = varargin(index(k)+1); %First logical arguments needed
                if index(k) == 1
                    if find(validLogicInpt == varargin{2},1)
                        c(k,5) = {'N/A'};
                        c(k,6) = varargin(3);%Threshold values
                    else
                        c(k,5) = varargin(3);
                        c(k,6) = varargin(4);%Threshold values
                    end
                elseif index(k+1) ~= index(k)+1
                    c(k,5) = varargin(index(k)+2);
                    c(k,6) = varargin(index(k)+3);%Threshold values
                else
                    c(k,5) = {'N/A'};
                    c(k,6) = varargin(index(k)+1);%Threshold values
                end
        end
    end
    %Sort rows based on first column
    c=sortrows(c,1);
    %Check that the threshold values are not empty
    del=zeros(size(c,1),1);
    for k = 1:size(c,1)
        if isempty(c{k,6})
            del(k)=k;
        end
    end
    del(del==0)=[];c(del,:)=[];
    if~isempty(c)
        %Filter the table based on this ordering
        %Use while loop to check if the table has
        %been reduced to an empty output
        chk=1;k=1;
```

```
        while k<=size(c,1) && chk
            %Extract required information from the cell array
            col = c{k,2}; %Column name
            type = c{k,3}; %data type
            L1 = c{k,4}; %logic argument 1
            L2 = c{k,5}; %Logic argument 2
            val = c{k,6};%valid arrays;

            %Use data type as a switch

            switch type
                case 'string'
                    [chk,alldat] = emptyChk(T,col); %Check that the table is not empty
                    if chk
                        %Check that val is a string array
                        if isstring(val) || ischar(val)
                            valid=1;
                        else
                            error('%s requies a string array to filter the table',col)
                        end
                        %If val is a string array (possibly valid) use it to filter
                        %out any no compliant rows in the table
                        if valid
                            keep = zeros([length(alldat),length(val)+1]); %matrix to hold logic
array
                            for k1 = 1:length(val)
                                keep(:,k1) = alldat == val(k1);
                            end
                            keep(:,k1+1) = sum(keep,2); %Sum up the coluumn, with positive
indicating a match
                            del = find(keep(:,k1+1) == 0); %Array of rows to delete
                        end
                    end
                    logchk=0; %No need to apply a variable logic check
                case 'double'
                    [chk,alldat] = emptyChk(T,col); %Check that the table is not empty
                    if chk
                        %Check that the inputted array is double and in ascending
                        %order (and is no more than a length 2 vector
                        if isa(val,'double')
                            if isequal(size(val),[1,1]) || isequal(size(val),[1,2]) ||
isequal(size(val),[2,1])
                                if length(val) == 2
                                    if val(1)>val(2)
                                        val = [val(2),val(1)];
                                        fprintf('Rearranged the submitted array for %s\n',col)
                                    end
                                    if ~isempty(find(isnan(val),1))
                                        error('NaN values can only be used when considering a
parameter == NaT or ~= NaN')
                                    end
                                end
                                valid=1;
```

```
                            else
                                error('%s requires either a single double value or 2x1 double
vector to filter events',col)
                            end
                        else
                            error('%s requires either a single double value or 2x1 double vector
to filter events',col)
                        end
                        if valid
                            logchk =1; %Applay a variable logic check
                            data=alldat;
                        end
                    end
                case 'datetime'
                    [chk,alldat] = emptyChk(T,col); %Check that the table is not empty
                    if chk
                        if isequal(L2,'N/A') %inputted date time for comparison
                            %Check that a valid date time has been entered
                            if isdatetime(val) && (isequal(size(val),[1,1]) ||
isequal(size(val),[1,2]) || isequal(size(val),[2,1]))
                                if length(val) == 2
                                    if val(1)>val(2)
                                        val = [val(2),val(1)];
                                        fprintf('Rearranged the submitted array for %s\n',col)
                                    end
                                    if ~isempty(find(isnat(val),1))
                                        error('NaT values can only be used when considering a
parameter == NaT or ~= NaT')
                                    end
                                end
                                valid = 1;data=alldat;
                            else
                                error('%s requires either a single datetime value or 2x1 datetime
vector to filter events',col)
                            end

                        else %compare a component of the date-time value
                            %Convert the data to the the required format
                            switch L1
                                case 'year'
                                    data = year(alldat);valid=1;
                                case 'month'
                                    if isa(val,'string') %val needs to be converted to month
number
                                        %Generate a list of short and long
                                        %format string values to compare to val
                                        shortf = ["Jan","Feb","Mar","Apr","May","Jun","Jul",...
                                            "Aug","Sep","Oct","Nov","Dec"];
                                        longf = ["January","February","March","April",...
                                            "May","June","July","August","September",...
                                            "October","November","December"];
                                        num = zeros(size(val));
```

```
                                    for k1 = 1:length(val)
                                        datchk=1;k2=1;
                                        while k2<=12 && datchk
                                            if strcmpi(shortf(k2),val(k1))
                                                num(k1)=k2;datchk=0;
                                            elseif strcmpi(longf(k2),val(k1))
                                                num(k1)=k2;datchk=0;
                                            else
                                                k2=k2+1;
                                            end
                                        end
                                    end
                                    val=sort(num);
                                end
                                data = month(alldat);valid=1;
                            case 'doy'
                                data=day(alldat,'doy');valid=1;
                            case 'hour'
                                data=hour(alldat);valid=1;
                            otherwise
                                error("%s is not a valid entery for filtering date-time data.
Please enter 'year', 'month','doy', or 'hour' to filter the Event Table.",L1)
                        end
                        type = 'double';
                    end
                    if valid
                        logchk =1; %Applay a variable logic check
                    end
                end
            end
            if logchk && chk
                if isequal(L2,'N/A')
                    logarg = L1; %logical argument is in L1
                else
                    logarg = L2; %logical argument is in L2
                end
                logindex = find(validLogicInpt == logarg);
                if isempty(logindex)
                    error('%s is an invalid logical argument input. Please see documentation for
the valid logical argument inputs.',logarg)
                end
                %Error check
                [logindex1] = errorCheck(val,type,logindex);
                switch logindex1
                    case 1 %Keep all data equal to the threshold threshold
                        if isequal(type,'double')
                            if ~isempty(find(isnan(val),1)) %The code is searching for NaN values
                                keep = isnan(data);
                            else
                                keep = data == val;
                            end
                        elseif isequal(type,'datetime')
```

```
                        if ~isempty(find(isnat(val),1)) %The code is searching for NaT values
                            keep = isnat(data);
                        else
                            keep = data == val;
                        end
                    end
                case 2 %Keep all data not equal to the threshold
                    if isequal(type,'double') %The code is searching for NaN values
                        if ~isempty(find(isnan(val),1)) %The code is searching for NaN values
                            keep = ~isnan(data);
                        else
                            keep = data == val;
                        end
                    elseif isequal(type,'datetime')
                        if ~isempty(find(isnat(val),1)) %The code is searching for NaT values
                            keep = ~isnat(data);
                        else
                            keep = data == val;
                        end
                    end
                case 3 %Keep all data strictly less than the threshold
                    keep=data<val;
                case 4 %Keep all data strictly greater than the threshold
                    keep=data>val;
                case 5 %Keep all data less than or equal to the threshold
                    keep=data<=val;
                case 6 %Keep all data greater than or equal to the threshold
                    keep=data>=val;
                case 7 %Keep all data strictly greater than the lower threshold or strictly
lower than the upper threshold
                    keep1 = data>val(1);keep2=data<val(2);keep = and(keep1,keep2);
                case 8 %Keep all data greater than or equal to the lower threshold or
strictly lower than the upper threshold
                    keep1 = data>=val(1);keep2=data<val(2);keep = and(keep1,keep2);
                case 9 %Keep all data greater than the lower threshold or lower than or equal
to the upper threshold
                    keep1 = data>val(1);keep2=data<=val(2);keep = and(keep1,keep2);
                case 10 %Keep all data greater than or equal to the lower threshold or less
than or equal to the upper threshold
                    keep1 = data>=val(1);keep2=data<=val(2);keep = and(keep1,keep2);
                otherwise
                    error('%s is not a valid logical argument input. See documentation for a
list of valdid logical argument formatting',logarg)
            end
            if size(keep,2)>1 %There are multiple subcolumns in this keep array
                keep = min(sum(keep,2),1); %If there is one value to keep in the row, keep
the whole row
            end
            del = find(keep==0);
        end
        if chk     %Delete all flagged rows if there anything to clear
            if ~isempty(del)
                T(del,:)=[];
            end
```

```
                k=k+1;
            end
        end
    else
        T=[];fprintf('\nThreshold values were empty.\n')
        fprintf('\nAlways double check program inputs because errors such as\nmistyped column
names would be ignored by the program.\n')
    end
end
end

%SUBFUNCTIONS
function [chk,alldat] = emptyChk(T,col)
alldat = T.(col);
if isempty(alldat)
    chk=0;
else
    chk=1;
end
end

function [logindex] = errorCheck(val,type,logindex)
%Conducts an error check. If no errors arise. Continue processing input
            switch logindex
                case 1 %Keep all data equal to the threshold threshold
                    if length(val)>1
                        error('Only a single value can be used for an == logical argument')
                    end
                case 2 %Keep all data not equal to the threshold
                    if length(val)>1
                        error('Only a single value can be used for a ~= logical argument')
                    end
                case 3 %Keep all data strictly less than the threshold
                    if length(val)>1
                        error('Only a single value can be used for a < logical argument')
                    end
                    if isequal(type,'double') %The code is searching for NaN values
                        if ~isempty(find(isnan(val),1))
                            error('NaN values cannot be used for a < argument')
                        end
                    elseif isequal(type,'datetime')
                        if ~isempty(find(isnat(val),1)) %The code is searching for NaT values
                            error('NaT values cannot be used for a < argument')
                        end
                    end
                case 4 %Keep all data strictly greater than the threshold
                    if length(val)>1
                        error('Only a single value can be used for a > logical argument')
                    end
                    if isequal(type,'double')
                        if ~isempty(find(isnan(val),1)) %The code is searching for NaN values
                            error('NaN values cannot be used for a > argument')
                        end
```

```
                    elseif isequal(type,'datetime')
                        if ~isempty(find(isnat(val),1)) %The code is searching for NaT values
                            error('NaT values cannot be used for a > argument')
                        end
                    end
                case 5 %Keep all data less than or equal to the threshold
                    if length(val)>1
                        error('Only a single value can be used for a <= logical argument')
                    end
                    if isequal(type,'double')
                        if ~isempty(find(isnan(val),1)) %The code is searching for NaN values
                            error('NaN values cannot be used for a <= argument')
                        end
                    elseif isequal(type,'datetime')
                        if ~isempty(find(isnat(val),1)) %The code is searching for NaT values
                            error('NaT values cannot be used for a <= argument')
                        end
                    end
                 case 6 %Keep all data greater than or equal to the threshold
                    if length(val)>1
                        error('Only a single value can be used for a >= logical argument')
                    end
                    if isequal(type,'double')
                        if ~isempty(find(isnan(val),1)) %The code is searching for NaN values
                            error('NaN values cannot be used for a >= argument')
                        end
                    elseif isequal(type,'datetime')
                        if ~isempty(find(isnat(val),1)) %The code is searching for NaT values
                            error('NaT values cannot be used for a >= argument')
                        end
                    end
                case 7 %Keep all data strictly greater than the lower threshold or strictly lower
than the upper threshold
                    if length(val)==1
                        error('Two values are needed for a >< logical argument')
                    end
                    if isequal(type,'double')
                        if ~isempty(find(isnan(val),1)) %The code is searching for NaN values
                            error('NaN values cannot be used for a >< argument')
                        end
                    elseif isequal(type,'datetime')
                        if ~isempty(find(isnat(val),1)) %The code is searching for NaT values
                            error('NaT values cannot be used for a >< argument')
                        end
                    end
                case 8 %Keep all data greater than or equal to the lower threshold or strictly
lower than the upper threshold
                    if length(val)==1
                        error('Two values are needed for a >=< logical argument')
                    end
                    if isequal(type,'double')
                        if ~isempty(find(isnan(val),1)) %The code is searching for NaN values
                            error('NaN values cannot be used for a >=< argument')
```

```
                        end
                    elseif isequal(type,'datetime')
                        if ~isempty(find(isnat(val),1)) %The code is searching for NaT values
                            error('NaT values cannot be used for a >=< argument')
                        end
                    end
                case 9 %Keep all data greater than the lower threshold or lower than or equal to
the upper threshold
                    if length(val)==1
                        error('Two values are needed for a ><= logical argument')
                    end
                    if isequal(type,'double')
                        if ~isempty(find(isnan(val),1)) %The code is searching for NaN values
                            error('NaN values cannot be used for a ><= argument')
                        end
                    elseif isequal(type,'datetime')
                        if ~isempty(find(isnat(val),1)) %The code is searching for NaT values
                            error('NaT values cannot be used for a ><= argument')
                        end
                    end
                case 10 %Keep all data greater than or equal to the lower threshold or less than
or equal to the upper threshold
                    if length(val)==1
                        error('Two values are needed for a >=<= logical argument')
                    end
                    if isequal(type,'double')
                        if ~isempty(find(isnan(val),1)) %The code is searching for NaN values
                            error('NaN values cannot be used for a >=<= argument')
                        end
                   elseif isequal(type,'datetime')
                        if ~isempty(find(isnat(val),1)) %The code is searching for NaT values
                            error('NaT values cannot be used for a >=<= argument')
                        end
                    end
                otherwise
                    error('%s is not a valid logical argument input. See documentation for a list
of valid logical argument formatting',logarg)
            end
end
```

Published with MATLAB® R2021a

```
function [TS,IDs,IDindex,T] = GetTimeSeries(varargin)
%GetTimeSeries Function extracts a timeseries structure from the database
%based on submitted inputs. There are two methods this function uses to
%extract the timeseries data from the database.
%
% i) Enter the string "IDList" followed by a cell or string of
% either Deployment IDs or Event IDs (can be mixed).If a cell/string array
% of potential timeseries IDs is submitted, the function verifies that the
% IDs are correct and then output a structure containing these time series.
% Note that in the case of combining Event time series and Deployment time
% series, there will be two seperate sub structures for the different types
% of time series.
%
% ii) The user requests timeseries that fit a certain set of parameters
% using name-value pairs. The type of time-series can be indicated using
% "Type" followed by either "Event" or "Deployment". If not
% specified, the function will output both types of time series.
%
% All other name value pairs follow the same procedure as for the
% GetTable() function, as it it used to refine the time series list.
%OUTPUT
%  TS - structure containing the requested timeseries

%  IDs - ID list (useful in plotting time-series in a particular order in
%  SCPlot

%  IDindex - a 2x1 array that states which sub-structure each time series
%  is in, and the index of the time series within the time structure.
%   Ex. the 65th time series in Events sub structure when all time-series
%   are outputted(second sub structure) will have the index of [2 65].
%   If only Events time series are outputted, it would be [1 65].
%   T - Event Table;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Author: Sean R. Boyd January 19th, 2022
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Check number of arguments
direct = pwd;
%Load Database
SupercoolingDatabase=GetDatabase();
%Extract structure of timeseries type
TS=SupercoolingDatabase.Time_Series; %Base output of the function
%Extract all IDs
deploy = [TS.Deployments.Deployment_ID];
evts = string({TS.Supercooling_Events.Event_ID});
IDs = [deploy,evts];
%Create a list of ID indicies
IDindex = zeros(length(IDs),2);
for k = 1:length(deploy)
    IDindex(k,1)=1;IDindex(k,2)=k;
end
for k = length(deploy)+1:length(IDs)
    IDindex(k,1)=2;IDindex(k,2)=k-length(deploy);
end
```

```
%Process based on function inputs
if isempty(varargin)
    cd(direct);
    if nargout == 4
        T=GetTable();
    end
    return
elseif nargin>=2 %name value pairs. Use GetTable to build a list
    %Determine if the Time Series type is specified
    %Search for "IDList" in time series. If found, extract the
    chk=1;k=1;
    while chk && k<=nargin
        if strcmpi(varargin{k},'IDList')
            if k<nargin
                IDs =  varargin{k+1};chk=0;idlistchk=1;
                %check that IDs is a cell/string
                if ~isa(IDs,'string')
                    try
                        IDs = string(IDs);
                    catch
                        error('List of IDs could not be converted to a string array.')
                    end
                end
            else
                error('Time Series List was not submitted. Please submit a list of Deployment or
Event IDs that you want the time series for.')
            end
            %If this is an array, convert to a vector
            if ~isvector(IDs)
                IDs = reshape(IDs,1,[]);
            end
        else
            k=k+1;
        end
    end
    if k>nargin
        IDs=[];idlistchk=0;
    end

    %If  IDs is not empty, skip the rest of the function
    if isempty(IDs)
        chk=1;k=1;
    else
        chk=0;k=nargin+1;
        %Check that ID list is valid
        type = string(size(IDs));
        for k1 = 1:length(IDs)
            if find(deploy == IDs(k1),1)
                type(k1) = "Deployment";
            elseif find(evts == IDs(k1),1)
                type(k1) = "Event";
            else
                fprintf('\nInvalid Time Series ID: %s\n',IDs(k1))
            end
```

```
        end
        %Generate a summary table for valid IDs
        T=GetTable("Deployment_ID",IDs(type == "Deployment"),"Event_ID",IDs(type == "Event"));
        if isempty(T) %No valid inputs from IDlist
            T="No valid Supercooling Events in teh submitted Time Series";
        end
    end
 while chk && k<=nargin %Search for a value pair for 'Type'
        if strcmpi(varargin{k},'Type')
            if k<nargin
                type =  varargin{k+1};chk=0;
            else
                error("Time Series type is not specified. Please enter 'Deployment'/'Deployments'
or 'Event'/'Events' to specify the type of timeseries. Otherwise, leave blank to output both
sets")
            end
        else
            k=k+1;
        end
    end
    if k>nargin
        type=[];
    end
    if ~idlistchk
        if ~isempty(type)
            %Determine if the type submitted was valid
            if strcmpi(type,'Deployment') || strcmpi(type,'Deployments')
                TS.Supercooling_Events(:,:)=[];
            elseif strcmpi(type,'Event') || strcmpi(type,'Events')
                TS.Deployments(:,:)=[];
            else
                warning("Time_Series type was invalid (not  'Deployment'/'Deployments' or
'Event'/'Events'). Thus both sets of timeseries will be outputted.")
            end
            %Remove the type indicator from varargin before building your table
            inpts = varargin;inpts(k:k+1)=[];
        else %No type indicator to remove
            inpts = varargin;
        end
        %Generate Table
        T=GetTable(inpts{:});
        if strcmpi(type,'Deployment') || strcmpi(type,'Deployments')
            IDs = unique(T.Deployment_ID);
        elseif strcmpi(type,'Event') || strcmpi(type,'Events')
            IDs = T.Event_ID;
        else
            %Create an IDs list of remaining deployment and event IDs
            IDs = [unique(T.Deployment_ID),;T.Event_ID];
        end
    end
    %Filter for Deployment and Events
    [TS,IDindex] = filterTS(TS,IDs);
```

```
%After the TS sub structures have been refined, remove any empty
    %substructures.
    if isempty(TS.Deployments)
        TS=rmfield(TS,'Deployments');
    end
   if isempty(TS.Supercooling_Events)
        TS=rmfield(TS,'Supercooling_Events');
    end
else
    warning('Insufficent number of entries to filter time series based on name-value pairs.
Complete Time-series set outputted instead.')
    return
end
end
%SUB-FUNCTIONS
function [TS,IDindex] = filterTS(TS,IDs)
    %Get the list of Deployment IDs
    IDs = string(IDs); %String array of the submitted IDs
    IDindex = zeros(length(IDs),2); %Storage array of index
    deploys=string({TS.Deployments.Deployment_ID});
    %Get the list of Event IDs
    evts=string({TS.Supercooling_Events.Event_ID});
    %For each time series set, determine which of those sets are in the
    %submitted list
    % Deployments
    del=zeros(size(deploys));
    for k = 1:length(deploys)
        rows = find(IDs == deploys(k));
        if isempty(rows)
            del(k)=k;
        else
            IDindex(rows,1)=1;
        end
    end
    del(del==0)=[];TS.Deployments(:,del)=[];deploys(del)=[];

    %Review revised Deployment Substructure to and adjust the secondary index
    if ~isempty(TS.Deployments)
        deploy = [TS.Deployments.Deployment_ID];
        for k = 1:length(IDs)
            r = find(deploy == IDs(k),1);
            if ~isempty(r)
               IDindex(k,2) = r;
            end
        end
    end
```

```
    %Events
    del=zeros(size(evts));
    for k = 1:length(evts)
        rows = find(IDs == evts(k));
        if isempty(rows)
            del(k)=k;
        else
            IDindex(rows,1)=2;
        end
    end
    del(del==0)=[];TS.Supercooling_Events(:,del)=[];
    %Review revised Event Sustructure to and adjust the secondary index
    if ~isempty(TS.Supercooling_Events)
        evts = string({TS.Supercooling_Events.Event_ID});
        for k = 1:length(IDs)
            r = find(evts == IDs(k),1);
            if ~isempty(r)
               IDindex(k,2) = r;
            end
        end
    end
    %If only the Event Substruture exist, change the 1st index to 1.
            if isempty(deploys) %There is only 1 substructure
                IDindex(:,1) =1;
            end
end
```

Published with MATLAB® R2021a

```
function ClearDatabase(clearAll)
%ClearDatabase: Function resets the SupercoolingDatabase, and prompts the
%user if they wish to reset the Screened Periods records (if this section
%is cleared all previously screened periods will be re-flagged.)
%INPUT
%   clearAll: Enter 'All' or "Reset' to skip the query menu to confirm that
%   you wish to clear the database.
%OUTPUT
%   - sets CurrentSuperCoolingDatabase to an empty structure (keeping the
%    list of screened events and the change list for the rawfilelist)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Author: Sean R. Boyd January 19th, 2022
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Check number of inputs
narginchk(0,1)
direct=pwd; %Current Directory
DB=strcat(direct,'\Databases'); %Folder that holds the Database
Lists=strcat(direct,'\Raw Data Files'); %Folder that holds the RBR .rsk files and Lists of
Processed and New Files
cd(DB)
if nargin == 0 || strcmpi(clearAll,'All')|| strcmpi(clearAll,'Reset')%Reset the entire database
    if nargin == 0 %Check if call for reset is unintentional
        clearDatabase = input('Do you wish to clear the current Super-Cooling Database? (Y/N):
','s');
```

```
        if strcmpi(clearDatabase,'y')
            %Load old database to extract archive and screened periods list
            load('CurrentSuperCoolingDatabase.mat','SupercoolingDatabase');
            archive = SupercoolingDatabase.Archive_Notes;screen =
SupercoolingDatabase.Screened_Periods;
            SupercoolingDatabase =
struct('Observation_Summaries',struct('Rivers',{},'Sites',struct('Site',{},'Deployments',table())
),...

'Event_Table',table(),'Time_Series',struct('Deployments',struct('Deployment_ID',{},'Deployment_Da
taTable',timetable()),...

'Supercooling_Events',struct('Event_ID',{},'Event_DataTable',timetable())),'Screened_Periods',[],
'Archive_Notes',archive);
            %Reset the 'Processed' column of RSKFileList to "No"
            cd (Lists);
            load('CurrentRawFileList.mat','rawfileList')
            rawfileList.Processed(:) = "No";save('CurrentRawFileList.mat','rawfileList')
            cd (direct)
        else
            cd (direct);disp('ClearDatabase terminated');pause(1);clc;
            return
        end
    else %Deliberate Reset of Database
        load('CurrentSuperCoolingDatabase.mat','SupercoolingDatabase');
        archive = SupercoolingDatabase.Archive_Notes;screen =
SupercoolingDatabase.Screened_Periods;
        SupercoolingDatabase =
struct('Observation_Summaries',struct('Rivers',{},'Sites',struct('Site',{},'Deployments',table())
),...

'Event_Table',table(),'Time_Series',struct('Deployments',struct('Deployment_ID',{},'Deployment_Da
taTable',timetable()),...

'Supercooling_Events',struct('Event_ID',{},'Event_DataTable',timetable())),'Screened_Periods',[],
'Archive_Notes',archive);
        %Reset the 'Processed' column of CurrentRSKFileList to "No"
        cd (Lists);
        load('CurrentRawFileList.mat','rawfileList')
        rawfileList.Processed(:) = "No";save('CurrentRawFileList.mat','rawfileList')
    end
else
    cd (direct)
    error("Invalid input. ClearDatabase runs a confirmation query if no input is given, or
accepts 'All' or 'Reset' as valid inputs to skip the query.")
end
%Check if the user wants to clear screened events
if ~isempty(screen.Summary)
    chk=1;
else
   chk=0;
end
```

```
while chk
    inpt = input('Do you wish to clear the records of screened periods? (Y/N)?: ','s');
    if strcmpi(inpt,'y')
        screen =
struct('Summary',table(),'TimeSeries',struct("TimeSeries_ID",[],"TimeSeries",[]));
        fprintf('Records of screened periods have been cleared.\n');pause(0.9);clc;chk=0;
    elseif strcmpi(inpt,'n')
        fprintf('Records of screened periods have been kept.\n');pause(0.9);clc;chk=0;
    else
         fprintf('Invalid input.\n')
    end
end
%Assign the correct value for screened periods records
SupercoolingDatabase.Screened_Periods = screen;
%Save the database then return home
cd(DB)
save('CurrentSuperCoolingDatabase.mat','SupercoolingDatabase');
clc;disp('CurrentSuperCoolingDatabase Reset');pause(0.9);clc;cd(direct)
```

Published with MATLAB® R2021a

```
function SCPlot(varargin)
%SCPlot: Function plots times series from the database based on submitted
%inputs. Since this program utilises GetTimeSeries, it has a similar
%functionality in terms of two main methods for extracting time series:
%
% i) Enter the string "IDList" followed by a cell or string of
% Either Deployment IDs or Event IDs (can be mixed).If a cell/string array
% of potential timeseries IDs is submitted, the function verifies that the
% IDs are correct and then output a structure containing these time series.
% Note that in the case of combining Event time series and Deployment time
% series, there will be two seperate sub structures for the different types
% of time series.
%
% ii) The user requests timeseries that fit a certain set of parameters
% using name-value pairs. The type of time-series can be indicated using
% "Type" followed by either "Event" or "Deployment". If not
% specified, the function will output both types of time series.
%
% All other name value pairs follow the same procedure as for the GetTable()
% function, as it it used to refine the time series list.
%
%In addition to the above, there are name value pairs to help with plotting
%the timeseres as listed below.
%
%Since this plotting tool can be used to specify exact time series you wish
%to plot ("IDList",[string of time-series IDs]) as well as a query for
%generating figures about deployments and events that fit a certain
%specification, the program makes the following check:
%
```

```
%If you do not submit a list of specific time series (IDList) to print, a
%single value assignment to the plotting value pairs apply to all figures
%generated.
% Ex. If you write SCPlot("Deployment_Period","2015-2016","Shading","off")
% the program would generate a figure for every deployment in 2015-2016,
% and not shade in the recorded supercooling events, as they are all set to
% 'off'.
%
%LIST OF NAME - VALUE PAIRS FOR PLOTTING
%This is a list of the currently enabled Name Value Pairs. If you wish to
%add other name value pairs, please add a description of them to this
%documentation for clarity of use.
%
% 'FolderName' - [foldName] = Name of folder storing figures. Default
% is 'Generated Figures - mmm-dd-yyyy-hhmmss'. All figures generated in the
% same instance of this program will save to the same folder.
%
% 'FigureName' - [fig] = Array of figure names that each ID will be
%  plotted to.
%  There are 2 possible cases for this parameter:
%
%       i) If this is empty as well as Axes, each ID will have a figure
%       window generated, with the times series ID as the figure name
%
%       ii) If Axes are submitted (and thus time series are assigned to a
%       pre-existing figure window), and the parent figure of the axes does
%       not have a name, the FigureName will become the new figure name.
%       If multiple time series are assigned to a single figure with
%       different figure names, a generic figure name will be generated
%       instead.
%
% 'Axes'-[ax] = Array of the axes you wish to plot the IDs to.
%  These axes will have to be existing axes prior to running the program.
%  This enables you to format a figure window, then assign time-series data.
%  If this is not submitted, the function will generate the figure window
%  and axes for plotting
%
%       Ex. SCPlot("IDList",[ID1,ID2,ID3],"Axes",[ax1,ax2,ax3])
%       plots ID1 to ax1, ID2 to ax2, and ID3 to ax3.
%
%  'Title' - [ttl] = Array of the plot titles. Default is the times
%   series ID. This will enable for the user to tell which time series they
%   are looking at on which axes. if multiple time-series are plotted to
%   one axes, the program will ask if the user wishes to have multiple time
%   series on the axes. If so, the title will be updated to reflect the
%   time series on the plot
%
%  'Filter' -[fltr] = Array indicating if you are plotting filtered data.
%  The default is 'on', meaning your are filtering the data presented vs.
%  'off' where you are showing the unfiltered data. For deployment time
%   series, the filter removes the portion of the time series between end
%   of freeze-up and start of break-up. For Event time time series, the
%   filter removes all the data before or after the event, leaving you
%   with just the timeseries of supercooling temperature.
```

```
%
%  'MarkFreezeBreak' - [markfrzbrk] = Array indicating if you wish to mark
%   the end of freeze-up and start of break-up on the plot. If these
%   markers are not in the range of dates, the default is 'off', otherwise
%   it is 'on'.
%
%  'CRP' - [slope] = Array indicating if you wish to plot the
%   principal supercooling average cooling rate. Default is 'off'.
%
%  'SensorAccuracy' - [acc] - Array indicating if you wish to plot the
%   sensor accracy of the sensor used. Default is 'off'.
%
%  'NegativeThreshold' - [thresh] - Array indicating how you wish to handle
%  the negative threshold used during analysis. If it is 'on', the lowest
%   bound of the data plotted will be the negative theshold. If it is
%   'off', the lowest bound is -1 C at most, and the negative threshold is
%   plotted on the figure. Default value is 'on'
%
%   'Shading' - [s] - Array indicating if you want supercooling events to
%   be shaded. Default is 'on'.
%
%  'Legend' - [leg] - Either 'on' or 'off' to indicate if you want a
%   legend for your figure. Default is 'off'. If legend is on for multiple
%   axes in a figure, then one legend is formed incorporating both axes by
%   printing a legend that contains all the results printed across all the
%   axes
%
%   'Grid' - [grd] - Array indicating if you want the grid on or off.
%   Default is 'on'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Author: Sean R. Boyd January 19th, 2022
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Set Directories
direct=pwd;resultsfold=strcat(direct,'\Generated Results');
%Get Time-Series (this will ignore any plotting inputs)
fprintf("Getting Time-Series data...")
[TS,IDs,IDex,T]=GetTimeSeries(varargin{:});
if isempty(IDs)
    clc;fprintf('\nNo Events were found to meet the submitted requirements.\n')
else
    %Get observation summary for accessing any values
    [summary] = GetObSummary("IDList",IDs,"Print","n");
    fprintf("Complete\nDetermining plot settings...")
    %Create default arrays for the plottig variable, and checks for if the
    %parameter has been assigned
    foldName = sprintf('Supercooling Figures - Generated %s',datestr(now,'mmm dd yyyy HHMMss'));
foldchk=0;
    figNames = IDs; %Default figure name is the ID
    ax = [];%Default assume a new axes is generated for each time series
    ttl = strings(size(IDs));
```

```
for k=1:length(IDs)
        ttl(k) = sprintf('Time Series: %s',IDs(k));
    end
    fltr = strings(size(IDs));fltr(:,:)='on'; %Default assumption is that the filter is on (do
not show filtered data)
    markfrzbrk = strings(size(IDs));markfrzbrk(:,:)='off';%Default assumption is to not mark
freeze-up and break-up
    slope = strings(size(IDs));slope(:,:)='off';%Default assumption is to not mark the CRP
    acc = strings(size(IDs));acc(:,:)='off';%Default assumption is to not show sensor accuracy
    thresh = strings(size(IDs));thresh(:,:)='on'; %Default assumption is to not show negative
threshold line
    s = strings(size(IDs));s(:,:)='on'; %Default assumption is to shade supercooling events
    leg = strings(size(IDs));leg(:,:)='off'; %Default assumption is to not show the legend for
the plot
    grd = strings(size(IDs));grd(:,:)='on';
    %Determine if a known number of events were submitted (IDList for time
    %series). If so, flag so that single string pairs apply to all figures
    %generated
    idlist=0;
    while ~idlist && k<=nargin
        if isstring(varargin{k})
            if strcmp(varargin{k},'IDList')
                idlist=1;
            end
        end
        k=k+1;
    end
    %Parse varargin for submitted plotting inputs. Verify plotting inputs
    %(output warnings when elements are corrected, and errors when there are
    %incompatible conflicts
    chk=1;k=1;
    while chk && k<=nargin
        if isstring(varargin{k}) ||ischar(varargin{k})
            switch varargin{k}
                case 'FolderName'
                    varglength(k,varargin)
                    foldName = string(varargin{k+1}); %Select folder name (next input)
                    if ~isstring(foldName)
                        foldName = string(foldName);
                    end
                    %Create a folder
                    try
                        [figFolder] = FolderBuilder(resultsfold,foldName);foldchk=1;
                    catch
                        error('%s is not a valid folder name',foldName)
                    end
                case 'FigureName'
                    varglength(k,varargin)
                    inpts = string(varargin{k+1});
```

```
            %Assign input to fig as a string
                for k1 = 1:length(inpts)
                    try
                        figNames(k1) = inpts(k1);
                        if length(inpts)<length(IDs)
                            fprintf('Not all time series have a figure name assigned.\nThose
without an assigned figure name will use a default one.')
                        end
                    catch
                        error('%s could not be assigned as a string value for a figure
name.',inpts(k1))
                    end
                end
            case 'Axes'
                varglength(k,varargin)
                ax = varargin{k+1};
                if length(ax)<length(IDs)
                    fprintf('Not all time series have an axes assigned.\nThose without an
assigned axes will have a\nfigure generated for them')
                end
            case 'Title'
                varglength(k,varargin)
                inpts = string(varargin{k+1});
                %Assign input to ttl as a string
                for k1 = 1:length(inpts)
                    try
                        ttl(k1) = inpts(k1);
                        if length(inpts)<length(IDs)
                            fprintf('Not all time series have a title assigned.\nThose
without an assigned title will use a default one.')
                        end
                    catch
                        error('%s could not be assigned as a string value for a
title.',inpts(k1))
                    end
                end
            case 'Filter'
                varglength(k,varargin)
                inpts = string(varargin{k+1});
                [fltr] = onoffArray(fltr,inpts,varargin{k},idlist);
            case 'MarkFreezeBreak'
                varglength(k,varargin)
                inpts = string(varargin{k+1});
                [markfrzbrk] = onoffArray(markfrzbrk,inpts,varargin{k},idlist);
            case 'CRP'
                varglength(k,varargin)
                inpts = string(varargin{k+1});
                [slope] = onoffArray(slope,inpts,varargin{k},idlist);
            case 'SensorAccuracy'
                varglength(k,varargin)
                inpts = string(varargin{k+1});
                [acc] = onoffArray(acc,inpts,varargin{k},idlist);
```

```
                case 'NegativeThreshold'
                    varglength(k,varargin)
                    inpts = string(varargin{k+1});
                    [thresh] = onoffArray(thresh,inpts,varargin{k},idlist);
                case 'Shading'
                    varglength(k,varargin)
                    inpts = string(varargin{k+1});
                    [s] = onoffArray(s,inpts,varargin{k},idlist);
                case 'Legend'
                    varglength(k,varargin)
                    inpts = string(varargin{k+1});
                    [leg] = onoffArray(leg,inpts,varargin{k},idlist);
                case 'Grid'
                    inpts = string(varargin{k+1});
                    [grd] = onoffArray(grd,inpts,varargin{k},idlist);
            end
        end
        k=k+1;
    end
    %For anything not assigned, set the defaults
    %Folder
    if ~foldchk
        %Create a folder
        try
            [figFolder] = FolderBuilder(resultsfold,foldName);
        catch
            error('%s is not a valid folder name',foldName)
        end
    end
    %Create a reference table to store the time-series with their name value
    %pairs
    tab = table('Size',[length(IDs),
14],'VariableType',["double","double","string","double","string",...

"string","string","string","string","string","string","string","string","string"],'VariableNames'
,...

["Fig_Index","Ax_Index","ID","ID_Index","FigureName","Title","Filter","FreezeBreak","CRP","Acc",.
..
        "Thresh","Shade","Grid","Leg"]);
    for k = 1:length(IDs)
        tab.ID(k) = IDs(k);
        tab.ID_Index(k,1:size(IDex,2)) = IDex(k,:);tab.FigureName(k) = figNames(k);tab.Title(k) =
ttl(k);
        tab.Filter(k)= fltr(k);tab.FreezeBreak(k)=markfrzbrk(k);tab.CRP(k)=slope(k);tab.Acc(k) =
acc(k);
        tab.Thresh(k)=thresh(k);tab.Leg(k)=leg(k);tab.Shade(k)=s(k);tab.Grid(k) = grd(k);
    end
    %Review current titles and replace underscores with spaces
    for k = 1:height(tab)
       tab.Title(k) = strrep(tab.Title(k),"_"," ");
    end
```

```
    %Use the reference table to organize how time-series are being plotted
    % The hierarchy goes: 1) Figure, 2) Axes withing figure, 3) Alphabetical
    % order of IDs
    %1) Go through all axes and determine if there are any parent figures, and
    %note which time series belong to that parent figure
    if ~isempty(ax) %ax is in the order of the IDs they will be assigned to
        figs = matlab.ui.Figure.empty(length(ax),0); %Array of parent figures for the axes
        for k = 1:length(ax)
            if strcmp(get(ax, 'type'), 'axes')
                figs(k) = ax(k).Parent;
            else
                error("There were non-axes inputs for the 'Axes' parameter")
            end
        end
    else
        figs=[];
    end
    fprintf('Complete.');pause(1.2);clc;fprintf('Generating Figures...\n')
    if isempty(figs) %There are no pre-generated figures/axes. Each time series gets their own
figure/axes
        for k =1:height(tab)
            figname=tab.FigureName(k);
            clc;fprintf('Generating Figures...\nFigure %d of %d: %s',k,height(tab),figname)
            f=figure('Name',figname,'IntegerHandle','off');tsax = gca;box on;
            TSTab = tab(k,:); %Table summarry of the time-series
            %Plot time-series
            [~]=TSPlot(tsax,TSTab,TS,T,summary);
            %save figure
            saveas(f,sprintf('%s/%s.fig',figFolder,figname))
        end
    else %There are pre-generated figures/axes.
        %Determine how many unique figures there are
        uniquefigs=unique(figs);uniqueaxs=unique(ax); %Get the unique figures and axes
        %Currently, figs and ax are ordered according to the time series that
        %is assigned to them (ax(1) if from figs(1) is for the time series of
        %ID(1), etc.), which is reflected in the ordering of IDs in tab. Assign
        %a fig and ax index for each time series
        for k = 1:height(tab)
            tab.Fig_Index(k) = find(uniquefigs == figs(k),1); %Index of the time series figure in
uniquefigs
            tab.Ax_Index(k) = find(uniqueaxs == ax(k),1); %Index of the time series figure in
uniqueaxes
        end
        %Check if any time series share figures and assess the figure names
        %submitted in case of conflicts
        for k = 1:length(uniquefigs)
            figtab = tab(tab.Fig_Index == k,:);
            fignames = unique(figtab.FigureName);
            if length(fignames)>1
                tab.FigureName(tab.Fig_Index == k) = sprintf('Figure %d',k);
                fprintf('\n Conflict has been found in the suggested name for a figure.\nA new
name has been generated.\n');pause(2)
            end
        end
```

```
        %Check if any time series share the same axes. If so, switch hold on
        %for those time series, and adjust title to reflect the time series on
        %the axes.
        for k = 1:length(uniqueaxs)
            axtab = tab(tab.Ax_Index == k,:);
            %Adjust title as needed
            axttls = unique(axtab.Title);
            if length(axttls)>1
                for k1 = 1:length(axttls)
                    if k1 == 1
                        newttl = axtab.Title(k1);
                    elseif k1==length(axttls)
                        newttl = sprintf('%s & %s',newttl,axtab.ID(k1));
                    else
                        newttl = sprintf('%s, %s',newttl,axtab.ID(k1));
                    end
                    %Replace anny underscores with a space
                    newttl = strrep(newttl,"_"," ");
                end
                tab.Title(tab.Ax_Index == k) = newttl;
                fprintf('\n Multiple time series have been plotted to the same axes\nA new axes
title has been generated.\n');pause(2)
            end
        end
        %Plot all time-series in the following order:
        %1) Figure
        for k = 1:length(uniquefigs)
            figtab = tab(tab.Fig_Index == k,:); %Table of time tables in the specified figure
            figname = unique(figtab.FigureName);
            figaxs = unique(figtab.Ax_Index); %All unique axes in this figure window
            %Print out progress
            clc;fprintf('Generating Figures...\nFigure %d of %d:
%s\n',k,height(tab),tab.FigureName(k));
            uniquefigs(k).Name = figname;
            %2) Axes within figure
            for k1 = 1:length(figaxs)
                axtab = figtab(figtab.Ax_Index == figaxs(k1),:); %All unique time series in a
specific axes
                axttl = unique(axtab.Title);
                fprintf('Axes %d of %d   Title: %s\n', k1,length(figaxs),axttl)
                %3) Alphabetical order of IDs
                for k2 = 1:height(axtab)
                    if height(axtab)>1
                        fprintf('Time-Series %d of %d: %s\n',k2,height(axtab),axtab.ID(k2))
                    else
                        fprintf('Time-Series: %s\n',axtab.ID(k2))
                    end
                    TSTab = tab(tab.ID == axtab.ID(k2),:);%Table row containing the information
for the specific time series
                    tsax = uniqueaxs(figaxs(k1));[~]=TSPlot(tsax,TSTab,TS,T,summary);
                end
            end
```

```
            %Save figure
             saveas(uniquefigs(k),sprintf('%s/%s.fig',figFolder,figname))
        end
        %Generate unique figures for time series with no submitted figures
        tab1 = tab(tab.Fig_Index == 0,:);
        if ~isempty(tab1)
            for k =1:height(tab1)
                figname=tab1.figNames;
                clc;fprintf('Generating Figures...\nFigure %d of %d:
%s',k,height(tab1),tab1.FigureName)
                figure('Name',figname,'IntegerHandle','off');tsax = gca;
                [f]=TSPlot(tsax,TSTab,TS,T,summary);
                saveas(f,sprintf('%s/%s.fig',figFolder,figname))
            end
        end
    end
    % After all figures have been generated, display that
    % the program is complete
    fprintf('All Full Season Figures Printed and saved to Generated Results under:
%s',foldName);pause(3);clc;
end
end

%Subfunctions
%Folder Builder
function [mainFolder,subFolders,exitCue] =
FolderBuilder(parentFolder,mainFolderName,subFolderNames,requestOverWrite)
%FolderBuilder This Function generates a folder with any required
%sub-folders as specified by the sub-folder name array.  If the folder
%already exists at the destination, this function automatically overwrites
%the pre-existing folder(s) unless requestOverwrite is set to 'on'.
%INPUTS
%   parentFolder = the parent folder that this main folder will be sent to
%   mainFolderName = the name of the folder that will be generated
%   subFolderNames = the name(s) of any sub-folders within the main folder
%   that you wish to add. Default: empty cell array
%   requestOverWrite = Set to 'On' if you wish for the function to confirm
%   that an existing folder with the same name at the destination should be
%   over written before proceeding. Default: 'off'
%OUTPUTS
%   mainFolder; string array of the file path destinations/saving purposes
%   subFolders: String array containing string arrays of the paths for the subfolders
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
narginchk(2,4)
exitCue = 0;
if nargin == 2
    subFolderNames=[];requestOverWrite = 'off';stopProg = 'off';
elseif nargin == 3
    requestOverWrite = 'off';stopProg = 'off';
elseif nargin == 4
    if strcmp(requestOverWrite,'on')
        stopProg = 'on';
    else
        stopProg = 'off';
```

```
    end
end
%Generate folder and sub folder names
if isstring(parentFolder)
    parentFolder=parentFolder{:};
end
if isstring(mainFolderName)
    mainFolderName=mainFolderName{:};
end
mainFolder = strcat(parentFolder,'\',mainFolderName); %Create full path for main folder
if ~isempty(subFolderNames)
    %Check size of subfolders to see if it is a character or string vector
    if iscell(subFolderNames) %Convert cell to string
        subFolderNames = string(subFolderNames);
    end
    if ~isvector(subFolderNames) || (~isstring(subFolderNames) && ~ischar(subFolderNames))
        error('SubFolderNames should be a vector array of character or string, or a cell vector
of the same')
    end
    subFolders = cell(size(subFolderNames));
    for k = 1:length(subFolderNames)
        subFolders{k}=strcat(mainFolder,'\',subFolderNames{k});
    end
end
%Assess if file is already in directory and checks if you want to continue
[status,msg]=mkdir(string(parentFolder),mainFolderName);
if status == 0
    error('File failed to write to Directory')
elseif isempty(msg) == 0 && strcmp(requestOverWrite,'on')
    MSG = strcat('Do you wish to overwrite the pe-existing copy of'," ",mainFolderName,"
",'[Y,N]?');
    [request,exitCue] = InputCheck('char',MSG,{'Y','y','N','n'},1,1,stopProg);
    if strcmp(request,'n') || strcmp(request,'N')
        exitCue=1;fprintf('%s is not overwritten\n',mainFolderName)
    end
end
if isempty(msg) == 0 && (strcmp(requestOverWrite,'off') || strcmp(request,'y') ||
strcmp(request,'Y'))
    rmdir(mainFolder, 's'); mkdir(mainFolder)
end
%Generate Sub folders if required
if isempty(subFolderNames) == 0 && exitCue == 0
    for k = 1:length(subFolderNames)
        mkdir(mainFolder,[subFolderNames{k}]);
    end
end
end

%Plotting prep subfunction
function [t,Tw,tsType,evtsTab,sensAcc,negThresh,tRange,frz,brk]=TSPlotPrep(TSTab,TS,T,summary)
%Determine the type of the TS
fldname = string(fieldnames(TS));tsType = fldname(TSTab.ID_Index(1));
%Get Time Series
if strcmp(tsType,"Supercooling_Events") && strcmp(TSTab.Filter,'off')
```

```
    %Process like a deployment time series for this event but rescale
    %x limits to this event after plotting the data.
    %Get the required x-limits
    evtsTab = T(T.Event_ID == TSTab.ID,:);eStart = evtsTab.Start_Time;
    eEnd = evtsTab.End_Time;dur = evtsTab.Duration;
    %Get required deployment information
    tsType='Deployments';evtfiltoff=1;
    [TS,~,~,evtsTab] = GetTimeSeries("IDList",unique(evtsTab.Deployment_ID));
else
    evtfiltoff=0;
end
if strcmp(tsType,"Supercooling_Events")
    %Get the event time series if the filter is on
    tt = TS.Supercooling_Events(TSTab.ID_Index(2)).Event_DataTable;
    if isempty(tt)
        t=NaT();Tw=NaN();
    else
        t=tt.Time;Tw = tt.Water_Temperature;
    end
    %The rest applies to all Event plots
    evtsTab = T(T.Event_ID == TSTab.ID,:);
    eStart = evtsTab.Start_Time;eEnd = evtsTab.End_Time;dur = evtsTab.Duration;
elseif strcmp(tsType,"Deployments")
    if evtfiltoff
        %Get the deployment time series for this event
        tt = TS.Deployments(1).Deployment_DataTable;
        t=tt.Time;Tw = tt.Water_Temperature;
    else %Regular deployment plot
        %The rest applies to all Deployment plots
        evtsTab = GetTable("Deployment_ID",TSTab.ID);
        eStart = evtsTab.Start_Time(1);eEnd = evtsTab.End_Time(end);dur = hours(eEnd-eStart);
        tt = TS.Deployments(TSTab.ID_Index(2)).Deployment_DataTable;
        if strcmp(TSTab.Filter,'off')
            t=tt.Time;Tw = tt.Raw_Water_Temperature;
            %Get End of Freeze-Up and Start of Break-up (plot
            %the period between the two dates as a dashed line
        else
            t=tt.Time;Tw = tt.Water_Temperature;
        end
    end
else
    error("%s is an unexpected Time Series type. Time Series Type should either be Deployments or
Supercooling_Events",tsType)
end
%Get the information from the summary. Status values in TSTab will determine if anything is done
with the information
%Determine teh key values from evtsTab to locate the specific deployment in
%the summary
tsRiv = unique(evtsTab.River);tsSite = unique(evtsTab.Site); tsDeploy =
unique(evtsTab.Deployment_ID);
allrivs = string({summary.Rivers});r = find(allrivs==tsRiv,1); %river index
allsites = string({summary(r).Sites.Site});s = find(allsites==tsSite,1); %site index
siteT = summary(r).Sites(s).Deployments  ; %Deployment summary table
alldeploy = string([siteT.Properties.RowNames]);d = find(alldeploy==tsDeploy,1); %deployment
```

```
index

%Freeze-Up and Break-Up dates
frz=siteT.Freeze_Up_End_Date(d);brk=siteT.Break_Up_Start_Date(d);
%Sensor Accuracy
sensAcc=siteT.Sensor_Accuracy(d);
%Negative Threshold
negThresh=siteT.Negative_Threshold(d);
%Set tRange
if isnan(dur) %There are no supercooling events to scale the season
    dur = days(t(end)-t(1));eStart = t(1);eEnd = t(end);
end
%Sets the time range to make the middle 80% of the plot taken up by the time-series
shift = 0.1*dur;tRange = [eStart-hours(shift) eEnd+hours(shift)];
end

%Plotting subfunction
function [f]=TSPlot(tsax,TSTab,TS,T,summary)
%Prep the inputs from the inputs
[t,Tw,tsType,evtsTab,sensAcc,negThresh,tRange,frz,brk]=TSPlotPrep(TSTab,TS,T,summary);
if ~isnat(t) %There is a time period to plot
%Set the function to plot to the correct axes
f=tsax.Parent;set(f,'CurrentAxes',tsax);g=TSTab.Grid;hold on;

    %Get the Start and End Indicies of supercooling events
    times = unique([evtsTab.Start_Time(:)',evtsTab.End_Time(:)']);
    if~isnat(evtsTab.Start_Time(1))
    indicies = zeros(size(times));
    for k = 1:length(times)
        indicies(k) = find(t == times(k),1);
    end
    else
        indicies = [];
    end
    if strcmp(TSTab.Shade,'on')
        if~isempty(indicies)
        %Mark Super-Cooling Events
        for k = 1:2:length(indicies)-1

shade(t(indicies(k):indicies(k+1)),Tw(indicies(k):indicies(k+1)),'FillType',[0,1],'FillColor',[0.
30,0.75,0.93],'FillAlpha',0.25);
            if find(Tw(indicies(k):indicies(k+1))>0,1)

shade(t(indicies(k):indicies(k+1)),Tw(indicies(k):indicies(k+1)),'FillType',[1,0],'FillColor',[0.
30,0.75,0.93],'FillAlpha',0.25);
            end
        end
        %Make only one shade object visible for legend documentation. Turn off
        %visibility for all other elements of teh shading.

shadePatches=findobj(tsax,'type','patch');shadeLines=findobj(tsax,'type','line');scCount=0;
        for k = 1:length(shadePatches)
            if isequal(shadePatches(k).FaceColor,[0.30,0.75,0.93])
                scCount = scCount+1;
```

```
                if scCount>1
                    shadePatches(k).HandleVisibility='off';
                end
            else
                shadePatches(k).HandleVisibility='off';
            end
        end
        for k = 1:length(shadeLines)
            shadeLines(k).HandleVisibility='off';shadeLines(k).Color='none';
        end
        else
           fprintf('No supercooling events recorded for %s.',TSTab.ID)
        end
    end
%Plot other plot components
%Water Temperature
if strcmp(tsType,'Deployments') && strcmp(TSTab.Filter,'off') && ~isnat(frz) %Show the filtered
out midwinter plot
    %Divide the data into 3 sets: before freeze-up end, between freeze-up end and break up start,
and after break-up
        r1 = find(t<=frz);r1=r1(end);r2 = find(t >= brk,1); %Indicies in the divides
         t1 = t(1:r1);Tw1=Tw(1:r1);t2 = t(r1+1:r2);Tw2=Tw(r1+1:r2);t3 =
t(r2+1:end);Tw3=Tw(r2+1:end);
    %Plot the water temperature from before freeze-up and after break-up as
    %a solid line
   hold on;p1=plot(t1,Tw1,'k-','LineWidth',2);p7=plot(t3,Tw3,'k-
','LineWidth',2,"HandleVisibility",'off');
    %Plot temperature between the two dates as dashed lines
    p6=plot(t2,Tw2,'--','LineWidth',2,'Color',[0.5 0.5 0.5]);
    legFilt = "Filtered Water Temperature";
else %All other cases
    hold on;p1=plot(t,Tw,'k-','LineWidth',2);grid on;
    p6=[];p7=[]; %These variables are not needed
    legFilt='';
end
%Sensor Accuracy
if strcmp(TSTab.Acc,'on')
    a = ones(size(t))*sensAcc;
   p2a = plot(t,a,'k:');p2b=plot(t,-a,'k:','HandleVisibility','off');
   legAcc="Sensor Accuracy";
else
    p2a=[];p2b=[];legAcc="";
end
%OC line
p3 = plot(t,zeros(length(t),1),'k--','HandleVisibility','off');
%Start & End Points
p4 =  scatter(t(indicies),Tw(indicies),'ko','filled');
if~isempty(indicies)
legStart="Start/End of Events";
else
 legStart='';
end
```

```
% Mark end of Freeze-Up and Start of Break-Up
if strcmp(TSTab.FreezeBreak,'on')
    if ~isnat(frz)
        F=plot([frz,frz],[-0.05 0.05],'b--');
        if tRange(1) <=frz && tRange(2)>=frz
            legFreeze = 'End of Freeze-Up/Start of Consolidated Cover';
        elseif xL(1) > frz || xL(2) < frz
            legFreeze='End of Freeze-Up/Start of Consolidated Cover (outside timeframe of Super-
Cooling Observations)';
        end
    else
        plot(NaT,NaN,'HandleVisibility','off')
        legFreeze = '';
    end
    if ~isnat(brk)
        B=plot([brk,brk],[-0.05 0.05],'r--');
        if tRange(2) >= brk && tRange(1) < brk
            legBreak = 'End of Consolidated Cover /Start of Break-Up';
        elseif xL(2) < startBreakUp || xL(1) > startBreakUp
            legBreak='End of Consolidated Cover /Start of Break-Up (outside timeframe of Super-
Cooling Observations)';
        end
    else
        plot(NaT,NaN,'HandleVisibility','off')
        legBreak='';
    end
else
  legFreeze = '';legBreak='';
end
%Negative Threshold
if strcmp(TSTab.Thresh,'on')
     p5=[];legThresh='';
else %Plot full range of negative water temperature (down to -1 C) and mark the negative
threshold on the plot
     p5 = plot(t,ones(length(t),1)*negThresh,'k','LineWidth',3);legThresh = 'Negative Threshold
of Analysis';
end

%CRP
if strcmp(TSTab.CRP,'on')
    %Storage array for the slope lines, primary peak supercooling mark, and
    %mark for data tip for cooling rate
    crpline=cell(height(evtsTab),1);Tp1mark = cell(height(evtsTab),1);
crpmk=cell(height(evtsTab),1);
    m = zeros(height(evtsTab),1);pdur=m;perdur=m; %Starage array for slope, principal
supercooling duration, and percent of total duration
   for k = 1:height(evtsTab)
       if k==1
          vis = 'on';
       else
           vis='off';
       end
```

```
%Extract values
       sct1 = evtsTab.Start_Time(k);sct2 = evtsTab.First_Time_of_Peak_Supercooling(k);Tw2 =
evtsTab.Peak_Supercooling(k);

m(k)=evtsTab.Principal_Supercooling_Average_Cooling_Rate(k);pdur(k)=evtsTab.Principal_Supercoolin
g_Duration(k);
       perdur(k)=evtsTab.Principal_Supercooling_Percent_of_Duration(k);
       %Create marker value for crp data tipe text
       tm = (sct2-sct1)/2;Twm = m(k)*minutes(tm);
       %Plot Slope Line and mark the peak supercooling tempearature
       crpline{k} = plot([sct1 sct2],[0 Tw2],'b-','HandleVisibility',vis); %slope line
       Tp1mark{k} = scatter(sct2,Tw2,'kd','filled','HandleVisibility',vis); %Marks the first
occurance of peak supercooling
       crpmk{k} = scatter(sct1+tm,Twm,'bo','filled','HandleVisibility','off'); %data tip marker
for slope information
   end
   legCRP='Principal Supercooling Average Cooling Rate';legTP='Peak Supercooling';
else
    m = [];pdur=m;perdur=m; %Empty arrays
    legCRP='';legTP='';
end

%Add data tip texts
%Water Temperature (p1, and potentially p6 & p7 as well)
if strcmp(tsType,'Deployments') && strcmp(TSTab.Filter,'off') && ~isnat(frz) %Additional
presentation required
    p1.DataTipTemplate.DataTipRows(1).Label = 'Date-Time: ';
    p1.DataTipTemplate.DataTipRows(2).Label = 'Water Temperature (deg C): ';
    str = strings(size(t1));str(:)='Freeze-Up';row1 = dataTipTextRow('River Ice Process: ',str);
    str(:)=TSTab.ID;row2=dataTipTextRow('Time-Series ID: ',str);
    p1.DataTipTemplate.DataTipRows(end+1)=row2;

    if~isempty(p6)
        p6.DataTipTemplate.DataTipRows(1).Label = 'Date-Time: ';
        p6.DataTipTemplate.DataTipRows(2).Label = 'Water Temperature (deg C): ';
        str = strings(size(t2));str(:)='Consolidated Cover';row = dataTipTextRow('River Ice
Process: ',str);
        p6.DataTipTemplate.DataTipRows(end+1)=row;
        str(:)=TSTab.ID;row2=dataTipTextRow('Time-Series ID: ',str);
        p6.DataTipTemplate.DataTipRows(end+1)=row2;
    end
    if~isempty(p7)
        p7.DataTipTemplate.DataTipRows(1).Label = 'Date-Time: ';
        p7.DataTipTemplate.DataTipRows(2).Label = 'Water Temperature (deg C): ';
        str = strings(size(t3));str(:)='Break-Up';row = dataTipTextRow('River Ice Process:
',str);
        p7.DataTipTemplate.DataTipRows(end+1)=row;
        str(:)=TSTab.ID;row2=dataTipTextRow('Time-Series ID: ',str);
        p7.DataTipTemplate.DataTipRows(end+1)=row2;
    end
```

```
else %All other cases
    p1.DataTipTemplate.DataTipRows(1).Label = 'Date-Time: ';
    p1.DataTipTemplate.DataTipRows(2).Label = 'Water Temperature (deg C): ';
    str = strings(size(t));
    if~isnat(frz)
        str(t<=frz)='Freeze-Up';str(t>=brk);
    else
        str(:)='Freeze-Up';
    end
    row = dataTipTextRow('River Ice Process: ',str);
    p1.DataTipTemplate.DataTipRows(end+1)=row;
    str(:)=TSTab.ID;row2=dataTipTextRow('Time-Series ID: ',str);
    p1.DataTipTemplate.DataTipRows(end+1)=row2;
end
%OC line (p3)
p3.DataTipTemplate.DataTipRows(1).Label = 'Date-Time: ';
p3.DataTipTemplate.DataTipRows(2).Label = 'Water Temperature (deg C): ';
%Sensor Accuracy (p2)
if strcmp(TSTab.Acc,'on')
    p2a.DataTipTemplate.DataTipRows(1).Label = 'Date-Time: ';
    p2a.DataTipTemplate.DataTipRows(2).Label = 'Sensor Accuracy (deg C): ';
    p2b.DataTipTemplate.DataTipRows(1).Label = 'Date-Time: ';
    p2b.DataTipTemplate.DataTipRows(2).Label = 'Sensor Accuracy (deg C): ';
end

%Start and End Times of Events
%Create String Array for Start/End Labels and Period Classification
eventLabel=strings([2,height(evtsTab)]);classLabel=eventLabel;seasLabel=eventLabel; %Storage
array of labels
for k = 1:height(evtsTab)
    eventLabel(:,k) = evtsTab.Event_ID(k);
    classLabel(:,k) = evtsTab.River_Ice_Process(k);
    seasLabel(:,k) = evtsTab.Season(k);
end
%Reshape eventLabel and classLabel to be read by the datatiptext, then
%add datatiptext to figures
eventLabel=reshape(eventLabel,1,[]);classLabel=reshape(classLabel,1,[]);seasLabel=reshape(seasLab
el,1,[]);
%Label Start/End of Event Markers
for k = 1:length(p4)
    ddt=p4.DataTipTemplate;
    ddt.DataTipRows(1).Label = 'Date-Time';
    ddt.DataTipRows(2).Label = 'Water Temperature (deg C): ';
    row = dataTipTextRow('Season: ',seasLabel);
    ddt.DataTipRows(end+1) = row;
    row = dataTipTextRow('Event ID: ',eventLabel);
    ddt.DataTipRows(end+1)=row;
    row = dataTipTextRow('River Ice Process: ',classLabel);
    ddt.DataTipRows(end+1) = row;
end
```

```
%Freeze-Up & Break-Up (F & B)
if strcmp(TSTab.FreezeBreak,'on')
    if ~isnat(frz)
        ddtF=F.DataTipTemplate;ddtF.DataTipRows(1).Label = 'Date-Time';
        periodRow = dataTipTextRow('River Ice Process Boundary:',{'Freeze-Up | Consolidated
Cover','Freeze-Up | Consolidated Cover'},'auto');
        ddtF.DataTipRows(2) = periodRow;
    end

    if ~isnat(brk)
        ddtB=B.DataTipTemplate;ddtB.DataTipRows(1).Label = 'Date-Time';
        periodRow = dataTipTextRow('River Ice Process Boundary:',{'Consolidated Cover | Break-
Up','Consolidated Cover  | Break-Up'},'auto');
        ddtB.DataTipRows(2) = periodRow;
    end
end
%Negative Threshold (p5)
if strcmp(TSTab.Thresh,'off')
    p5.DataTipTemplate.DataTipRows(1).Label = 'Date-Time: ';
    p5.DataTipTemplate.DataTipRows(2).Label = 'Negative Threshold (deg C): ';
end

%CRP (crpline & Tp1mark)
if strcmp(TSTab.CRP,'on')
    for k = 1:height(evtsTab)
        crpline{k}.DataTipTemplate.DataTipRows(1).Label='Date-Time: ';
        crpline{k}.DataTipTemplate.DataTipRows(2).Label='Water Temperature (deg C): ';
        Tp1mark{k}.DataTipTemplate.DataTipRows(1).Label = 'End of Principal Supercooling: ';
        Tp1mark{k}.DataTipTemplate.DataTipRows(2).Label = 'Peak Supercooling Temperature (deg C):
';
        ddt=crpmk{k}.DataTipTemplate; %Get the data-tip template for the marker
        ddt.DataTipRows(1).Label = "Principal Supercooling Mid-Point: ";ddt.DataTipRows(2).Label
= "Principal Supercooling Mid-Point Temperature (deg C): ";
        ddt.DataTipRows(end+1).Label = "Principal Supercooling Average Cooling Rate (deg
C/minute): ";ddt.DataTipRows(end+1).Value = m(k);
        ddt.DataTipRows(end+1).Label = "Principal Supercooling Duration (hours):
";ddt.DataTipRows(end+1).Value = pdur(k);
        ddt.DataTipRows(end+1).Label = "Principal Supercooling Percent of Event (%):
";ddt.DataTipRows(end+1).Value = perdur(k);
    end
end

%Set xlimits
xlim(tRange);
%Set ylimits
%Determine the range of water temperatures in the xlimits
range1=t>=tRange(1);range2=t<=tRange(2);
rangeindex=and(range1,range2); %Times that satisfy both range1 and range2 are 1.
range = find(rangeindex == 1); %Only elegible indicies are left
```

```
%Set Ylimits based on status of Negative Threshold
if strcmp(TSTab.Thresh,'on')
    ylim([max(-0.2,min(Tw(range))*1.5)...
        min(0.2,max(1.5*(Tw(range))))]); %A rough y limits that keeps the minimum above the
negative threshold
else %Plot full range of negative water temperature (down to -1 C) and mark the negative
threshold on the plot
    ylim([max(-1,min(Tw(indicies(1):indicies(end)))*1.5)...
        min(1,max(1.5*(Tw(indicies))))]); %A rough y limits that keeps the minimum above the
negative threshold
end
%Set labels
xlabel('t');ylabel('T_{w} (^{o}C)');
title(TSTab.Title);grid(g);
%Set legend icon if asked for
if strcmp(TSTab.Leg,'on')
    %Create a list of Legend components
    legstr = ["Recorded Supercooling","Water
Temperature",legFilt,legAcc,legStart,legFreeze,legBreak,legThresh,legCRP,legTP];
    %Delete all Empty elements of the string
    legstr(legstr=="")=[];
    %Add legend to axes
    legend(legstr,"Location",'best');
end
else %skip this period
    warning('There is no time series to plot');f=[];
end
end

%Shade
function h = shade(varargin)
%SHADE  Filled area linear plot.
%
%   SHADE should be called using the same syntax as the built-in PLOT.
%
%   SHADE(X,Y) plots vector Y versus vector X, filling the area under the
%   curve. If either is a matrix, this function behaves like PLOT.
%
%   SHADE(Y) plots the columns of Y versus their index.
%
%   SHADE(X,Y,S) plots Y versus X using the line type, marker symbols and
%   colors as specified by S. For more information on the line specifier S,
%   see PLOT.
%
%   SHADE(X1,Y1,X2,Y2,X3,Y3,...) combines the plots defined by
%   the (X,Y) pairs.
%
%   SHADE(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...) combines the plots defined by
%   the (X,Y,S) triples.
%
%   SHADE(AX,...) plots into the axes with handle AX.
%
%   H = SHADE(...) returns a column vector of handles to graphics objects.
%
```

```
%   SHADE(...,Name,Value) specifies additional properties of the lines (see
%   help for PLOT) or the filled areas (see below).
%
%   Three additional properties are provided to control the filling:
%
%     - 'FillType' specifies the filling behaviour. The input should be a
%       matrix of size [N,2], such as [A1,B1;A2,B2;...;An,Bn], where Ai and
%       Bi indicate the upper and lower limits, respectively, of each of
%       the N areas to be filled. Each Ai and Bi is an index pointing to
%       one of the lines drawn by PLOT. If, for a particular combination of
%       inputs, PLOT draws M lines, then each Ai and Bi should be a number
%       between 1 and M. In addition, the special cases 0, -1 and -2 are
%       allowed, each representing the x-axis, the bottom of the active
%       axes and the top of the active axes, respectively. 'FillType' may
%       also be specified using a cell array of size [N,2], in which case
%       one may also use the keywords 'axis', 'bottom' and 'top' instead
%       of 0, -1 and -2. By default, the areas between each of the curves
%       and the x-axis are filled, which corresponds to the input matrix
%       [1,0;0,1;2,0;0,2;...;M,0;0,M].
%
%     - 'FillColor' specifies the color of the fillings. This should be a
%       matrix of size [N,3], where each row is the RGB triplet for the
%       corresponding area as specified above. 'FillColor' may also be
%       specified as a cell array of length N, in which case each entry may
%       be either an RGB triplet or any of the color names commonly used in
%       MATLAB. If only one RGB value or color name is provided, all areas
%       are treated equally. If this parameter is not specified, colors are
%       determined by the corresponding lines.
%
%     - 'FillAlpha' specifies the transparency of the fillings. This should
%       be a vector of length N, where each entry specifies the alpha value
%       for each area. If only one alpha value is provided, all areas are
%       treated equally. If this parameter is not specified, an alpha value
%       of 0.3 is used for all areas.
%
%   See also PLOT.

% Copyright (c) 2018 Javier Montalt Tordera.

% accepted params
names = {'FillType','FillColor','FillAlpha'};

% init fill params
fp = cell(1,3);
```

```
% extract filling parameters, if present
for n = 1:length(names)
    for i = 1:length(varargin)
        % if found
        if strcmpi(names{n},varargin{i})
            if i+1 > nargin
                error(["Expected an input value after the name '" names{i} "'."]);
            end
            % save filling info
```

```
            fp{n} = varargin{i+1};
            % delete from varargin array - otherwise plot will fail as it won't
            % understand the input
            varargin(i:i+1) = [];
            break;
        end
    end
end

% check if an axes object was specified
if isscalar(varargin{1}) && ishandle(varargin{1}(1))
    ax = varargin{1};
else
    ax = gca;
end
% initial hold state
tf = ishold(ax);
% plot lines
ls = plot(varargin{:});
hold(ax,'on');
% provide default filling params
fd = fp{1};
fc = fp{2};
fa = fp{3};
% validate fill type
fd = validatetype(fd,ls);
% number of fillings
nf = size(fd,1);
% validate fill color
fc = validatecolor(fc,fd,ls,nf);
% validate fill alpha
fa = validatealpha(fa,nf);
% array to hold patch objects
ps = gobjects(nf,1);
```

```
% for each filling
for i = 1:nf
    x = cell(1,2);
    y = cell(1,2);
    % get data
    for j = 1:2
        switch fd(i,j)
            case {-2,-1}
                y{j} = ylim;
                y{j} = y{j}(abs(fd(i,j)));
            case 0
                y{j} = 0;
            otherwise
                x{j} = ls(fd(i,j)).XData;
                y{j} = ls(fd(i,j)).YData;
        end
    end

    if isequal(x{1},x{2})
```

```
        x = x{1};
    elseif isempty(x{1})
        x = x{2};
        y{1} = y{1} * ones(size(x));
    elseif isempty(x{2})
        x = x{1};
        y{2} = y{2} * ones(size(x));
    else
        x = sort([x{1} x{2}]);
        y{1} = interp1(x{1},y{1},x,'linear','extrap');
        y{2} = interp1(x{2},y{2},x,'linear','extrap');
    end
    % crossings
    x0 = [x(1) zcross(x,y{1} - y{2}) x(end)];
    y0 = interp1(x,y{1},x0);
    % for each zero-crossing
    for j = 0:length(x0)-2
        % index
        idx = x >= x0(j+1) & x <= x0(j+2) & y{1} >= y{2};
        if all(~idx), continue; end
        % polygon corners
        xv = [x0(j+1) x(idx) x0(j+2) fliplr(x(idx))];
        yv = [y0(j+1) y{1}(idx) y0(j+2) fliplr(y{2}(idx))];
        % fill polygon
        ps(i) = fill(ax,xv,yv,fc(i,:),'LineStyle','none','FaceAlpha',fa(i));
    end
end
% release if the hold state was off when called
if tf == 0
    hold(ax,'off');
end
```

```
% set output argument if requested
if nargout == 1
    h = [ls;ps];
end

end
function z = zcross(x,y)
% find zero crossings of line Y versus X

% logical index
c = y > 0;

% find point pairs where there is a sign change
d = abs(diff(c));
p1 = find(d == 1);  % before change
p2 = p1 + 1;        % after change

% zero-crossing positions
z = x(p1) + abs(y(p1)) ./ (abs(y(p1)) + abs(y(p2))) .* (x(p2) - x(p1));
end
function fd = validatetype(fd,ls)
% if no filling specified, fill to x-axis
```

```
if isempty(fd)
    fd = [(1:length(ls))' zeros(size(ls))];
    fd = [fd;fliplr(fd)];
    fd = fd([1:2:length(fd) 2:2:length(fd)],:);
end
% if the filling was specified in cell form, convert to matrix
if iscell(fd)
    tmp = zeros(size(fd));
    for i = 1:numel(fd)
        if ischar(fd{i})
            tmp(i) = str2ind(validatestring(fd{i},{'axis','bottom','top'},'shade','FillType'));
        else
            validateattributes(fd{i},{'numeric'},{'scalar'},'shade','FillType');
            tmp(i) = fd{i};
        end
    end
    fd = tmp;
end
validateattributes(fd,{'numeric'},{'integer','size',[nan 2],'>=',-
2,'<=',length(ls)},'shade','FillType');
end
function fc = validatecolor(fc,fd,ls,nf)
% if no color specified, get it from plot lines
if isempty(fc)
    fc = zeros(nf,3);
    for i = 1:nf
        if fd(i,1) <= 0
            fc(i,:) = ls(fd(i,2)).Color;
        elseif fd(i,2) <= 0
            fc(i,:) = ls(fd(i,1)).Color;
        else
            fc(i,:) = mean([ls(fd(i,1)).Color;ls(fd(i,2)).Color],1);
        end
    end
end

% if length 1, repeat
if ischar(fc)
    fc = {fc};
    fc = repmat(fc,nf,1);
elseif (iscell(fc) && numel(fc) == 1) || (~iscell(fc) && size(fc,1) == 1)
    fc = repmat(fc,nf,1);
end

% if color is specified in cell form, convert to matrix
if iscell(fc)
    validateattributes(fc,{'cell'},{'vector','numel',nf},'shade','FillColor');
    tmp = zeros(numel(fc),3);
    for i = 1:numel(fc)
        if ischar(fc{i})
            tmp(i,:) =
str2rgb(validatestring(fc{i},{'y','m','c','r','g','b','w','k','yellow','magenta','cyan','red','gr
een','blue','white','black'},'shade','FillColor'));
        else
```

```
            validateattributes(fc{i},{'numeric'},{'vector','numel',3},'shade','FillColor');
            if iscolumn(fc{i})
                fc{i} = fc{i}';
            end
            tmp(i,:) = fc{i};
        end
    end
    fc = tmp;
end
validateattributes(fc,{'numeric'},{'real','size',[nf 3],'>=',0,'<=',1},'shade','FillColor');
end

function fa = validatealpha(fa,nf)
% if no alpha specified, choose a value of 0.2
if isempty(fa)
    fa = 0.3 * ones(nf,1);
end
% if length 1, repeat
if length(fa) == 1
    fa = repmat(fa,nf,1);
end

if isrow(fa)
    fa = fa';
end

validateattributes(fa,{'numeric'},{'real','vector','numel',nf,'>=',0,'<=',1},'shade','FillAlpha')
;
end

function n = str2ind(s)
% convert string to index
switch s
    case 'axis'
        n = 0;
    case 'bottom'
        n = -1;
    case 'top'
        n = -2;
end
end

function n = str2rgb(s)
% convert string to RBG triplet
switch s
    case {'y','yellow'}
        n = [1 1 0];
    case {'m','magenta'}
        n = [1 0 1];
    case {'c','cyan'}
        n = [0 1 1];
    case {'r','red'}
        n = [1 0 0];
    case {'g','green'}
```

```
        n = [0 1 0];
    case {'b','blue'}
        n = [0 0 1];
    case {'w','white'}
        n = [1 1 1];
    case {'k','black'}
        n = [0 0 0];
end
end
```

```
function [array] = onoffArray(array,inpts,name,idlist)
%Standard procedure for setting 'on' or 'off' values for the various
%plotting parameters

errchk=0; %Assume no warning needs to be given regarding invalid inputs
default = unique(array); % before inputs are added, only one vaue type in the array in this array
if idlist || (~idlist && length(inpts)>1)
    for k = 1:length(inpts)
        try
            if strcmpi(inpts(k),'on')
                array(k) = 'on';
            elseif strcmpi(inpts(k),'off')
                array(k) = 'off';
            else
                errchk = 1;
            end
        catch
            errchk=1;continue
        end
    end
else %No IDlist, you can convert single length inputs to the full array switch
    if strcmpi(inpts,'on')
        array(:) = 'on';
    elseif strcmpi(inpts,'off')
        array(:) = 'off';
    else
        errchk = 1;
    end
end
if errchk
    warning("The only permissible inputs for %s are 'on' or 'off'. If invalid values were
submitted, the default value of '%s' is assumed.",name,default)
end
end

function varglength(k,inpts)
if k == length(inpts)
   error("'%s' is not followed by a value to assign the parameter",inpts{k})
end
end
```

Published with MATLAB® R2021a

```
function [T]=StatTable(Table,tableColumns)
%StatTable generates a statistics table for a sbmitted table of events
%using the submitted Variable Names (tableColumns). This function checks
%that the inputted table column exists & that the data in the column is
%quantatative (can be statistically analyzed). Inputs that fail this
%criterion are omitted.
%
%INPUT
%  Table = sumitted table object
%  tableColumns = Column headings of submitted table to be statistically
%  summarized
%
```

```
%OUTPUT
% T = Statistics summary (min, median, mean, maximum, standard deviation,
% and number of data points) used for each column analyzed.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Author: Sean R. Boyd January 19th, 2022
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
narginchk(1,2);format short
%Verify that a table was submitted
if ~isa(Table,'table')
    error("'Table' must be a table object")
end
if nargin == 1 %Analysis all table columns

    tableColumns = string([Table.Properties.VariableNames]);
else
%Verify column names
    if ~isa(tableColumns,'string')
        try
            tableColumns = string(tableColumns);
        catch
            error("'tableColumns' cannot be converted into a string")
        end
    end
    %Verify that the tablecolumns exist
    colChk = string([Table.Properties.VariableNames]);del = zeros(size(tableColumns));
    for k = 1:length(tableColumns)
        if  isempty(find(colChk == tableColumns(k),1))
            del(k)=k;
        end
    end
    del(del == 0)=[];tableColumns(del)=[];%Deleter invalid column names
    if ~isempty(del) && ~isempty(tableColumns) %Some table columns were removed
        fprintf('Invalid column names were ignored.\n')
    elseif isempty(tableColumns)
        error('Submitted column names were invalid.');
    end
end
```

```
%Filter out columns with the incorrect data type
del = zeros(size(tableColumns));
```

```
for k = 1:length(tableColumns)
    if ~isa(Table.(tableColumns(k)),'double')
        del(k)=k;
    end
end
del(del==0)=[];
if~isempty(del)
    tableColumns(del)=[];
    fprintf('Removed qualatative data (cannot be analysed statistically by this function.\n')
```

```
    if isempty(tableColumns)
        fprintf('None of the submitted table columns could be statistically analysed by this
function.\n')
        clearvars T;pause(1);clc
    end
end
    vartypes = strings(size(tableColumns));vartypes(:)='double';
    T =
table('Size',[6,length(tableColumns)],'VariableTypes',vartypes,'RowNames',{'Minimum','Median','Me
an','Maximum','Standard_Deviation','Number_of_Events'},...
        'Variablenames',tableColumns);
    units=strings(size(tableColumns));
    for k = 1:length(tableColumns)
        data = Table.(tableColumns(k));
        data(isnan(data))=[]; %Remove NaN values
        units(k) = Table.Properties.VariableUnits(tableColumns(k));
        T{1,k} = min(data); T{3,k} = mean(data);T{2,k} = median(data); T{4,k} = max(data);
        T{5,k} = std(data,'omitnan'); %Standard Deviation (omit NaN values)
        T{6,k}=length(data);%Number of events in set
    end
    T.Properties.VariableUnits = units;
end
```

Published with MATLAB® R2021a

B-4 Energy budget analysis

B-4.1 Summary of functions

The program presented for the energy budget analysis, *Surfengbudgt(),* is a streamlined version of the energy budget analysis codes used for Chapter 3. The refinement include removal of initial analysis codes that were ignored as the scope of the study was finalized, as well as streamline and generalization that makes the code easier to apply to new scenarios. The function takes inputs of a season data table and a summary table of supercooling events (though only the start, end, and time of peak supercooling temperature are required) and calculates the surface energy budget using the same method as used in Chapter 3. The other programs developed for this study, *MassSurfengbudgt()* and *massTable()* automated the application of *Surfengbudgt()* to all deployments, and compiled event summary tables for analysis. Table B-3 summarizes the programs along with reference pages.

Table B. 3: Summary of energy budget analysis programs. Page indicates the starting page of the copied MATLAB code.

Program	Summary of Function	Page
Surfengbudgt ()	Calculates the surface energy budget for the given data table, then analyzes the energy budget during supercooling events specified in the submitted supercooling event summary table for parameters of interest.	210
MassSurfengbudgt ()	Applies *Surfengbudgt ()* to all data table and supercooling event summary table pairs in a submitted data structure	216
massTable ()	Compiles all the supercooling event summary tables into a single table. Can filter for specific river.	216

B-4.2 Program code

```
function [dataTable,supercoolingeventTable] = Surfengbudgt(dataTable,supercoolingeventTable)
%Summary: Surfengbudgt takes the dataTable containing water temperature
%and local weather parameters and calculates the heatfluxes from those
%parameters. If a supercoolingeventTable, the start time, end time, and
%time of first peak supercooling temperature are used to compute average
%and extreme temperatures.
%
%INPUT
%   dataTable: The dataTable is a time table that contains the water
%   temperature and weather parameters to be used in calculating heat
%   fluxes.
%   supercoolingeventTable: Summary table of supercooling events that
%   occured during the timeframe of the datatable. Note that dataTable must
%   have exact timing of the start and end time of supercooling events
%   (thus interpolated OC values already present). Any events whose start
%   and end time are not found in the dataTable will not have additional
%   parameters computed.
%
%DATATABLE FORMATTING
%In order for the function to read and calculate heat fluxes, the dataTable
%needs a specific formatting:

%HEADINGS
%   Actual Vapour Pressure [units: mb]: Act_Vapour_Press
%   Air Temperature [units: deg C]: Air_Temperature
%   Barometric Pressure [units: mb]: Baro_Pressure
%   Cloud Cover Fraction [units decimal fraction]: Cloud_Cover_Fraction
%   Ice Concentration [units: decimal fraction]: Surface_Ice_Conc
%   Relative Humidity [units: percentage]: Relative_Humidity
%   Saturated Vapour Pressure [units: mb]: Sat_Vapour_Press
%   Solar Radiation [units: W/m^2]: Solar_Radiation
%   Water Temperature [units: deg C]: Water_Temperature
%   Wind Speed [units: m/s]: Wind_Speed
%
%   All heat Fluxes: [units: W/m^2] - The input table does not need columns
%   for the heat fluxes, as they are added during the calculations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Author: Sean R. Boyd (2022)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Set required unit strings
udT = ["degree C","degree C/minute","degree C","W/m2","fraction","fraction",...
    "mb","%","mb","mb","m/s","W/m^2","W/m^2","W/m^2","W/m^2","W/m^2"];
uEvt = ["","date-time","date-time","hours","degree C","date-time","date-time","hours"...
    "degree C/minute","hours","%","degree C*minute","","","degree C","degree
C","fraction","m/s",...
    "%","mb","W/m^2","W/m^2","W/m^2","W/m^2","W/m^2","W/m^2","W/m^2","W/m^2",...
    "W/m^2","W/m^2","W/m^2","W/m^2","W/m^2","W/m^2","W/m^2","W/m^2","W/m^2","W/m^2",...
    "W/m^2","W/m^2","W/m^2","W/m^2","W/m^2","W/m^2","W/m^2","J/m^2",...
    "","fraction","","fraction","","fraction","","fraction"];
```

```
%Reference data outside table for convenience
cloud = dataTable.Cloud_Cover_Fraction;
conc = dataTable.Surface_Ice_Conc;
sol=dataTable.Solar_Radiation;
airT = dataTable.Air_Temperature;
watT =dataTable.Water_Temperature;
RelHum = dataTable.Relative_Humidity;
Wind = dataTable.Wind_Speed;
Press = dataTable.Baro_Pressure;

%Calculate Saturated Vapour Pressure and Actual Vapour Pressure
[satvapress, actvapress] = vapourpressure(airT,conc,RelHum);
dataTable.Sat_Vapour_Press=satvapress;
dataTable.Act_Vapour_Press=actvapress;

%Determine Shortwave Radiation Heat Flux
[shortwave] = shortwaveHeatFlux(conc,sol);
dataTable.Shortwave_HeatFlux=shortwave;

%Determine Longwave Heat Flux
[longwave] = longwaveHeatFlux(cloud,airT,watT,conc);
dataTable.Longwave_HeatFlux=longwave;


%Calculate Evaporative Heat Flux
[evaporative, sensible] = evapsenseHeatFlux(airT,watT,Wind,Press,satvapress,actvapress,conc);
dataTable.Evaporative_HeatFlux=evaporative;
dataTable.Sensible_HeatFlux=sensible;

%Determine Net Heat Flux (Sum of all Net heat fluxes)
netFlux = shortwave + longwave + evaporative + sensible;
dataTable.Net_HeatFlux=netFlux;

%Compare the datatable to the supercooling events and compute the heat flux
%behaviour during supercooling

for event = 1: height(supercoolingeventTable) %[START HERE]
    %Get start time, end of principal supercooling, and end time
    startTime = supercoolingeventTable.Start_Time(event);
    princTime = supercoolingeventTable.First_Time_of_Peak_Supercooling(event);
    endTime = supercoolingeventTable.End_Time(event);
    %Determine the rows of the season data table that align with these
    %times in the supercooling event
    startRow = find(dataTable.Time == startTime);
    pricRow = find(dataTable.Time == princTime);
    endRow = find(dataTable.Time == endTime);

    %Add average weather conditions to table
    supercoolingeventTable.Average_Water_Temperature(event) = mean(watT(startRow:endRow));
    supercoolingeventTable.Average_Air_Temperature(event) = mean(airT(startRow:endRow));
    supercoolingeventTable.Average_Cloud_Cover(event) = mean(cloud(startRow:endRow));
    supercoolingeventTable.Average_Wind_Speed(event) = mean(Wind(startRow:endRow));
    supercoolingeventTable.Average_Relative_Humidity(event) = mean(RelHum(startRow:endRow));
    supercoolingeventTable.Average_Baro_Press(event) = mean(Press(startRow:endRow));
```

```
    %Determine Average value of the Heat Flux component for the
    %event [START HERE}
    flux = ["Shortwave","Longwave","Sensible","Evaporative","Net"];
    for f = 1:length(flux)
        [supercoolingeventTable] =
addFluxStat(supercoolingeventTable,event,dataTable,startRow,endRow,pricRow,flux(f));
    end
    %Determine cumulative Net Heat Flux (net energy)
    Timesecs = seconds(dataTable.Time(startRow:endRow)-dataTable.Time(startRow));

supercoolingeventTable.Net_Energy(event)=trapz(Timesecs,dataTable.Net_HeatFlux(startRow:endRow));
    %Determine the strictly negative and positive event averaged heat flux (ie.
    %the summation outputs the event averaged net heat flux)
    ttevt = dataTable(startRow:endRow,:);ttprincp = dataTable(startRow:pricRow,:);
    %Average of strictly negative heat fluxes
     %Get mean values of the total negative heat flux for event and principal supercooling
    [meantot] = meantotFlux(ttevt,-1);[meanprcp] = meantotFlux(ttprincp,-1);
    %Determine the dominant negative heat flux and the fraction of the total
    %negative component
    [supercoolingeventTable] = dominantFlux(supercoolingeventTable,event,'Principal',meanprcp);
    [supercoolingeventTable] = dominantFlux(supercoolingeventTable,event,'Event',meantot);
    %Average of strictly positive heat fluxes
     %Get mean values of the total positive heat flux for event and principal supercooling
    [meantot] = meantotFlux(ttevt,1);[meanprcp] = meantotFlux(ttprincp,1);
    %Determine the dominant positive heat flux and the fraction of the total
    %positive component
    [supercoolingeventTable] = dominantFlux(supercoolingeventTable,event,'Principal',meanprcp);
    [supercoolingeventTable] = dominantFlux(supercoolingeventTable,event,'Event',meantot);
end
%Assign units
dataTable.Properties.VariableUnits = udT;
supercoolingeventTable.Properties.VariableUnits = uEvt;
end
%SUB-FUNCTIONS

%Saturated and Actual Vapour Pressure
function [satvapress, actvapress] =
vapourpressure(airTemperature,iceConcentration,percentRelHumidity)
%SUMMARY: Determines the weighted average for saturated and actual vapour
%pressure over a water surface based on surface ice concentration.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Determine weighting factors for the open water case
openwatFactor = ones(size(iceConcentration)) - iceConcentration;
%Determine the vapour pressure cases
numor = 17.62*airTemperature;denom = (243.12+airTemperature); %numorator and denominator of the
following equation
openwatvap = exp(numor./denom);
numor = 22.46*airTemperature;denom = (272.62+airTemperature); %numorator and denominator of the
following equation
icesurfvap = exp(numor./denom);
```

```
%Calculate weighted average of vapor pressure (units mb)
satvapress = 6.11*(openwatFactor.*openwatvap + iceConcentration.*icesurfvap);
actvapress = (percentRelHumidity/100).*satvapress;
end

%Shortwave Heat Flux
function [shortwave] = shortwaveHeatFlux(iceConcentration,solarRadiation)
%Determine shortwave heat flux
shortwave = (ones(size(solarRadiation)) - iceConcentration)*0.9.*solarRadiation;
end

%Longwave Heat Flux
function [longwave] =
longwaveHeatFlux(cloudFactor,airTemperature,waterTemperature,iceConcentration)
%Source: Richard, Morse, and Daly (2015)
n3 = cloudFactor.^3; %Square the cloud factor for the emmissivity term
air4 = (airTemperature+273.15).^4;wat4 = (waterTemperature+273.15).^4; %4th power of temperature
in Kelvin
emiss = (0.22*n3+0.765); %Cloud influenced emissivity of air
longwave = 5.5*10^(-8)*(emiss.*air4-wat4).*(1-iceConcentration);
end

%Evaporative and Sensible Heat Flux
function [evaporative, sensible] =
evapsenseHeatFlux(airTemperature,waterTemperature,windSpeed,baroPress,saturatedVapress,actVapress
,iceConcentration)
% Calculates the evaporative and sensible heat fluxes using equations
% originating from Ryan et al. (1974)
%Source: Ashton, G. D. (2013). "Thermal Processes." River Ice Formation, S. Beltaos, ed.,
%Committee on River Ice Processes and the Environment, CGU-HS, Edmonton, Alberta, 19-76.
%Note that the equation for the sensible heat flux does not use the
%approximation of the 0.46*P/760 ~0.6 for the Bowen ratio between
%evaporative and sensible heat flux that Ashton (2013) does, but instead
%uses the full term to derive the sensible heat flux equation. If one uses
%the value of baroPressure to generate the the coefficent approximation of
%0.6 (~991.304348), the coefficents are the same coefficents as calculated by Ashton (2013)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Calculate the difference needed for the equations
tempDiff = waterTemperature - airTemperature; %Difference between water and air temperature
vapdiff = (saturatedVapress - actVapress); %Difference in saturated and actual vapour pressure
%Calculate the Virtual temperatures
Ta = virtualTemp(airTemperature,actVapress,baroPress); %Air virtual temperature
Tw = virtualTemp(waterTemperature,saturatedVapress,baroPress); %Water virtual temperature
dTv = Tw-Ta;rootdTv = nthroot(dTv,3);%The cube root on the difference in virtual temperatures

%Calculate evaporative and sensible heat fluxes
evaporative = -(1-iceConcentration).*(2.70*rootdTv + 3.2*windSpeed).*vapdiff;
sensible = -(1-iceConcentration).*(1.634*rootdTv+1.937*windSpeed).*tempDiff.*baroPress/1000;
```

```
%Subfunction: Virtual Temperature
    function [Tv] = virtualTemp(actTemp,vapPress,baroPress)
        %Convert input temperature (in degree C) to a virtual temperature
        %(in Kelvin) dependend on the ratio between the vapour and
        %atmospheric pressure.
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %Convert temperature to Kelvin
        T = actTemp+274.15;
        %Determine the ratio between vpour pressure and atmospheric
        %pressure
        r = vapPress./baroPress;
        %Calculate denominator and then virtual temperature
        d = -0.378*r+1;Tv = T./d;
    end
end

%Assign Flux Statistics
function [supercoolingeventTable] =
addFluxStat(supercoolingeventTable,event,dataTable,startRow,endRow,princpRow,flux)
ttevt = dataTable(startRow:endRow,:); %subset of time series data
ttpricp = dataTable(startRow:princpRow,:); %subset of time series data
col = sprintf('%s_HeatFlux',flux);%Column header of datatable
%Colums of the supercooling event table
princpCol = sprintf('Average_Principal_%s',col);
minCol = sprintf('Minimum_%s',col);
avgCol = sprintf('Mean_%s',col);
maxCol = sprintf('Maximum_%s',col);
devCol = sprintf('Std_Dev_%s',col);
%Calculate the relavent parameter and add to event row
supercoolingeventTable.(princpCol)(event) = mean(ttpricp.(col));
supercoolingeventTable.(minCol)(event) = min(ttevt.(col));
supercoolingeventTable.(avgCol)(event) = mean(ttevt.(col));
supercoolingeventTable.(maxCol)(event) = max(ttevt.(col));
supercoolingeventTable.(devCol)(event) = std(ttevt.(col));
end

%Calculate Mean Total Flux (the mean value of strictly positive or negative
%fluxes in an event)
function [meanval] = meantotFlux(tt,sign)
%Get heat flux components
short = tt.Shortwave_HeatFlux;long = tt.Longwave_HeatFlux;
sens = tt.Sensible_HeatFlux;evap = tt.Evaporative_HeatFlux;
flux = {short,long,sens,evap}; %Storage array
%Determine the mean value dependent on sign
meanval=0;
for k = 1:length(flux)
    comp = flux{k}; %Get Flux array
    switch sign
        case -1
            f = comp(comp<=0);
        case 1
            f = comp(comp>0);
    end
```

```
    if ~isempty(f)
        meanval = meanval + mean(f);
    else
        continue
    end
end
end

%Flux Dominance [Move the dominance analysis to this subfunction] [FIGURE
%THIS OUT}
function [supercoolingeventTable] = dominantFlux(supercoolingeventTable,event,col,meanvalue)
%Set up the array of values to be compared
switch col
    case "Principal"
        array = supercoolingeventTable{event,[21,26,31,36]};
    case "Event"
        array = supercoolingeventTable{event,[23,28,33,38]};
end
if meanvalue<0 %Negative heat flux
    col1 = sprintf('Dominant_Negative_%s_Averaged_HeatFlux',col);
    col2 = sprintf('Fraction_Negative_%s_Averaged_HeatFlux',col);
    m = min(array);index = find(array == m,1);
else %Positive heat flux
    col1 = sprintf('Dominant_Positive_%s_Averaged_HeatFlux',col);
    col2 = sprintf('Fraction_Positive_%s_Averaged_HeatFlux',col);
    m = max(array);index = find(array == m,1);
end
switch index
    case 1
        supercoolingeventTable.(col1)(event) = "Shortwave";
    case 2
        supercoolingeventTable.(col1)(event) = "Longwave";
    case 3
        supercoolingeventTable.(col1)(event) = "Sensible";
    case 4
        supercoolingeventTable.(col1)(event) = "Evaporative";
end
supercoolingeventTable.(col2)(event) = m/meanvalue;
end
%REFERENCES
%Ashton, G. 2013, Thermal Processes, in River Ice Formation, Committee on River Ice Processes
%and the Environment Canadian Geophysical Union, Hydrology Section,
%Edmonton, Alberta, Canada, ISBN 978-0-9920022-0-6
%
%Hicks, Faye (2016), An Introduction to River Ice Engineering for Civil Engineers and
%Geoscientists, ISBN 9781927659045
%
%Richard, Martin & Morse, Brian & Daly, Steven. (2015), Modeling Frazil Ice Growth in the St.
%Lawrence River, Canadian Journal of Civil Engineering, 42,
%150106144557001. 10.1139/cjce-2014-0082.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Published with MATLAB® R2021a

```
function [s] = MassSurfengbudgt(s)
%UNTITLED3 %This function loops the Surfengbudgt() function for all Season_DataTable &
%Supercooling_Events pairs for the submitted surfengbudgt structure
%
%INPUT
%   s = input surface energy budget structure with the
%   time series for a deploymnent along with catalogued supercooling event
%   table
%OUTPUT
%   - adds calculated heat fluxes and heat flux statistics to the time
%   series table and supercooling event summary table, respectively.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for riv = 1:length(s)
    fprintf('River: %s\n',s(riv).River);
    for seas = 1:length(s(riv).Season_TimeSeries)
        fprintf('   Season ID: %s\n',s(riv).Season_TimeSeries(seas).Season_ID);
        tt = s(riv).Season_TimeSeries(seas).Season_DataTable; %Time Series data
        evtTab = s(riv).Season_TimeSeries(seas).Supercooling_Events ;
        [tt,evtTab] = Surfengbudgt(tt,evtTab);
        s(riv).Season_TimeSeries(seas).Season_DataTable = tt;
        s(riv).Season_TimeSeries(seas).Supercooling_Events = evtTab;
    end
end
fprintf('Complete\n')
end
```

Published with MATLAB® R2021a

```
function [T] = massTable(seasonTimeseriesStruct,River)
%MASSTABLE: Compiles all individual season time series in structure into one table
narginchk(1,2)
if nargin == 1 %Nor River is specified; compile all data
chk = 0; %Check if a table variable has been started (may matter if a structure is skipped)
for i = 1:length(seasonTimeseriesStruct)
    s = seasonTimeseriesStruct(i).Season_TimeSeries;
    if ~isempty(s)
        for k = 1:length(s)
            tab = s(k).Supercooling_Events;%Old table
            for k1 = 1:height(tab)
                tab.Time_Series(k1) =  string({s(k).Season_ID});%Add timeseries identifier
            end
            tab=tab(:,[end,1:end-1]);
            if ~chk
                T = tab; chk=1;
            else
                T=[T;tab];
            end
        end
    end
end
```

```
else %A river is specified, only consider the specific river
    chk = 0; %Check if a table variable has been started (may matter if a structure is skipped)
    %Determine which row of structure to analyse
    rivs = [seasonTimeseriesStruct.River]; %List of all rivers in structure
    riv = find(rivs == River, 1); %Determine if the specified river is in the structure
    if isempty(riv)
        error('%s is not one of the rivers listed in seasonTimeSeries structure.',River)
    else
        s = seasonTimeseriesStruct(riv).Season_TimeSeries;
        if ~isempty(s)
            for i = 1:length(s)
                tab = s(i).Supercooling_Events;%Old table
                for k = 1:height(tab)
                    tab.Time_Series(k) =  string({s(i).Season_ID});%Add timeseries identifier
                end
                tab=tab(:,[end,1:end-1]);
                if ~chk
                    T =  tab;chk=1;
                else
                    T=[T;tab];
                end
            end
        end
    end
end
```

Published with MATLAB® R2021a