---

title: "Marine Noise Registry Outputs"

author: "RP"

date: "July 2017"

output:

html\_document:

code\_folding: show

fig\_caption: yes

highlight: textmate

number\_sections: yes

self\_contatined: yes

toc: yes

toc\_float:

collapsed: yes

smooth\_scroll: yes

pdf\_document:

highlight: tango

number\_sections: yes

toc: yes

toc\_depth: 3

word\_document:

toc: yes

toc\_depth: '3'

---

<script>

$(document).ready(function() {

$head = $('#header');

$head.prepend('<img src=\"Inputs\\Example\_figures\\JNCC\_logo.jpg\" style=\"float: right;width: 250px;\"/>')

});

</script>

\pagebreak

```{r markdown, echo = FALSE, purl = FALSE, warning = FALSE, eval = TRUE}

library(knitr)

library(rmdformats)

library(bookdown)

library(formatR)

opts\_chunk$set(echo = TRUE, eval = FALSE, warning = FALSE, error = FALSE, comment = NA, tidy.opts=list(width.cutoff=60),tidy=TRUE, root.dir = "Network path")

```

Version 1.0

#Introduction

This document produces outputs from the Marine Noise Registry (MNR); a PostgreSQL database. It connects to the postgreSQL database to pull out queries and manipulates the data to produce three output types.

1. Figures in PNG format.

2. Tables in CSV format.

3. Shapefiles in SHP format.

All of these outputs can be found in the 'Outputs' folder on the J Drive in the project folder 'MNR outputs'.

This script will also call in shapefiles from the Reference folders on the J Drive so please make sure that you are using the most up to date shapefile and edit the file location as appropriate should the file be deprecated.

This script was produced using the following R version:

R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"

Copyright (C) 2016 The R Foundation for Statistical Computing

Platform: i386-w64-mingw32/i386 (32-bit)

This script will also work the \*\*64-bit\*\* version of R.

2020-12-07 ‘GF’ updates and notes

\* removed pacman library from script and sourced functions

\* geom parameter for st\_read has changed names to geometry\_column

\* reporting year needed changing in 3 places, updated to only needed changing once at top of script

\* copyright date updated (line ~200)

\* error with over function CRS not the same (line ~ 1225). Changed CRS to be the same without reprojection as both spatial objects are in WGS84

\* error on bind\_rows as some dataframes are empty (line ~ 1342). Updated to exclude empty data frames in bind\_rows

\* replaced qdap::bracketXtract with readr::parse\_number to remove qdap package which requires Java (line ~ 1370)

\* URL to NoiseImpulsiveRegistry XML schema no longer occurs http://ices.dk/marine-data/Documents/NoiseRegistry/NoiseImpulsiveRegistry.xsd (line ~ 1440) and couldn't find updated location. Previous used XMLSchema taken from network

# error resulting from incorrect number of activities counted as only used raw\_poly\_ogb.df data frame

and not included additional activities in raw\_point\_all.df. Updated to include additional activitieds in raw\_point\_all.df (line ~ 1665)

\* error in scalebar function as dd2km parameter deprecated for transform. dist\_unit = 'km' parameter required (line ~ 1701 and line ~ 1879)

\* added dbDisconnect at end of script

##Setup

In order to produce any outputs the following set up steps need to be taken.

###Install and load packages

The following code checks to see if you have the packages already loaded into R and if not it will install them for you and load them into the global environment (i.e. the project so it can be used).

```{r Install and Load Packages, echo=TRUE, message = FALSE}

rm(list = ls())

library(RPostgres)

library(dplyr)

library(ggplot2)

library(postGIStools)

library(reshape2)

library(raster)

library(tidyverse)

library(rgdal)

library(ggsn)

library(DBI)

library(RColorBrewer)

library(sf)

library(xml2)

library(stringr)

```

Set the year of reporting here

```{r report\_year}

report\_year = 2019

```

###Connect to the Marine Noise Registry PostgreSQL database

Before you attempt to connect to the database you will require a login to read the tables. Please contact JNCC's Data Services team to get one set up.

```{r PostgreSQL connection}

#Load function

# 217.168.21.91

postgres\_connection <- function(port = 5432, host = "XXX", dbname = "XX) {

conn <- DBI::dbConnect(RPostgres::Postgres(),

port = port,

host = host,

dbname = dbname,

connect\_timeout="10",

sslmode = "require",

user = rstudioapi::askForPassword("Enter username: "),

password = rstudioapi::askForPassword("Enter password: ")

)

}

#Run function and enter login credentials

conn <- postgres\_connection()

```

###Read in shapefiles

Read in the shapefiles and make them easier to read for plotting maps.

Shapefiles to read in:

- Marine Strategy Framework Directive (MSFD) sub-regions

- MSFD sub-regions boundary line

- UK & Ireland landmass

- Oil and gas blocks (taken from the MNR)

- Oil and gas quadrants

- Oil and gas block land polygon

```{r read in shapefiles}

#read in OGB shape file

# Need to fix this to avoid get\_postgis\_query - not supported by RPostgres

# OGB <- ?get\_postgis\_query(conn, "SELECT \* from oilandgasblock", geom\_name = "geom")

# This has been replaced with the sf query tool

OGB <- sf::st\_read(conn, query = "SELECT \* from jncc.oilandgasblock", geometry\_column = "geom")

OGB <- sf::as\_Spatial(OGB)

# Need to convert pq\_geometry object OGB$geom to sf object

#fortify oil and gas blocks

OGB\_fort <- fortify(OGB, region = "id")

source("Network path")

#read in UK & Ireland landmass

UK.df <- read\_shp\_ggplot2(filepath = "Network path", filename = "UKIreland\_Coastline\_OSGB36", regionid = "id", joinAttributes = TRUE )

#MSFD sub region polygons

MSFD\_poly\_points <- read\_shp\_ggplot2(filepath = "Network path", filename = "P20170228\_MarineSubRegions\_UKCS\_WGS84", joinAttributes = FALSE)

#MSFD subregion line - this is used to primarily for the legend purpose

MSFD\_line\_points <- read\_shp\_ggplot2(filepath = " Network path ",

filename = "Edited\_MSFD\_boundary\_line\_WGS84\_20170202", regionid = "id", joinAttributes = FALSE )

IoM\_bound<-read\_shp\_ggplot2(filepath =" Network path ", filename="20140311\_IOM\_TS\_UKHO\_WGS84",joinAttributes = FALSE)

#read in oil and gas quadrants

QUAD <- readOGR("Network path ", "OGB\_Quadrants")

#fortify oil and gas quadrants

quad\_fort <- fortify(QUAD, region = "id")

#find the centroids for labelling

QUAD\_centroids <- as.data.frame(coordinates(QUAD))

#update names in the data frame

names(QUAD\_centroids) <- c("long", "lat")

#take out quadrant numbers

idList\_quad = QUAD@data$quadrnt

#create a data frame of the centroids and quadrants

centroids\_quad.df <- data.frame(id = idList\_quad, QUAD\_centroids)

#read in oil and gas blocks on land - should already be in WGS84

OGB\_land\_points <- readOGR("Network path", "OGB\_internal\_onland\_C20171005\_WGS84")

OGB\_land\_points <- spTransform(OGB\_land\_points, CRS("+proj=longlat +datum=WGS84 +no\_defs +ellps=WGS84 +towgs84=0,0,0"))

```

###Copyright statement

Make sure the following copyright statement is correct for the shapefiles used above and is still valid when creating the maps. This information can be found on TopCat.

```{r copyright}

copyright <-

"Contains JNCC information licensed under the Open Government Licence v3.0. JNCC December 2020.

Contains derived data from Ordnance Survey Crown copyright. 100019741 (2015).

Contains Ordnance Survey data Crown copyright and database right 2012. Contains United Kingdom Hydrographic Office data Crown copyright 2012.

Contains UK Oil and Gas Licensing Blocks Oil and Gas Authority 2016.

EEA 2017. Contains MSFD Sub-region boundary data decided on during MSFD committee meeting 2016.

(http://www.arcgis.com/home/webmap/viewer.html?url=http%3A%2F%2Fmarine.discomap.eea.europa.eu%2Farcgis%2Frest%2Fservices%2FMarine%2FMarine\_regions\_subregions%2FMapServer&source=sd)

Contains data collated in the UK Marine Noise Registry to fulfil the UK requirement for monitoring loud, low to mid frequency impulsive noise for MSFD D.11.1.

Not to be used for navigation. Data displayed using WGS84 coordinates."

```

#Calculate Pulse-Block-Days (PBD)

This section includes the code required to create maps and tabular outputs associated with close out pulse-block-day data.

Enter the date range to calculate PBDs. When the date range is entered it should the following format:

\*\*\*"'YYYY-MM-DD' and 'YYYY-MM-DD'"\*\*\*

Enter the month range and year for the titles of the plots. Enter this as: \*\*\*"Month 1 - Month 2 YYYY"\*\*\*

```{r date range}

#e.g. "'2015-01-01' and '2015-12-31'"

# dates <- "'2018-01-01' and '2018-12-31'"

dates <- str\_glue("'{report\_year}-01-01' and '{report\_year}-12-31'")

#e.g. "January - December 2015"

# month <- "January - December 2018"

month <- str\_glue("January - December {report\_year}")

#set up for plotting

#calculate the pulse block day breaks

breaks <- c(0, 1, 5, 10, 20,30,40,50,100, 200,365)

#Blue colour palette

col\_pal\_new3 <- colorRampPalette(c("white","steelblue","black"), space = "Lab", interpolate = "spline")

#the number of colours from the colour palette

col\_pal\_new3 <- col\_pal\_new3(10)

```

## Extract data from Marine Noise Registry Database

Extract all the data from a given date range (this is specified in the code above). The raw closed data entered as points or polygons will need to be intersected with the oil and gas block geometries. This will allow each point or polygon to be assigned a oil and gas block.

Once both the point and polygon data are intersected by oil and gas blocks and the data already entered with oil and gas blocks are extracted from the database they will need to joined together. After which they will be tested for any blocks that contain less than 5% water. This is in the final line of code in the chunk below.

Please follow the instructions printed on the R console on how to proceed with the next steps of analysis.

```{r PBD data}

# Original query by RP - does not bring in acoustic data which is required for screening to ensure MBES /

# Seismic and Sub-Bottom Profiler frequencies are not erroneous (above threshold for MNR entry) ORIGINAL BY RP

# #edit date in the query below

# qry\_locations\_noOGB <-

# paste0("SELECT

# oilandgasblock.\*,

# activityapplication.\*,

# activitylocation.id as activitylocation\_id,

# activitylocation.activityapplication\_id,

# activitylocationdate.activity\_date,

# activitytype.name

# FROM

# jncc.oilandgasblock,

# jncc.activitylocation

# INNER JOIN jncc.activitylocationdate

# ON activitylocationdate.activitylocation\_id = activitylocation.id

# INNER JOIN jncc.activityapplication

# ON activityapplication.id = activitylocation.activityapplication\_id

# INNER JOIN jncc.activitytype

# ON activityapplication.activitytype\_id = activitytype.id

# WHERE

# ST\_Intersects(oilandgasblock.geom,

# ST\_Collect(activitylocation.entered\_point, activitylocation.entered\_polygon))

# AND

# activitylocationdate.activity\_date BETWEEN", dates,

# "ORDER BY activity\_date DESC")

##################################

# NEW VERSION TO REMOVE STATUS == DELETED DATA 29/01/2020

#edit date in the query below

qry\_locations\_noOGB <-

paste0("SELECT

oilandgasblock.\*,

activityapplication.\*,

activitylocation.id as activitylocation\_id,

activitylocation.activityapplication\_id,

activitylocationdate.activity\_date,

activitytype.name

FROM

jncc.oilandgasblock,

jncc.activitylocation

INNER JOIN jncc.activitylocationdate

ON activitylocationdate.activitylocation\_id = activitylocation.id

INNER JOIN jncc.activityapplication

ON activityapplication.id = activitylocation.activityapplication\_id

INNER JOIN jncc.activitytype

ON activityapplication.activitytype\_id = activitytype.id

WHERE

ST\_Intersects(oilandgasblock.geom,

ST\_Collect(activitylocation.entered\_point, activitylocation.entered\_polygon))

AND

not (

(activityapplication.status = 'Deleted')

)

AND

activitylocationdate.activity\_date BETWEEN", dates,

"ORDER BY activity\_date DESC")

##################################

# Updated query by LM to include acoustic frequency levels 06/12/2019 - DOES NOT WORK WITH INNER JOINS, RETURNS EMPTY DF

# LEFT JOIN RETURNS TOO MUCH DATA FOR MEMORY

# qry\_locations\_noOGB <-

# paste0("SELECT

# oilandgasblock.\*,

# activityapplication.\*,

#

# activityacousticdd.frequency\_actual,

# activitysubbottomprofilers.frequency\_actual,

# activitymultibeames.frequency\_actual,

#

# activitylocation.id as activitylocation\_id,

# activitylocation.activityapplication\_id,

#

# activitylocationdate.activity\_date,

# activitytype.name

#

# FROM

#

# jncc.oilandgasblock,

# jncc.activitylocation INNER JOIN jncc.activitylocationdate

# ON activitylocationdate.activitylocation\_id = activitylocation.id

#

# INNER JOIN jncc.activityapplication

# ON activityapplication.id = activitylocation.activityapplication\_id

#

# LEFT JOIN jncc.activityacousticdd

# ON activityacousticdd.activityapplication\_id = activityapplication.id

#

# LEFT JOIN jncc.activitysubbottomprofilers

# ON activitysubbottomprofilers.activityapplication\_id = activityapplication.id

#

# LEFT JOIN jncc.activitymultibeames

# ON activitymultibeames.activityapplication\_id = activityapplication.id

#

# INNER JOIN jncc.activitytype

# ON activityapplication.activitytype\_id = activitytype.id

#

# WHERE not (

# (activityacousticdd.frequency\_actual > 10 and activitytype.name = 'Acoustic Deterrent Device')

# or

# (activitysubbottomprofilers.frequency\_actual > 10000 and activitytype.name = 'Sub-bottom Profilers')

# or

# (activitymultibeames.frequency\_actual > 12000 and activitytype.name = 'Multibeam Echosounders')

# )

# or activityacousticdd.frequency\_actual is null

# or activitysubbottomprofilers.frequency\_actual is null

# or activitymultibeames.frequency\_actual is null

#

# AND

#

# ST\_Intersects(oilandgasblock.geom,

# ST\_Collect(activitylocation.entered\_point, activitylocation.entered\_polygon))

# AND

# activitylocationdate.activity\_date BETWEEN", dates,

# "ORDER BY activity\_date DESC")

# NEW ATTEMPT - 28/11/2019

# Execute query and return data from the MNR DB as a DataFrame

shp\_locations\_noOGB <- sf::st\_read(conn, query = qry\_locations\_noOGB, geometry\_column = "geom")

# Check the loaded DF for blank geometries, remove any empty entries and also cast the data into an sp / spatial object using a pipe '%>%'

shp\_locations\_noOGB <- shp\_locations\_noOGB[!st\_is\_empty(shp\_locations\_noOGB),,drop=FALSE] %>%

sf::as\_Spatial()

# Original query by RP - does not bring in acoustic data which is required for screening to ensure MBES /

# Seismic and Sub-Bottom Profiler frequencies are not erroneous (above threshold for MNR entry)

# pull out locations with oil and gas blocks associated

qry\_locations\_w\_OGB <-

paste0("SELECT

oilandgasblock.\*,

activityapplication.\*,

activitylocation.id as activitylocation\_id,

activitylocation.activityapplication\_id,

activitylocationdate.activity\_date,

activitytype.name

FROM jncc.activitylocation

INNER JOIN jncc.oilandgasblock

ON oilandgasblock.block\_code = activitylocation.entered\_ogb\_code

INNER JOIN jncc.activitylocationdate

ON activitylocationdate.activitylocation\_id = activitylocation.id

INNER JOIN jncc.activityapplication

ON activityapplication.id = activitylocation.activityapplication\_id

INNER JOIN jncc.activitytype

ON activityapplication.activitytype\_id = activitytype.id

WHERE

not (

(activityapplication.status = 'Deleted')

)

AND

activitylocationdate.activity\_date BETWEEN", dates, "")

#

#

# # Updated query by LM to include acoustic frequency levels 06/12/2019

#

# # This query uses left joins in place of inner joins to return the frequency data and not ommit the entries without # associated frequencies (e.g. activities which are not MBES, SBP, Seismic etc.)

# # However, this returns empty geometries which need to be screened out below.

# qry\_locations\_w\_OGB <-

# paste0("SELECT

# oilandgasblock.\*,

# activityapplication.\*,

# activitylocation.id as activitylocation\_id,

# activitylocation.activityapplication\_id,

# activitylocationdate.activity\_date,

# activityacousticdd.frequency\_actual,

# activitytype.name

#

# FROM jncc.activitylocation

#

# LEFT JOIN jncc.oilandgasblock

# ON oilandgasblock.block\_code = activitylocation.entered\_ogb\_code

#

# LEFT JOIN jncc.activitylocationdate

# ON activitylocationdate.activitylocation\_id = activitylocation.id

#

# LEFT JOIN jncc.activityapplication

# ON activityapplication.id = activitylocation.activityapplication\_id

#

# LEFT JOIN jncc.activityacousticdd

# ON activityapplication.id = activityacousticdd.activityapplication\_id

#

# LEFT JOIN jncc.activitytype

# ON activityapplication.activitytype\_id = activitytype.id

#

# WHERE not (

# (activityacousticdd.frequency\_actual >= 12 and activitytype.name = 'Multibeam Echosounders')

# or

# (activityacousticdd.frequency\_actual >= 10 and activitytype.name IN ('Sub-bottom Profilers','Acoustic Deterrent Device'))

# )

# or activityacousticdd.frequency\_actual is null

#

# AND geometrytype(geom) = 'POLYGON'

#

# AND

# activitylocationdate.activity\_date BETWEEN", dates, "")

# Eroneous - not supported by RPostGres - replaced with sf features alternative and converted back to sp feature

# shp\_locations\_w\_OGB <- get\_postgis\_query(conn, qry\_locations\_w\_OGB, geom\_name = "geom")

# EXISTING AJ /LM ATTEMPT

# shp\_locations\_w\_OGB <- sf::st\_read(conn, query = qry\_locations\_w\_OGB)

# shp\_locations\_w\_OGB <- sf::as\_Spatial(shp\_locations\_w\_OGB)

# NEW ATTEMPT 28/11/2019

# Execute query and return data from the MNR DB as a DataFrame

shp\_locations\_w\_OGB <- sf::st\_read(conn, query = qry\_locations\_w\_OGB, geometry\_column = "geom")

# Check the loaded DF for blank geometries, remove any empty entries and also cast the data into an sp / spatial # object using a pipe '%>%'

shp\_locations\_w\_OGB <- shp\_locations\_w\_OGB[!st\_is\_empty(shp\_locations\_w\_OGB),,drop=FALSE] %>%

sf::as\_Spatial()

#bind the two spatial data frames together

shp\_all\_locations <- rbind(shp\_locations\_noOGB, shp\_locations\_w\_OGB)

#check if any blocks need reassigning

if(any(shp\_all\_locations@data$lessthan\_five == TRUE)){print("Run the '2.2 Block Reassignment' code chunk")

} else{print("Skip to '2.3 Calculate pulse-block-day' code chunk")}

```

If block reassignment test = "skip to 'PBD' chunk". Skip to \*\*2.3 Calculate pulse block days\*\*.

##Check for any point on land

Extract points on land to Outputs folder.

Shapefiles and tables produced from this next chunk of code

1. Points on land (shapefile)

2. Polygons on land (shapefile)

3. Data on land (csv)

```{r find land points}

#extract all the closed point data from the MNR

onland\_points<-

paste0("SELECT

activityapplication.\*,

activitylocation.id as activitylocation\_id,

activitylocation.activityapplication\_id,

activitylocationdate.activity\_date,

activitylocation.entered\_ogb\_code as block\_code,

activitytype.name AS activity,

regulator.organisation\_id AS reg\_org\_id,

noiseproducer.organisation\_id AS noise\_org\_id,

organisation.organisation\_name AS regulator\_organisation ,

activitylocation.entered\_point

FROM

jncc.activitylocation

INNER JOIN jncc.activitylocationdate

ON activitylocationdate.activitylocation\_id = activitylocation.id

INNER JOIN jncc.activityapplication

ON activityapplication.id = activitylocation.activityapplication\_id

INNER JOIN jncc.activitytype

ON activityapplication.activitytype\_id = activitytype.id

INNER JOIN jncc.regulator

ON activityapplication.regulator\_id =regulator.id

INNER JOIN jncc.noiseproducer

ON activityapplication.noiseproducer\_id = noiseproducer.id

INNER JOIN jncc.organisation

ON regulator.organisation\_id = organisation.id

WHERE activitylocation.entered\_point IS NOT NULL

AND

not (

(activityapplication.status = 'Deleted')

)

AND

activitylocationdate.activity\_date BETWEEN", dates,

"ORDER BY activity\_date DESC")

# onland\_points <- get\_postgis\_query(conn, onland\_points, geom\_name = "entered\_point")

# Eroneous - not supported by RPostGres - replaced with sf features alternative and converted back to sp feature

onland\_points<- sf::st\_read(conn, query = onland\_points)

onland\_points <- sf::as\_Spatial(onland\_points)

#intersect the point data with the polygon of internal OGB blocks (i.e. on land)

onland\_points <- raster::intersect(onland\_points, OGB\_land\_points)

#write this out as a shapefile

writeOGR(onland\_points, dsn = paste("Network path"), layer=paste0("Points\_onland\_C", Sys.Date()), driver = "ESRI Shapefile")

#remove the spatial aspect of the datato create a csv later

onland <- onland\_points@data

#write out as csv

write.csv(onland, paste0("Network path "), row.names = FALSE)

```

##Block reassignment

There are number of blocks that been identified to have less than 5% water. These blocks need to be reassigned before the pulse-block-days can be calculated.

```{r ogb reassignment}

#convert OGB file from SPDF to simple features data frame

ogb.sf.df <- OGB@data

#ogb.sf.df<- as.data.frame(ogb.sf.df[,1:8])

#create a new column for all of the block codes including those that need reassigning

ogb.sf.df$new\_block\_code <- ifelse(ogb.sf.df$lessthan\_five == TRUE, ogb.sf.df$assignment\_block\_code, ogb.sf.df$block\_code)

#change the column names to reflect the changes

colnames(ogb.sf.df)[colnames(ogb.sf.df)=="block\_code"] <- "old\_block\_code"

colnames(ogb.sf.df)[colnames(ogb.sf.df)=="new\_block\_code"] <- "block\_code"

ogb.sf.df <- distinct(as.data.frame(ogb.sf.df[, "block\_code"]),.keep\_all = TRUE)

colnames(ogb.sf.df)[colnames(ogb.sf.df)=="ogb.sf.df[, \"block\_code\"]"] <- "block\_code"

#change the geometry

ogb.sf.geom <- st\_as\_sf(OGB)

ogb.sf.geom\_new<- distinct(inner\_join(ogb.sf.geom, ogb.sf.df),.keep\_all = TRUE)

#calculate the number of blocks that need reassigning

shp\_reassigned.sf <- anti\_join(shp\_all\_locations@data, ogb.sf.df, by = "block\_code")

nrow(shp\_reassigned.sf)

#convert all closed data to a simple features data frame

shp\_all\_locations.sf <- st\_as\_sf(shp\_all\_locations)

shp\_all\_locations.sf$geometry <- NULL

shp\_all\_locations.sf <- as.data.frame(shp\_all\_locations.sf)

shp\_all\_locations.sf$new\_block\_code <- ifelse(shp\_all\_locations.sf$lessthan\_five == TRUE, shp\_all\_locations.sf$assignment\_block\_code, shp\_all\_locations.sf$block\_code)

#change the column names to reflect the changes

colnames(shp\_all\_locations.sf)[colnames(shp\_all\_locations.sf)=="block\_code"] <- "old\_block\_code"

colnames(shp\_all\_locations.sf)[colnames(shp\_all\_locations.sf)=="new\_block\_code"] <- "block\_code"

#join by only the matching block codes from both datasets

shp\_all\_locations.sf<- inner\_join(shp\_all\_locations@data, ogb.sf.geom\_new, by = "block\_code")

#make spatial again

shp\_all\_locations.sf1 <- st\_as\_sf(shp\_all\_locations.sf)

#update column names so simple features will recognise it as spatial data frame again

#only keep the distinct rows

shp\_all\_locations.sf1 <- distinct\_all(shp\_all\_locations.sf1)

#convert back to a spatial data frame

shp\_all\_locations <- as(shp\_all\_locations.sf1, 'Spatial')

```

##Calculate pulse-block-days

Now the data is ready to calculate the pulse-block-days. The data is aggregated up by block code and activity date. The number of matching instances of block code and activity are summed to produce the pulse-block-day number for a given block.

The pulse-block-day information is added to the spatial data frame ready for plotting.

```{r calculate PBD}

#calculate the pulse block days by aggregating up by block code and activity date

block\_count<- aggregate(activity\_date ~ block\_code, data = distinct(shp\_all\_locations@data[,c("block\_code", "activity\_date")]), FUN = length)

#rename column to be pulse block day

colnames(block\_count)[2] <- "pulse\_block\_day"

#update the spatial data frame by joining this to the data frame created to calculate the pulse block days

shp\_all\_locations@data <- data.frame(shp\_all\_locations@data, block\_count[match(shp\_all\_locations@data[, "block\_code"], block\_count[, "block\_code"]), ])

```

##Pulse-block-day map

The pulse-block-day map will be saved as a PNG e.g. \*"Pulse\_block\_day\_map\_CYYYYMMDD.png"\*. This file will be saved to Outputs folder in section [Save Pulse-block-day outputs].

The map shows the distribution of impulsive noise events for all activity types combined, across UK waters for a supplied date range. Data recorded at point (Lat Long) resolution is translated to Oil and Gas licensing block level.

The map includes the following:

\* Title

\* Legend

+ UK landmass

+ UK oil and gas block grid

+ Number of pulse-block-days using colour to differentiate between the each range of PBD.

+ MSFD sub regions

\* Scale bar & north arrow

\* Latitude longitude – graticule grid labels

\* Projection/datum (WGS84)

\* Copyright statement

An example map can be found in the APPENDIX [Pulse-block-day example map].

```{r plot}

#fortify pulse block day map and add attribute data after fortifying

shp\_all\_locations@data$id = rownames(shp\_all\_locations@data)

all\_loc.points = fortify(shp\_all\_locations, region = "id")

shp\_all\_locations.df = left\_join(all\_loc.points, shp\_all\_locations@data, by="id")

source(' Network path’)

PBD\_FULL\_MAP <- Pulse\_block\_day\_map(

OGB\_fort, long, lat, group, MSFD\_poly\_points, MSFD\_line\_points, IoM\_bound, col\_pal\_new3, UK.df,

shp\_all\_locations.df, breaks, month, copyright)

```

##Pulse-block-day data table

The PBD data table is the information that sits behind the map. Data is taken from the data frame that produced the PBD map and counts the number of times a block and activity have taken place. The data frame is then transformed so the activities are individual columns and joined with PBD data table by block code to include the PBD information. MSFD subregions are then assigned to each block.

```{r PBD table}

##EXTRACT DATA FROM THE SHAPEFILE, THE COLUMNS REQUIRED FOR ANALYSIS:

## 1. OIL AND GAS BLOCK CODE

## 2. ACTIVITY NAME

## 3. DATE OF ACTIVITY

##REMOVE DUPLICATES THAT MAY OF COME FROM THE MNR

Closeout <- distinct(shp\_all\_locations@data[, c("block\_code", "name", "activity\_date")])

##COUNT THE NUMBER OF TIMES AN BLOCK AND ACTIVITY HAVE TAKEN PLACE

Closeout <- aggregate(activity\_date ~ block\_code + name, data = Closeout, FUN = length)

##RESHAPE THE DATAFRAME SO THE ACTIVITIES ARE INDIVIDUAL COLUMNS WITH THE PULSE BLOCK DAYS

Closeout <- dcast(Closeout, block\_code ~ name, value.var = "activity\_date")

##JOIN THIS DATA WITH THE PULSE BLOCK DAY DATA

Closeout <- inner\_join(block\_count, Closeout)

#EXTRACT DATA FROM OIL AND GAS BLOCKS - THERE ARE 4407 BLOCKS

OGB\_blocks <- OGB@data

##JOIN OUTPUTS WITH THE OIL AND GAS BLOCKS

##THIS JOIN WILL KEEP ALL OIL AND GAS BLOCKS AND ADD THE ACTIVITY PULSE BLOCK DAY ASSOCIATED WITH THAT BLOCK IF APPLICABLE ELSE AN 'NA' WILL BE RETURNED IN THE APPROPRIATE COLUMN

Closeout<- left\_join(OGB\_blocks, Closeout)

##READ THE OIL AND GAS BLOCKS THAT HAVE BEEN ASSIGNED A MSFD SUBREGION

assigned\_blocks <- read.csv("Network path ", header = TRUE, stringsAsFactors = FALSE)

#RENAME COLUMN 1 TO JOIN TO THE ACTIVITY PULSE BLOCK OUTPUT

colnames(assigned\_blocks)[1] <- "block\_code"

##JOIN OIL AND GAS BLOCK ASSIGNED TO THE OUTPUTS

Closeout <- left\_join(Closeout, assigned\_blocks)

```

###Check for duplicates

There are sometimes duplicates in the MNR for a particular activity and block. The following code will check for duplicates and write out the output as e.g. \*"Pulse\_block\_day\_closeout\_duplicates\_CYYYY-MM-DD.csv"\*

```{r check for duplicates}

##FIND THE DUPLICATES IN THIS DATASET AND WRITE OUT THE RESULTS

CloseoutDups <- shp\_all\_locations@data[, c("block\_code", "name", "activity\_date")]

CloseoutDups <- CloseoutDups[duplicated(CloseoutDups),]

###JOIN TO PULSE BLOCK DAY OUTPUTS

Closeout4Dups<- inner\_join(shp\_all\_locations@data, CloseoutDups)

write.csv(Closeout4Dups, paste0("Network path ", Sys.Date(), ".csv"), row.names = FALSE)

```

##Raw data table - impulsive noise records

Tabular output (CSV / spreadsheet) of raw (non-summarised) impulsive noise records. The data used will be from ‘close-out reports’ (status = ‘closed’) for a given date range, based on inputs of start date and end date. Spatial locations are as Well Known Text (WKT) for both point records and block level records. Full event attributes are included.

```{r raw impulsive noise data}

# This output will produce a csv (tabular output) of closed activity data from a given time period

#extract the closed acitivities with WKT including those that have only ogb entered

WKT\_Activities<-

paste0("SELECT

activityapplication.\*,

activitylocation.id as activitylocation\_id,

activitylocation.activityapplication\_id,

activitylocationdate.activity\_date,

activitylocation.entered\_ogb\_code as block\_code,

activitytype.name AS activity,

ST\_AsText(ST\_Collect(activitylocation.entered\_point, activitylocation.entered\_polygon)) AS WKT,

regulator.organisation\_id AS reg\_org\_id,

noiseproducer.organisation\_id AS noise\_org\_id,

organisation.organisation\_name AS regulator\_organisation

FROM

jncc.activitylocation

INNER JOIN jncc.activitylocationdate

ON activitylocationdate.activitylocation\_id = activitylocation.id

INNER JOIN jncc.activityapplication

ON activityapplication.id = activitylocation.activityapplication\_id

INNER JOIN jncc.activitytype

ON activityapplication.activitytype\_id = activitytype.id

INNER JOIN jncc.regulator

ON activityapplication.regulator\_id =regulator.id

INNER JOIN jncc.noiseproducer

ON activityapplication.noiseproducer\_id = noiseproducer.id

INNER JOIN jncc.organisation

ON regulator.organisation\_id = organisation.id

WHERE

not (

(activityapplication.status = 'Deleted')

)

AND

activitylocationdate.activity\_date BETWEEN", dates, "ORDER BY activity\_date DESC")

activities\_WKT <- DBI::dbGetQuery(conn, WKT\_Activities)

activities\_WKT[is.na(activities\_WKT$wkt), 'wkt'] <- 'POINT EMPTY'

# activities\_WKT <- activities\_WKT[!is.na(activities\_WKT$wkt), ]

activities\_WKT <- sf::st\_as\_sf(activities\_WKT, wkt='wkt')

# activities\_WKT <- sf::as\_Spatial(activities\_WKT)

#activities\_WKT <- sf::st\_read(conn, query = WKT\_Activities, geom\_column = 'WKT')

#get organisational information

qry\_org <-

"SELECT organisation.id AS noise\_org\_id,

organisation.organisation\_name AS noiseproducer\_organisation

FROM jncc.organisation"

org <- DBI::dbGetQuery(conn, qry\_org)

#join output with organisational data

activities\_WKT <- left\_join(activities\_WKT, org)

#extract ogb with WKT

ogbwkt <-

"SELECT oilandgasblock.block\_code,

ST\_AsText(geom) as wkt

FROM jncc.oilandgasblock"

ogbpostgis\_WKT <- DBI::dbGetQuery(conn, ogbwkt)

ogbpostgis\_WKT <- sf::st\_as\_sf(ogbpostgis\_WKT, wkt='wkt')

#ogbpostgis\_WKT <- sf::st\_read(conn, query = ogbwkt, geom\_column='wkt')

#ogbpostgis\_WKT <- sf::as\_Spatial(ogbpostgis\_WKT)

#join ogb to output above to get the WKT for ogb

WKT\_ogb\_activities <- left\_join(as.data.frame(activities\_WKT),

as.data.frame(ogbpostgis\_WKT),

by = "block\_code")

#concatenate WKT into one column

WKT\_ogb\_activities$WKT <- ifelse(is.na(WKT\_ogb\_activities$wkt.x),

WKT\_ogb\_activities$wkt.y, WKT\_ogb\_activities$wkt.x)

#remove unwanted column WKT columns

WKT\_ogb\_activities$wkt.x <- NULL

WKT\_ogb\_activities$wkt.y <- NULL

#sf::st\_sf(WKT\_ogb\_activities, sf\_column\_name ='WKT')

#extract all activities

#extract data on sub bottom profilers

#r\_sub <- sf::st\_read(conn, query = "SELECT \* FROM jncc.activitysubbottomprofilers")

#r\_sub <- sf::as\_Spatial(r\_sub)

r\_sub <- DBI::dbGetQuery(conn, "SELECT \* FROM jncc.activitysubbottomprofilers")

#r\_sub <- sf::st\_as\_sf(ogbpostgis\_WKT, wkt='wkt')

sub\_WKT <- inner\_join(WKT\_ogb\_activities, r\_sub, by = "activityapplication\_id")

#piling

r\_pil <- DBI::dbGetQuery(conn, "SELECT \* FROM jncc.activitypiling")

#r\_pil <- sf::as\_Spatial(r\_pil)

pil\_WKT <- inner\_join(WKT\_ogb\_activities, r\_pil, by = "activityapplication\_id")

#ministry of defence

r\_mod <- DBI::dbGetQuery(conn, "SELECT \* FROM jncc.activitymod")

#r\_mod <- sf::as\_Spatial(r\_mod)

mod\_WKT <- inner\_join(WKT\_ogb\_activities, r\_mod, by = "activityapplication\_id")

#seismic

r\_seismic <- DBI::dbGetQuery(conn, "SELECT \* FROM jncc.activityseismic")

#r\_seismic <- sf::as\_Spatial(r\_seismic)

seismic\_WKT <- inner\_join(WKT\_ogb\_activities, r\_seismic, by = "activityapplication\_id")

#explosives

r\_explosives <- DBI::dbGetQuery(conn, "SELECT \* FROM jncc.activityexplosives")

#r\_explosives <- sf::as\_Spatial(r\_explosives)

explosives\_WKT <- inner\_join(WKT\_ogb\_activities, r\_explosives, by = "activityapplication\_id")

#acoustic deterrent devices

r\_add <- DBI::dbGetQuery(conn, "SELECT \* FROM jncc.activityacousticdd")

#r\_add <- sf::as\_Spatial(r\_add)

add\_WKT <- inner\_join(WKT\_ogb\_activities, r\_add, by = "activityapplication\_id")

#multibeam echo sounder

r\_multibeames <- DBI::dbGetQuery(conn, "SELECT \* FROM jncc.activitymultibeames")

#r\_multibeames <- sf::as\_Spatial(r\_multibeames)

multibeames\_WKT <- inner\_join(WKT\_ogb\_activities, r\_multibeames, by = "activityapplication\_id")

#bind the rows from

raw\_data\_table<- bind\_rows(pil\_WKT, explosives\_WKT, mod\_WKT, sub\_WKT, seismic\_WKT, add\_WKT, multibeames\_WKT)

# Filter out all MBES / SBP over threshold here???

#extract the block assigned from the PBD map data

shp\_noOGB.df <- shp\_locations\_noOGB@data[, c("name",

"block\_code",

"date\_closed",

"date\_due",

"date\_end",

"date\_start",

"date\_updated",

"duration",

"status",

"activityapplication\_id",

"non\_licensable")]

#make this into a data frame

shp\_noOGB.df <- shp\_locations\_noOGB@data

#rename columns

colnames(shp\_noOGB.df)[2]<- "NoOGB\_assign\_block\_code"

colnames(shp\_noOGB.df)[1] <- "activity\_type"

#join PBD data with raw WKT data

final\_raw\_data <- left\_join(raw\_data\_table, shp\_noOGB.df)

#assign the empty fields (i.e. data already assigned OGB) with original OGB block code

final\_raw\_data$NoOGB\_assign\_block\_code <- ifelse(is.na(final\_raw\_data$NoOGB\_assign\_block\_code),

final\_raw\_data$block\_code,

final\_raw\_data$NoOGB\_assign\_block\_code)

#only keep the columns we are interested in

colkeep <- c("activity", "date\_start", "date\_end" ,

"date\_closed" , "date\_due" , "date\_updated" ,

"duration" , "status", "non\_licensable" ,

"activity\_date" , "block\_code" , "NoOGB\_assign\_block\_code",

"regulator\_organisation" , "noiseproducer\_organisation" ,

"WKT" , "max\_hammer\_energy" ,

"sound\_pressure\_level" , "sound\_exposure\_level" , "sound\_pressure\_level\_actual",

"sound\_exposure\_level\_actual", "max\_hammer\_energy\_actual" , "tnt\_equivalent" ,

"tnt\_equivalent\_actual" , "source" , "frequency" ,

"frequency\_actual", "data\_type" , "max\_airgun\_volume",

"other\_survey\_type" , "survey\_type" , "max\_airgun\_volume\_actual" )

#rename this column for extraction

colnames(final\_raw\_data)[colnames(final\_raw\_data)=="name"] <- "activity"

#extract the columns we are interested from above

final\_raw\_data <- final\_raw\_data[,colkeep]

```

##Pulse-block-day maps per activity type

A series of maps, one for each Activity Type, in the same format as the main PBD map showing the distribution of pulse–block-days for each activity type. There are seven activity types.

An example of this map can be found in the APPENDIX [Pulse-block-day maps per activity type example].

```{r PBD per activity type}

source(' Network path’)

PBD\_ActType\_FULL\_map <- PBD\_activity\_type(OGB\_fort, long, lat, group, shp\_all\_locations.df, breaks, MSFD\_poly\_points, MSFD\_line\_points,IoM\_bound, UK.df, col\_pal\_new3, name, month, copyright)

#set the below to the working directory where the outputs file is located

wd <- " Network path"

#save PBD per activity type output

ggsave(PBD\_ActType\_FULL\_map, filename = paste0(wd, "Outputs\\Figures\\Pulse-block-day\_per\_ActType\_C", Sys.Date(), ".png"), width = 20/1.214, height = 20, device = "png", dpi = 600)

```

##Activity type block contribution map

This map shows the locations of different activity types in oil and gas blocks. Where two activities occur happen in the same block the activities are combined.

###Activity type block contribution map data

The data for this map is based off the closeout data table. This table is refined to only contain block code and activity type data. The activity type data is concatenated into a a singular column and where there is an overlap of activites they are combined together. See the R console to see which activities need combining (example code for this can be found below - follow the comments). This data is then combined with the main PBD map for plotting.

```{r Create Activity map}

#only extract data with pulse block days

actmap<- subset(Closeout, !is.na(Closeout$pulse\_block\_day))

source(' Network path’)

#extract block columns and activity columns

actmap <- actmap[, c(2, 10:ncol(actmap))]

actmap$AssessArea <- NULL

nocol <- length(colnames(actmap))

#extract explosive only data into a new column

if("Explosives" %in% colnames(actmap)){actmap$Explo <- ifelse(!is.na(actmap$Explosives), "Explosives", NA)}

#actmap$Explo <- switch(is.na(actmap$Explo), NULL)

#extract piling only data into a new column

if("Piling" %in% colnames(actmap)){actmap$Pil <- ifelse(!is.na(actmap$Piling), "Piling", NA)}

#extract seismic only data into a new column

if("Seismic Survey" %in% colnames(actmap)){actmap$SS <- ifelse(!is.na(actmap$`Seismic Survey`), "Seismic Survey", NA)}

#extract MOD only data into a new column

if("Ministry of Defence" %in% colnames(actmap)){actmap$MOD <- ifelse(!is.na(actmap$`Ministry of Defence`),"Ministry of Defence", NA)}

#extract sub-bottom profiling only data into a new column

if("Sub-bottom Profilers" %in% colnames(actmap)){actmap$SBP <- ifelse(!is.na(actmap$`Sub-bottom Profilers`), "Sub-bottom Profilers", NA)}

#extract multibeam echosounder only data into a new column

if("Multibeam Echosounders" %in% colnames(actmap)){actmap$MBES <- ifelse(!is.na(actmap$`Multibeam Echosounders`), "Multibeam Echosounders", NA)}

#extract acoustic deterrent devices only data into a new column

if("Acoustic Deterrent Device" %in% colnames(actmap)){actmap$ADD <- ifelse(!is.na(actmap$`Acoustic Deterrent Device`), "Acoustic Deterrent Device", NA)}

# actmap <- replace\_na(actmap, list(Explo = is.null(2)), Explo)

#actmap$Explo <- if (actmap$Explo == 2) { NULL} else { actmap$SBP}

#concatenate these into an Activity column

#list all the column names

nocol <- nocol + 1

# actmapAgg <- within(actmap, Activity <- paste0(actmap[, nocol:ncol(actmap)], sep= "\n"))

#actmapAgg$test <- stri\_c(actmap$Explo, actmap$Pil, sep = " & ", ignore\_null = TRUE)

#view the unique activities present

#unique(actmapAgg$Activity)

#if there are multiple instances where the activites overlap copy and paste the following code and rename appropriately.

actmap$Activity <- paste5(actmap[, nocol:ncol(actmap)], sep = " & ", na.rm = TRUE)

#actmapAgg$Activity <- ifelse(actmapAgg$Activity == "\n\nSeismic Survey\n\nSub-bottom Profilers", "Seismic Survey & Sub-bottom Profilers", actmapAgg$Activity)

#make a copy of the maine PBD map to add activities and for plotting later

actmapOGB <- shp\_all\_locations

#match data from above to the the PBD map by block code

actmapOGB@data <- data.frame(actmapOGB@data, actmap[match(actmapOGB@data[, "block\_code"], actmap[, "block\_code"]), ])

#fortify map

actmapOGB@data$id = rownames(actmapOGB@data)

all\_act.points = fortify(actmapOGB, region = "id")

actmapOGB.df = left\_join(all\_act.points, actmapOGB@data, by="id")

#set a colour palette

colpal2 <- colorRampPalette(c("#623734", "#7E566A", "#79829D",

"#62B1B4", "#82DAA5", "#DFF889"), space = "Lab", interpolate = "spline")

#the number of activities

colno <- length(unique(actmapOGB.df$Activity))

#tell the palette how many colours we need

colpal2 <- colpal2(colno)

#turn activity column into a factor rather than a character

#actmapOGB.df$Activity= factor(actmapOGB.df$Activity,levels=actmapOGB.df$Activity, ordered = is.ordered(actmapOGB.df$Activity))

actmapOGB.df[order(actmapOGB.df[,"Activity"]),]

#change all the /n into blank entries

actmapOGB.df$Activity <- gsub("\n","", actmapOGB.df$Activity)

```

###Activity type block contribution map plot

The following code will produce a map to show the locations of activities in oil and gas blocks. An example of this map can be found in the APPENDIX [Activity type block contribution map plot example].

```{r plot activity type contribution map}

source(' Network path’)

ActType\_cont\_FULL\_map <- ActType\_contribution\_FULL\_MAP(OGB\_fort, long, lat, group, MSFD\_poly\_points, MSFD\_line\_points,IoM\_bound, actmapOGB.df, Activity, colpal2, UK.df, month, copyright)

ggsave(ActType\_cont\_FULL\_map, filename = paste0(wd, "Outputs\\Figures\\Activity\_block\_contribution\_map\_C", Sys.Date(), ".png"), width = 20, height = (20/1.414), device = "png", dpi = 600)

```

##Save Pulse-block-day outputs

The outputs of a pulse-block-day will consist of:

1. PNG output - e.g \*"Pulse\_block\_day\_map\_CYYYY-MM-DD.png"\*

2. Closeout summary table output (Closed data table) - e.g. \*"Pulse\_block\_day\_closeout\_summary\_table\_CYYYY-MM-DD.csv"\*

3. ESRI Shapefile output - e.g. \*"Pulse\_block\_day\_map\_CYYYY-MM-DD.shp"\*

4. Raw closeout table - e.g. \*"Raw\_closeout\_table\_CYYYY-MM-DD.csv"\*

5. Pulse-block-day per activity type e.g. \*Pulse-block-day\_per\_ActType\_CYYYY-MM-DD.png\*

6. Activity type block contribution map e.g. \*Activity\_block\_contribution\_map\_CYYYY-MM-DD.png\*

This adheres to JNCC's spatial data management protocol, where 'C' = 'Created' indicating the date which the output was created. Followed by the date of creation (the current date on the system).

```{r save PBD outputs}

#save PBD main map

ggsave(PBD\_FULL\_MAP, filename = paste0(wd, "Outputs\\Figures\\Pulse\_block\_day\_map\_C", Sys.Date(), ".png"), width = 20, height = (20/1.414), device = "png", dpi = 600)

#closeout summary table

write.csv(Closeout, paste0(wd, "Outputs\\Tabular\\Pulse\_block\_day\_closeout\_summary\_table\_C", Sys.Date(), ".csv"), row.names = FALSE, quote = TRUE)

#save PBD main shapefile

writeOGR(shp\_all\_locations, dsn = paste0(wd, "Outputs\\Shapefiles"), layer = paste0("Pulse\_block\_day\_map\_C", Sys.Date()), driver = "ESRI Shapefile")

#write out raw closeout data table

write.csv(sf::st\_sf(final\_raw\_data, sf\_column\_name = 'WKT'), paste0(wd, "Outputs\\Tabular\\Raw\_closeout\_table\_C", Sys.Date(),".csv"), row.names = FALSE, quote = T)

#save PBD per activity type output

ggsave(PBD\_ActType\_FULL\_map, filename = paste0(wd, "Outputs\\Figures\\Pulse-block-day\_per\_ActType\_C", Sys.Date(), ".png"), width = 20/1.214, height = 20, device = "png", dpi = 600)

#save activity type block contribution map

ggsave(ActType\_cont\_FULL\_map, filename = paste0(wd, "Outputs\\Figures\\Activity\_block\_contribution\_map\_C", Sys.Date(), ".png"), width = 20, height = (20/1.414), device = "png", dpi = 600)

```

\pagebreak

#European Noise Registry output

The UK is required to submit data for impulsive noise to the European Noise Registry annually in an XML format for ingestion into their data systems. The XML schema is described at

http://ices.dk/marine-data/Documents/NoiseRegistry/NoiseImpulsiveRegistry.xsd

The output is a raw close-out report event data for a calendar year at point (lat Long) or block level with events classified as very low, low, medium, high or very high. Guidelines for the translation of UK MNR records to European standard records are provided in Appendix 3. The selection of data from the MNR is to be by activity type (select all or specific activity types) and by date.

##Organise data frame to parse XML

###Extract data from MNR

###Select year of extraction

Enter the date range to create XML output. When the date range is entered it should the following format:

\*\*\*"'YYYY-MM-DD' and 'YYYY-MM-DD'"\*\*\*

```{r XML date range}

#e.g. "'2015-01-01' and '2015-12-31'"

# <- "'2016-01-01' and '2016-12-31'"

dates <- str\_glue("'{report\_year}-01-01' and '{report\_year}-12-31'")

```

###Extract data from MNR

Extract the data from the MNR ready to manipulate.

```{r all activity data}

#extract the closed acitivities with WKT including those that have only ogb entered

XML\_Activities<-

paste0("SELECT

activityapplication.\*,

activitylocation.id as activitylocation\_id,

activitylocation.activityapplication\_id,

activitylocationdate.activity\_date,

activitylocation.entered\_ogb\_code as block\_code,

activitytype.name AS activity,

ST\_AsText(ST\_Collect(activitylocation.entered\_point, activitylocation.entered\_polygon)) AS WKT,

regulator.organisation\_id AS reg\_org\_id,

noiseproducer.organisation\_id AS noise\_org\_id,

organisation.organisation\_name AS regulator\_organisation

FROM

jncc.activitylocation

INNER JOIN jncc.activitylocationdate

ON activitylocationdate.activitylocation\_id = activitylocation.id

INNER JOIN jncc.activityapplication

ON activityapplication.id = activitylocation.activityapplication\_id

INNER JOIN jncc.activitytype

ON activityapplication.activitytype\_id = activitytype.id

INNER JOIN jncc.regulator

ON activityapplication.regulator\_id =regulator.id

INNER JOIN jncc.noiseproducer

ON activityapplication.noiseproducer\_id = noiseproducer.id

INNER JOIN jncc.organisation

ON regulator.organisation\_id = organisation.id

WHERE

not (

(activityapplication.status = 'Deleted')

)

AND

activitylocationdate.activity\_date BETWEEN", dates,

"ORDER BY activity\_date DESC")

# output\_XML <- get\_postgis\_query(conn, XML\_Activities)

output\_XML <- DBI::dbGetQuery(conn, XML\_Activities)

qry\_org <-

"SELECT organisation.id AS noise\_org\_id,

organisation.organisation\_name AS noiseproducer\_organisation

FROM jncc.organisation"

org <- DBI::dbGetQuery(conn, qry\_org)

output\_XML <- left\_join(output\_XML, org)

#extract ogb with WKT

ogbxml <-

"SELECT oilandgasblock.block\_code,

ST\_AsText(geom) as wkt

FROM jncc.oilandgasblock"

ogbpostgis\_xml <- DBI::dbGetQuery(conn, ogbxml)

#join ogb to output above to get the WKT for ogb

output\_XML2 <- left\_join(output\_XML, ogbpostgis\_xml, by = "block\_code")

#concatenate WKT into one column

output\_XML2$WKT <- ifelse(is.na(output\_XML2$wkt.x), output\_XML2$wkt.y, output\_XML2$wkt.x)

#remove unwanted column WKT columns

output\_XML2$wkt.x <- NULL

output\_XML2$wkt.y <- NULL

#intersect this with in the internal polygon of OGB to remove any points that fall on land

#read in shapefile

#intersect with activity location data with shapefile

```

###Check if there are any points on land

Please check if there are any points/polygons on land by referrring to the spreadsheet created in section Calculating Pulse Block Days.

This next section of code will remove points and polygons from the XML.

```{r remove onland points}

#extract all the closed point data

XML\_Activities\_onland\_points<-

paste0("SELECT

activityapplication.\*,

activitylocation.id as activitylocation\_id,

activitylocation.activityapplication\_id,

activitylocationdate.activity\_date,

activitylocation.entered\_ogb\_code as block\_code,

activitytype.name AS activity,

regulator.organisation\_id AS reg\_org\_id,

noiseproducer.organisation\_id AS noise\_org\_id,

organisation.organisation\_name AS regulator\_organisation ,

activitylocation.entered\_point

FROM

jncc.activitylocation

INNER JOIN jncc.activitylocationdate

ON activitylocationdate.activitylocation\_id = activitylocation.id

INNER JOIN jncc.activityapplication

ON activityapplication.id = activitylocation.activityapplication\_id

INNER JOIN jncc.activitytype

ON activityapplication.activitytype\_id = activitytype.id

INNER JOIN jncc.regulator

ON activityapplication.regulator\_id =regulator.id

INNER JOIN jncc.noiseproducer

ON activityapplication.noiseproducer\_id = noiseproducer.id

INNER JOIN jncc.organisation

ON regulator.organisation\_id = organisation.id

WHERE activitylocation.entered\_point IS NOT NULL

AND

not (

(activityapplication.status = 'Deleted')

)

AND

activitylocationdate.activity\_date BETWEEN", dates,

"ORDER BY activity\_date DESC")

XML\_onland\_points <- sf::st\_read(conn, query = XML\_Activities\_onland\_points, geometry\_column = "entered\_point")

XML\_onland\_points <- sf::as\_Spatial(XML\_onland\_points)

#intersect the closed point data with the internal OGB blocks

# XML\_onland <- raster::intersect(OGB\_land\_points, XML\_onland\_points) - original code (points / poly switched)

# 2020-12-02 ‘GF’ over function fails as CRS are not the same

# XML\_onland:"+proj=longlat +datum=WGS84 +no\_defs"

# OGB\_land\_points: "+proj=longlat +ellps=WGS84 +towgs84=0,0,0,0,0,0,0 +no\_defs"

# As differences appear slight with both in WGS84, XML\_onland\_points proj4string set to OGB\_land\_points proj4string without reprojection

proj4string(XML\_onland\_points) <- proj4string(OGB\_land\_points)

XML\_onland <- sp::over(x = XML\_onland\_points, y = OGB\_land\_points)

not\_onland <- suppressWarnings(apply(XML\_onland, 1, function(x) all(lapply(x, is.na))))

#extract the data frame only

XML\_onland <- XML\_onland\_points[not\_onland,]

XML\_onland <- as.data.frame(XML\_onland)

#only keep the data that isn't on land

output\_XML3 <- anti\_join(output\_XML2, XML\_onland)

##rename so the outputs can be used for the next section of code

output\_XML2 <- output\_XML3

```

###Generate dataframe for XML

Create a data ready to parse into a XML document.

```{r generate data frame}

#extract data on sub bottom profilers

r\_sub <- DBI::dbGetQuery(conn, "SELECT \* FROM jncc.activitysubbottomprofilers")

sub\_WKT <- inner\_join(output\_XML2, r\_sub, by = "activityapplication\_id")

#piling

r\_pil <- DBI::dbGetQuery(conn, "SELECT \* FROM jncc.activitypiling")

pil\_WKT <- inner\_join(output\_XML2, r\_pil, by = "activityapplication\_id")

#ministry of defence

r\_mod <- DBI::dbGetQuery(conn, "SELECT \* FROM jncc.activitymod")

mod\_WKT <- inner\_join(output\_XML2, r\_mod, by = "activityapplication\_id")

#seismic

r\_seismic <- DBI::dbGetQuery(conn, "SELECT \* FROM jncc.activityseismic")

seismic\_WKT <- inner\_join(output\_XML2, r\_seismic, by = "activityapplication\_id")

#explosives

r\_explosives <- DBI::dbGetQuery(conn, "SELECT \* FROM jncc.activityexplosives")

explosives\_WKT <- inner\_join(output\_XML2, r\_explosives, by = "activityapplication\_id")

#acoustic deterrent devices

r\_add <- DBI::dbGetQuery(conn, "SELECT \* FROM jncc.activityacousticdd")

add\_WKT <- inner\_join(output\_XML2, r\_add, by = "activityapplication\_id")

#multibeam echo sounder

r\_multibeames <- DBI::dbGetQuery(conn, "SELECT \* FROM jncc.activitymultibeames")

multibeames\_WKT <- inner\_join(output\_XML2, r\_multibeames, by = "activityapplication\_id")

#seismic survey

XML\_SS <- seismic\_WKT

XML\_SS$value\_code <- ifelse(XML\_SS$sound\_pressure\_level >= 209 & XML\_SS$sound\_pressure\_level <= 233, "very\_low",

ifelse(XML\_SS$sound\_pressure\_level >= 234 & XML\_SS$sound\_pressure\_level <= 243, "low",

ifelse(XML\_SS$sound\_pressure\_level >= 244 & XML\_SS$sound\_pressure\_level <= 253, "medium",

ifelse( XML\_SS$sound\_pressure\_level > 253, "high", NA))))

#acoustic deterrent devices

XML\_ADD <- add\_WKT

XML\_ADD$value\_code <- ifelse(XML\_ADD$sound\_pressure\_level >= 176 & XML\_ADD$sound\_pressure\_level <= 200, "very\_low",

ifelse(XML\_ADD$sound\_pressure\_level >= 201 & XML\_ADD$sound\_pressure\_level <= 210, "low",

ifelse(XML\_ADD$sound\_pressure\_level >= 211 & XML\_ADD$sound\_pressure\_level <= 220, "medium",

ifelse( XML\_ADD$sound\_pressure\_level > 220, "high", NA))))

#sub bottom profilers

XML\_SBP <- sub\_WKT

XML\_SBP$value\_code <- ifelse(XML\_SBP$sound\_exposure\_level >= 186 & XML\_SBP$sound\_exposure\_level <= 210, "very\_low",

ifelse(XML\_SBP$sound\_exposure\_level >= 211 & XML\_SBP$sound\_exposure\_level <= 220, "low",

ifelse(XML\_SBP$sound\_exposure\_level >= 221 & XML\_SBP$sound\_exposure\_level <= 230, "medium",

ifelse( XML\_SBP$sound\_exposure\_level > 230, "high", NA))))

#Multibeam echosounders

XML\_MBES <- multibeames\_WKT

XML\_MBES$value\_code <- ifelse(XML\_MBES$sound\_exposure\_level >= 186 & XML\_MBES$sound\_exposure\_level <= 210, "very\_low",

ifelse(XML\_MBES$sound\_exposure\_level >= 211 & XML\_MBES$sound\_exposure\_level <= 220, "low",

ifelse(XML\_MBES$sound\_exposure\_level >= 221 & XML\_MBES$sound\_exposure\_level <= 230, "medium",

ifelse( XML\_MBES$sound\_exposure\_level > 230, "high", NA))))

#need to make we are not using bankers rounding (i.e. to nearest even integer)

roundup <- function(x, digits=0) {

factor <- 10^digits

trunc(x\*factor + 0.5)/factor

}

#Piling

XML\_PIL <- pil\_WKT

XML\_PIL$max\_hammer\_energy <- roundup(XML\_PIL$max\_hammer\_energy, -1)

XML\_PIL$value\_code <- ifelse(XML\_PIL$max\_hammer\_energy <= 280 & !is.na(XML\_PIL$max\_hammer\_energy) , "very\_low",

ifelse(XML\_PIL$max\_hammer\_energy >= 290 & XML\_PIL$max\_hammer\_energy <= 2800, "low",

ifelse(XML\_PIL$max\_hammer\_energy >= 2810 & XML\_PIL$max\_hammer\_energy <= 28000, "medium",

ifelse( XML\_PIL$max\_hammer\_energy > 28000, "high", NA))))

#explosives

XML\_EXPL <- explosives\_WKT

XML\_EXPL$tnt\_equivalent <- roundup(XML\_EXPL$tnt\_equivalent, -1)

XML\_EXPL$value\_code <- ifelse(XML\_EXPL$tnt\_equivalent >= 0.008 & XML\_EXPL$tnt\_equivalent <= 0.21, "very\_low",

ifelse(XML\_EXPL$tnt\_equivalent >= 0.22 & XML\_EXPL$tnt\_equivalent <= 2.1, "low",

ifelse(XML\_EXPL$tnt\_equivalent >= 2.11 & XML\_EXPL$tnt\_equivalent <= 21, "medium",

ifelse( XML\_EXPL$tnt\_equivalent >= 22 & XML\_EXPL$tnt\_equivalent <= 210, "high",

ifelse(XML\_EXPL$tnt\_equivalent > 210, "very\_high", NA)))))

#MOD

XML\_MOD <- mod\_WKT

XML\_MOD$value\_code <- NA

# 2020-12-04 ‘GF’ error here as XML\_ADD is an empty data frame in 2019, updated to empty data frames

#XML\_ALL<- bind\_rows(XML\_ADD, XML\_EXPL, XML\_MBES, XML\_PIL, XML\_SBP, XML\_SS, XML\_MOD)

XML\_ALL <- map\_df(list(XML\_ADD, XML\_EXPL, XML\_MBES, XML\_PIL, XML\_SBP, XML\_SS, XML\_MOD),

function(x) {if(nrow(x) != 0) {bind\_rows(x)}})

#fill in the rest of the columns

#Geometry type column

XML\_ALL$Geometry\_type <- ifelse(XML\_ALL$block\_code != "", "UK\_Licence\_Blocks",

ifelse(str\_detect(XML\_ALL$WKT, "POINT"), "Point", NA))

#start date and end date column

XML\_ALL$start\_date <- format(XML\_ALL$activity\_date, "%Y-%m-%d")

XML\_ALL$end\_date <- format(XML\_ALL$activity\_date, "%Y-%m-%d")

#sound mitigation boolean is always no

XML\_ALL$sound\_mitigation\_bool <- "No"

#split up lat and long for point - regex everything inside the bracket up the comma for Long and the opposite for Lat

# XML\_ALL$Longitude <- ifelse(XML\_ALL$Geometry\_type == "Point", gsub( " .\*$", "", qdap::bracketXtract(XML\_ALL$WKT, bracket = "round")) , NA)

# XML\_ALL$Latitude <- ifelse(XML\_ALL$Geometry\_type == "Point", gsub( ".+? ", "", qdap::bracketXtract(XML\_ALL$WKT, bracket = "round")) , NA)

# 2020-12-07 ‘GF’ replaced with readr::parse\_number to remove qdap package which requires Java

XML\_ALL$Longitude <- ifelse(XML\_ALL$Geometry\_type == "Point", parse\_number(str\_extract(XML\_ALL$WKT, "^.+ ")) , NA)

XML\_ALL$Latitude <- ifelse(XML\_ALL$Geometry\_type == "Point", parse\_number(str\_extract(XML\_ALL$WKT, " .\*$")) , NA)

#data entry point ID is the same as activity application ID number

XML\_ALL$data\_entry\_point\_ID <- XML\_ALL$activityapplication\_id

#polygon\_ID - this is associated with ogb

XML\_ALL$polygon\_ID <- ifelse(!is.na(XML\_ALL$block\_code), XML\_ALL$block\_code, NA)

#source event - assigning the activities we have in to ICES activities/events

XML\_ALL$source\_event <- ifelse(XML\_ALL$activity == "Seismic Survey", "Airgun\_arrays",

ifelse(XML\_ALL$activity == "Sub-bottom Profilers" | XML\_ALL$activity == "Multibeam Echosounders", "Generic\_explicitly\_impulsive\_source",

ifelse(XML\_ALL$activity == "Piling","Impact\_pile\_driver",

ifelse(XML\_ALL$activity == "Explosives", "Explosions",

ifelse(XML\_ALL$activity == "Acoustic Deterrent Device", "Sonar\_or\_acoustic\_deterrents",

ifelse(str\_detect(XML\_ALL$source, "Sonar"), "Sonar\_or\_acoustic\_deterrents",

"Explosions"))))))

#extract the columns we need to create the XML doc

XML\_cols <- XML\_ALL[, c("data\_entry\_point\_ID", "start\_date", "end\_date", "Latitude", "Longitude", "Geometry\_type", "polygon\_ID", "source\_event", "value\_code", "sound\_mitigation\_bool" )]

```

## Load function to xml document

This function creates an XML document from the data frame created in the previous code chunks. The function will loop over each row in the data frame and create an entry. It will then combine all these together to create a full XML output of the closed data from a given year.

### MNR XML function

```{r fn create XML document}

create\_xml\_document <- function(data){

xml\_add\_nonblank\_sibling <- function(baseroot, element\_name, element\_value) {

if (!is.na(element\_value)) {

baseroot <- baseroot %>%

xml\_add\_sibling(element\_name, element\_value)

}

return(baseroot)

}

baseroot <- xml\_new\_root("noise\_impulsive\_registry") %>%

xml\_add\_child("country", "GB") %>%

xml\_add\_sibling("organization", "1375") %>%

xml\_add\_sibling("preparation\_date", format(Sys.Date(), "%Y-%m-%d")) %>%

xml\_root()

for (i in 1:nrow(data)) {

baseroot <- baseroot %>%

xml\_add\_child("station") %>%

xml\_add\_child("data\_entry\_point\_ID", data$data\_entry\_point\_ID[i]) %>%

xml\_add\_sibling("start\_date", as.character(data$start\_date[i])) %>%

xml\_add\_sibling("end\_date", as.character(data$end\_date[i])) %>%

xml\_add\_nonblank\_sibling("latitude", data$Latitude[i]) %>%

xml\_add\_nonblank\_sibling("longitude", data$Longitude[i]) %>%

xml\_add\_sibling("geometry\_type", data$Geometry\_type[i]) %>%

xml\_add\_sibling("polygon\_ID", data$polygon\_ID[i]) %>%

xml\_add\_sibling("event") %>%

xml\_add\_child("source\_event", data$source\_event[i]) %>%

xml\_add\_sibling("value\_code", data$value\_code[i]) %>%

xml\_add\_sibling("sound\_mitigation\_bool", data$sound\_mitigation\_bool[i]) %>%

xml\_root()

}

return(baseroot)

}

```

###Create XML document

Run the function loaded into the global environment from the previous chunk to create the XML document.

```{r run function on df}

xmlout <- create\_xml\_document(XML\_cols)

```

##Get ICES XML schema

Download the ICES schema to check the xml output created from above is validated.

```{r ICES schema}

# ICES\_schema <- read\_xml("http://ices.dk/marine-data/Documents/NoiseRegistry/NoiseImpulsiveRegistry.xsd")

# write\_xml(ICES\_schema, "xmlschema.xml")

ICES\_schema <- read\_xml("xmlschema.xml")

```

##Validate XML

Validate XML output with the ICES schema. If validated should return as

\*\*[1] TRUE

attr(,"errors")

character(0)\*\* in the R console.

```{r validate}

xml\_validate(xmlout, ICES\_schema)

```

##Save XML document

Save XML document as \*"XML\_CYYYY-MM-DD.xml"\*

```{r write out xml}

write\_xml(xmlout, paste0(wd, "Outputs\\XML\\XML\_C", Sys.Date(), ".xml"))

```

\pagebreak

#Planned noise events

The following chunks of codes will produce maps and tables for planned activities over a specific calendar year. Outputs will include a raw data tables and raw location data.

##Specify year and dates of planned noise

\* Enter the year of planned noise as \*\*"YYYY"\*\* e.g. \*"2015"\*

\* Enter start date and end date as \*\*"'YYYY-MM-DD'"\*\* e.g. \*"'2015-01-01'"

\* Enter months for title of plots as \*\*"Month 1 - Month 2"\*\* e.g. \*"January - December"\*

```{r year of planned noise}

#enter calendar of year of planned noise

Year\_planned\_noise <- as.character(report\_year)

#enter start date and enter date e.g. "'2015-01-01"

Start\_date <- str\_glue("{report\_year}-01-01")

End\_date <- str\_glue("{report\_year}-12-31")

#months for title of plots e.g. "January - December"

title\_months <- "January - December"

```

##Planned noise event map (raw location data)

### Extracting data from MNR

```{r planned raw data extraction}

#extract point data only

point\_raw\_qry <-

"SELECT

activityapplication.\*,

activitylocation.id as activitylocation\_id,

activitylocation.activityapplication\_id,

activitylocation.entered\_ogb\_code as block\_code,

activitytype.name,

activitylocation.entered\_point as geom\_point,

ST\_AsText(activitylocation.entered\_point) as point\_location

FROM jncc.activitylocation

INNER JOIN jncc.activityapplication

ON activityapplication.id = activitylocation.activityapplication\_id

INNER JOIN jncc.activitytype

ON activityapplication.activitytype\_id = activitytype.id

WHERE

not (

(activityapplication.status = 'Deleted')

)

AND

activitylocation.entered\_point IS NOT NULL"

planned\_raw\_point <- sf::st\_read(conn, query = point\_raw\_qry, geometry\_column = "geom\_point")

planned\_raw\_point <- sf::as\_Spatial(planned\_raw\_point)

#filter raw point data by start date within the two dates specified above

filtered\_raw\_point\_start <- subset(planned\_raw\_point, planned\_raw\_point$date\_start >= as.Date(Start\_date) & planned\_raw\_point$date\_start <= as.Date(End\_date))

#filter raw point data by end date within the two dates specfified above

filtered\_raw\_point\_end <- subset(planned\_raw\_point, planned\_raw\_point$date\_end >= as.Date(Start\_date) & planned\_raw\_point$date\_end <= as.Date(End\_date))

#join the filtered data together

raw\_point\_all <- rbind(filtered\_raw\_point\_start, filtered\_raw\_point\_end)

#make sure the data is in a data frame

raw\_point\_all.df <- as.data.frame(raw\_point\_all)

###extract polygon data

poly\_raw\_qry <-

"SELECT

activityapplication.\*,

activitylocation.id as activitylocation\_id,

activitylocation.activityapplication\_id,

activitylocation.entered\_ogb\_code as block\_code,

activitytype.name,

activitylocation.entered\_polygon as geom\_poly,

ST\_AsText(activitylocation.entered\_polygon) as polygon\_location

FROM jncc.activitylocation

INNER JOIN jncc.activityapplication

ON activityapplication.id = activitylocation.activityapplication\_id

INNER JOIN jncc.activitytype

ON activityapplication.activitytype\_id = activitytype.id

WHERE

not (

(activityapplication.status = 'Deleted')

)

AND

activitylocation.entered\_polygon IS NOT NULL"

planned\_raw\_poly <- sf::st\_read(conn, query = poly\_raw\_qry, geometry\_column = "geom\_poly")

planned\_raw\_poly <- sf::as\_Spatial(planned\_raw\_poly)

#filter start date by start date from the specified dates above

filtered\_raw\_poly\_start <- subset(planned\_raw\_poly, planned\_raw\_poly$date\_start >= as.Date(Start\_date) & planned\_raw\_poly$date\_start <= as.Date(End\_date))

#filter start date by end date from the specified dates above

filtered\_raw\_poly\_end <- subset(planned\_raw\_poly, planned\_raw\_poly$date\_end >= as.Date(Start\_date) & planned\_raw\_poly$date\_end <= as.Date(End\_date))

#bind together with spatial information

raw\_poly\_all <- rbind(filtered\_raw\_poly\_start, filtered\_raw\_poly\_end)

###extract ogb data

ogb\_raw\_qry <-

"SELECT

activityapplication.\*,

activitylocation.id as activitylocation\_id,

activitylocation.activityapplication\_id,

activitylocation.entered\_ogb\_code as block\_code,

activitytype.name,

activitylocation.entered\_ogb\_code,

oilandgasblock.\*

FROM jncc.activitylocation

INNER JOIN jncc.oilandgasblock

ON oilandgasblock.block\_code = activitylocation.entered\_ogb\_code

INNER JOIN jncc.activityapplication

ON activityapplication.id = activitylocation.activityapplication\_id

INNER JOIN jncc.activitytype

ON activityapplication.activitytype\_id = activitytype.id

WHERE

not (

(activityapplication.status = 'Deleted')

)

AND

activitylocation.entered\_ogb\_code IS NOT NULL"

planned\_raw\_ogb <- sf::st\_read(conn, query = ogb\_raw\_qry, geometry\_column = "geom")

planned\_raw\_ogb <- sf::as\_Spatial(planned\_raw\_ogb)

#subset start date for specified dates as specified above

filtered\_raw\_ogb\_start <- subset(planned\_raw\_ogb, planned\_raw\_ogb$date\_start >= as.Date(Start\_date) & planned\_raw\_ogb$date\_start <= as.Date(End\_date))

#subset end date for specified dates as specified above

filtered\_raw\_ogb\_end <- subset(planned\_raw\_ogb, planned\_raw\_ogb$date\_end >= as.Date(Start\_date) & planned\_raw\_ogb$date\_end <= as.Date(End\_date))

#combine data together

raw\_ogb\_all <- rbind(filtered\_raw\_ogb\_start, filtered\_raw\_ogb\_end)

#remove duplicates for aggregation

raw\_ogb\_all\_nodups<- distinct(raw\_ogb\_all@data)

#order by activity name

raw\_ogb\_all\_nodups <- raw\_ogb\_all\_nodups[order(raw\_ogb\_all\_nodups$name), ]

#aggregate up by activity and block - i.e. combining activities that are found in the same block

raw\_ogb\_all\_nodups1 <- raw\_ogb\_all\_nodups %>%

distinct(name, block\_code) %>%

group\_by(block\_code) %>%

summarise(name = paste(name, collapse = ' & ')) %>%

ungroup()

raw\_ogb\_all\_nodups$name <- NULL

raw\_ogb\_all\_nodups <- distinct(raw\_ogb\_all\_nodups)

raw\_ogb\_all\_nodups <- left\_join(raw\_ogb\_all\_nodups, raw\_ogb\_all\_nodups1)

#add spatial data back in

raw\_ogb\_all@data <- data.frame(raw\_ogb\_all@data,

raw\_ogb\_all\_nodups[match(raw\_ogb\_all@data[, "block\_code"],

raw\_ogb\_all\_nodups[, "block\_code"]), ])

#just keep the activity name column to bind polygon and ogb data together and rename so they match

raw\_poly\_all\_act <- raw\_poly\_all[,c("name")]

raw\_ogb\_all\_act <- raw\_ogb\_all[, c( "name.1")]

colnames(raw\_ogb\_all\_act@data)[1] <- "name"

#bind all raw polygon data together (OGB + polygon)

raw\_poly\_ogb <- rbind(raw\_poly\_all\_act, raw\_ogb\_all\_act)

raw\_poly\_ogb@data$id = rownames(raw\_poly\_ogb@data)

raw\_poly\_ogb.fort = fortify(raw\_poly\_ogb, region = "id")

raw\_poly\_ogb.df = left\_join(raw\_poly\_ogb.fort, raw\_poly\_ogb@data, by="id")

#find the number of colours we need

#set a colour palette

colpal2 <- colorRampPalette(c("#623734", "#7E566A", "#79829D",

"#62B1B4", "#82DAA5", "#DFF889"), space = "Lab", interpolate = "spline")

# 2020-12-07 Error resulting from incorrect number of activities counted as only used raw\_poly\_ogb.df data frame

# and not included additional activities in raw\_point\_all.df. Updated to include additional activitieds in raw\_point\_all.df

#the number of activities

#colno <- length(unique(raw\_poly\_ogb.df$name))

colno <- raw\_poly\_ogb.df %>% dplyr::select(name) %>%

bind\_rows(raw\_point\_all.df %>% dplyr::select(name)) %>%

distinct() %>%

nrow()

#tell the palette how many colours we need

colpal2 <- colpal2(colno)

```

###Plot planned raw noise events

The planned raw locations are now ready for plotting. An example map can be found in the APPENDIX [Plot planned raw noise events example].

```{r plot raw data}

planned\_raw\_p1 <- ggplot() +

#add oil and gas block quadrant polygons

geom\_polygon(data= quad\_fort, aes( x = long, y = lat, group = group, color = "Oil and Gas Block Quadrants"), fill = NA)+

#add MSFD polygon

geom\_polygon(data=MSFD\_poly\_points, aes(x = long, y = lat, group= group, color = "UK Continental Shelf"), fill = NA) +

#add MSFD sub regions boundary line

geom\_path(data=MSFD\_line\_points, aes(x = long, y = lat, group = group, size = "MSFD sub-regions"), color = "black") +

#add UK landmass

geom\_polygon(data = UK.df, aes(x = long, y = lat, group = group, color = "grey50", fill = "grey70"),fill = "grey70", color = "grey50", show.legend= F)+

#add raw polygon data (polygons and OGB data combined)

geom\_polygon(data = raw\_poly\_ogb.df[order(raw\_poly\_ogb.df$name), ], aes(x = long, y = lat, group = group, fill = name)) +

#add raw point data

geom\_point(data = raw\_point\_all.df, aes(x = coords.x1, y = coords.x2, fill = name), pch= 21, size = 3) +

#add labels to quadrant polygons

geom\_text(data = centroids\_quad.df, aes(x = long, y = lat, label = id), size = 3)+

#set theme to black and white and also theme size to 18

theme\_bw(18)

planned\_raw\_p2 <- planned\_raw\_p1 +

#add legend for sub region boundary

scale\_size\_manual(values = c("MSFD sub-regions"= 1.2), guide\_legend( order = 2, title = ""))+

#add legend for oil and gas block quadrants and UK continental shelf (MSFD regional seas boundary layer)

scale\_color\_manual(values = c("UK Continental Shelf" = "black", "Oil and Gas Block Quadrants" = "grey"), guide\_legend(order = 1))+

#add legend for activity type

scale\_fill\_manual(values = colpal2, drop = FALSE) +

###ADD SCALE BAR

scalebar( dist = 100, transform = TRUE, model = 'WGS84', dist\_unit = 'km', st.size = 4, st.dist = 0.2, x.min = 0, x.max = 2, y.min = 48.5, y.max = 49, height = 0.5)+

#theme attributes

theme(legend.box = "vertical", aspect.ratio = 1/1.414, plot.title = element\_text(hjust = 0.5), legend.background = element\_blank(), legend.justification=c(0,0.1), legend.key = element\_rect(linetype = "blank", colour = "black"), legend.key.size = unit(3, "line" ), legend.key.height = unit(1.7, "line"), legend.key.width = unit(1.7, "line"), legend.position=c(0.02,0.05),plot.caption = element\_text(size = 9, vjust = 0, hjust = 1) ) +

#Plot title

ggtitle(paste0("Proposed Activity Type Map - Raw Location Data (", title\_months, " ", Year\_planned\_noise, ")" )) +

#axis titles

ylab("Latitude") + xlab("Longitude") +

###ADD COPYRIGHT

labs(caption = copyright) +

####FIX COORDINATES

coord\_cartesian(xlim=c(-23,2.5), ylim = c(48.75,63.25)) +

#add north arrow

north(OGB\_fort, location = "topleft", symbol = 12, scale = 0.05)+

#order the legends and add titles

guides(colour= guide\_legend(order = 1, title = "Legend"), fill = guide\_legend(order= 3, title = "Activity"), size = guide\_legend( order = 2, title = NULL) )

```

###Save outputs

Save outputs from raw planned noise events. This will include:

1. Proposed raw activities map - e.g. \*Proposed\_raw\_activities\_CYYYY-MM-DD.png\*

2. Raw proposed point shapefile - e.g. \*Proposed\_raw\_point\_locations\_CYYYY-MM-DD.shp\*

3. Raw proposed polygon shapefile - e.g. \*Proposed\_raw\_polygon\_locations\_CYYYY-MM-DD.shp\*

4. Raw proposed oil and gas block shapefile - e.g. \*Proposed\_raw\_OGB\_locations\_CYYYY-MM-DD.shp\*

```{r save raw outputs}

#save PNG

ggsave(planned\_raw\_p2, filename = paste0(wd, "Outputs\\Figures\\Proposed\_raw\_activities\_C", Sys.Date(), ".png"), width = 20, height = (20/1.414), device = "png", dpi = 600)

#write out raw point data

writeOGR(raw\_point\_all, paste0(wd, "Outputs\\Shapefiles"), layer = paste0("Proposed\_raw\_point\_locations\_C", Sys.Date()), driver = "ESRI Shapefile")

#write out raw polygon data

writeOGR(raw\_poly\_all, paste0(wd, "Outputs\\Shapefiles"), layer = paste0("Proposed\_raw\_polygon\_locations\_C", Sys.Date()), driver = "ESRI Shapefile")

#write out raw OGB data

writeOGR(raw\_ogb\_all, paste0(wd, "Outputs\\Shapefiles"), layer = paste0("Proposed\_raw\_OGB\_locations\_C", Sys.Date()), driver = "ESRI Shapefile")

```

##Planned noise event map (in UK oil and blocks)

This map will show all the planned noise events in oil and gas blocks. The data which is in point or polygon form is intersected with the oil and gas block geometry to be assigned a block number. The same calendar year specified previously is used. The block quadrants are labelled in this map and where there are multiple activities in one block they are combined.

Note: \*\*To complete this section the 'Planned raw locations' data needs to run beforehand\*\*

###Extract data from MNR

```{r}

#read in quadrant shapefile

#QUAD <- readOGR("Inputs\\Shapefiles\\OGB\_vclean\_diss.shp", "OGB\_vclean\_diss")

#find the centroids for labelling

#QUAD\_centroids <- as.data.frame(coordinates(QUAD))

#rename columns to lat and long

#names(QUAD\_centroids) <- c("long", "lat")

#take the quadrant numbers

#idList\_quad = QUAD@data$quadrnt

#combine centroid coordinates with quadrant numbers

#centroids\_quad.df <- data.frame(id = idList\_quad, QUAD\_centroids)

###need to combine the OGB, polygon, and point data together

rawogb\_point\_poly\_qry <-

"SELECT

activityapplication.\*,

activitylocation.id as activitylocation\_id,

activitylocation.activityapplication\_id,

activitylocation.entered\_ogb\_code as block\_code,

activitytype.name,

activitylocation.entered\_point as geom\_point,

ST\_AsText(activitylocation.entered\_point) as point\_location,

oilandgasblock.\*,

activitylocation.entered\_polygon

FROM jncc.oilandgasblock, jncc.activitylocation

INNER JOIN jncc.activityapplication

ON activityapplication.id = activitylocation.activityapplication\_id

INNER JOIN jncc.activitytype

ON activityapplication.activitytype\_id = activitytype.id

WHERE

not (

(activityapplication.status = 'Deleted')

)

AND

ST\_Intersects(oilandgasblock.geom,

ST\_Collect(activitylocation.entered\_polygon, activitylocation.entered\_point))"

rawogb\_point\_poly <- sf::st\_read(conn, query = rawogb\_point\_poly\_qry, geometry\_column = "geom")

rawogb\_point\_poly <- sf::as\_Spatial(rawogb\_point\_poly)

#filter start date by dates specfied earlier

filtered\_rawogb\_point\_poly\_start <- subset(rawogb\_point\_poly, rawogb\_point\_poly$date\_start >= as.Date(Start\_date) & rawogb\_point\_poly$date\_start <= as.Date(End\_date))

#filter end date by dates specfied earlier

filtered\_rawogb\_point\_poly\_end <- subset(rawogb\_point\_poly, rawogb\_point\_poly$date\_end >= as.Date(Start\_date) & rawogb\_point\_poly$date\_end <= as.Date(End\_date))

#bind together with spatial information

rawogb\_point\_poly\_all <- rbind(filtered\_rawogb\_point\_poly\_start, filtered\_rawogb\_point\_poly\_end)

#extract the activity application, block code and activity name from the raw for writing out as a shapefile later

rawogb\_point\_poly\_all\_col <- rawogb\_point\_poly\_all[, c("activityapplication\_id", "block\_code..24", "name")]

#extract the activity application, block code and activity name from the raw for writing out as a shapefile later

raw\_ogb\_all\_col <- raw\_ogb\_all[, c("activityapplication\_id", "block\_code.1", "name")]

#combine raw ogb polygon, point and raw OGB planned data together

names(rawogb\_point\_poly\_all\_col) <- c("activityapplication\_id","block\_code.1","name")

raw\_planned\_all <- rbind(raw\_ogb\_all\_col, rawogb\_point\_poly\_all\_col)

#extract columns needed for aggregation

raw\_planned\_all\_v1 <- raw\_planned\_all[, c("block\_code.1", "name")]

#remove duplicates from dataset

raw\_planned\_all\_sum<- distinct(raw\_planned\_all\_v1@data)

#order activity names alphabetically

raw\_planned\_all\_sum <- raw\_planned\_all\_sum[order(raw\_planned\_all\_sum$name), ]

#aggregate up by name and block code and combine activities where they are found in the same bloack

raw\_planned\_all\_sum <- aggregate(name ~ block\_code.1, data = raw\_planned\_all\_sum, paste, collapse = ' & ')

#add spatial information

raw\_planned\_all\_v1@data <- data.frame(raw\_planned\_all\_v1@data, raw\_planned\_all\_sum[match(raw\_planned\_all\_v1@data[, "block\_code.1"], raw\_planned\_all\_sum[, "block\_code.1"]), ])

#extract attribute data from spatial polygons data frame

raw\_planned\_all\_v1@data$id = rownames(raw\_planned\_all\_v1@data)

#fortify raw ogb data for plotting

raw\_planned\_all.points = fortify(raw\_planned\_all\_v1, region = "id")

#join spatial data to attributes

raw\_planned\_all.df = left\_join(raw\_planned\_all.points, raw\_planned\_all\_v1@data, by = "id")

#create the colour palette

colpal2 <- colorRampPalette(c("#623734", "#7E566A", "#79829D", "#62B1B4", "#82DAA5", "#DFF889"), space = "Lab", interpolate = "spline")

#calculate the number of colours needed from the number of activities

colno <- length(unique(raw\_planned\_all.df$name.1))

#assign the palette to the numbers calculated

colpal2 <- colpal2(colno)

```

###Plot planned noise event map (oil and gas blocks)

An example of this map can be found in the APPENDIX [Plot planned noise event map (oil and gas blocks) example].

```{r plot raw ogb map}

#plot all the layers including the quadrants

rawOGB\_p1 <- ggplot() + geom\_polygon(data= quad\_fort, aes( x = long, y = lat, group = group, color = "Oil and gas block quadrants"), fill = NA)+

#add UK continental shelf

geom\_polygon(data=MSFD\_poly\_points, aes(x = long, y = lat, group= group, color = "UK Continental Shelf"), fill = NA) +

#add MSFD subregion

geom\_path(data=MSFD\_line\_points, aes(x = long, y = lat, group = group, size = "MSFD sub-regions"), color = "black") +

#add UK & Ireland landmass

geom\_polygon(data = UK.df, aes(x = long, y = lat, group = group, color = "grey50", fill = "grey70"),fill = "grey70", color = "grey50", show.legend= F)+

#add raw ogb planned event data

geom\_polygon(data = raw\_planned\_all.df[order(raw\_planned\_all.df$name), ], aes(x = long, y = lat, group = group, fill = name.1)) +

#label the quadrants

geom\_text(data = centroids\_quad.df, aes(x = long, y = lat, label = id), size = 3)+

#assign theme size

theme\_bw(18)

#add the legends and other aesthetics

rawOGB\_p2 <- rawOGB\_p1 + scale\_size\_manual(values = c("MSFD sub-regions"= 1.2), guide\_legend( order = 2, title = ""))+

#add colour legend

scale\_color\_manual(values = c("UK Continental Shelf" = "black", "Oil and gas block quadrants" = "grey"), guide\_legend(order = 1))+

#add fill legend

scale\_fill\_manual(values = colpal2, drop = FALSE) +

#ADD SCALE BAR

scalebar( dist = 100, transform = TRUE, dist\_unit = 'km', model = 'WGS84', st.size = 4, st.dist = 0.2, x.min = 0, x.max = 2, y.min = 48.5, y.max = 49, height = 0.5)+

#theme attributes

theme(legend.box = "vertical", aspect.ratio = 1/1.414, plot.title = element\_text(hjust = 0.5), legend.background = element\_blank(), legend.justification=c(0,0.1), legend.key = element\_rect(linetype = "blank", colour = "black"), legend.key.size = unit(3, "line" ), legend.key.height = unit(1.7, "line"), legend.key.width = unit(1.7, "line"), legend.position=c(0.02,0.05),plot.caption = element\_text(size = 9, vjust = 0, hjust = 1) ) +

#add plot title

ggtitle(paste0("Proposed Activity Type Block Contribution Map (", title\_months, " ", Year\_planned\_noise, ")") ) +

#add axis label

ylab("Latitude") + xlab("Longitude") +

#ADD COPYRIGHT

labs(caption = copyright) +

#FIX COORDINATES

coord\_cartesian(xlim=c(-23,2.5), ylim = c(48.75,63.25)) +

#add north arrow

north(OGB\_fort, location = "topleft", symbol = 12, scale = 0.05)+

#order the legends

guides(colour= guide\_legend(order = 1, title = "Legend"), fill = guide\_legend(order= 3, title = "Activity"), size = guide\_legend( order = 2, title = NULL) )+

#at the MSFD sub regions labels

annotate("text", x=-10, y=49.65, label = 'atop(bold("Celtic Seas"))', size = 7, parse = TRUE) + annotate("text", y=49.65, x= 1.65, size =7, label = 'atop(bold("Greater North Sea"))' , parse = TRUE)

```

###Save outputs

Save outputs from raw planned oil and gas block noise events. This will include:

1. Raw OGB activities map - e.g. \*Raw\_OGB\_activity\_CYYYY-MM-DD.png\*

2. Proposed OGB shapefile - e.g. \*Proposed\_OGB\_locations\_CYYYY-MM-DD.shp\*

```{r save raw ogb map outputs}

ggsave(rawOGB\_p2, filename = paste0(wd, "Outputs\\Figures\\Raw\_OGB\_activity\_C", Sys.Date(), ".png"), width = 20, height = (20/1.414), device = "png", dpi = 600)

writeOGR(raw\_planned\_all, paste0(wd, "Outputs\\Shapefiles"), layer = paste0("Proposed\_OGB\_locations\_C", Sys.Date()), driver = "ESRI Shapefile")

```

##Planned noise data table

Tabular output (CSV/spreadsheet) of raw (non-summarised) location data for proposed noise event data within the specified date range from earlier. This tables contains adequate information to produce a shape file from. To produce this table the previous sections of planned noise events need to be run.

###Manipulate data

```{r, manipulate data}

##POLYGON DATA

#intersect with ogb blocks

raw\_poly\_all\_ogb <- raster::intersect(raw\_poly\_all, OGB)

#add in extra columns for binding later

raw\_poly\_all\_ogb$point\_location <- NA

#extract the columns we need for binding

raw\_poly\_all\_ogb\_col <- raw\_poly\_all\_ogb[, c("block\_code.2", "activityapplication\_id", "name", "date\_start", "date\_end", "duration", "status", "point\_location", "polygon\_location")]

#rename columns for binding later

names(raw\_poly\_all\_ogb\_col)[names(raw\_poly\_all\_ogb\_col)=="block\_code.2"] <- "block\_code.1"

##POINT DATA

#intersect with ogb blocks

raw\_point\_all\_ogb <- raster::intersect(raw\_point\_all, OGB)

#add in extra columns for binding later

raw\_point\_all\_ogb$polygon\_location <- NA

#extract the columns we need for binding

raw\_point\_all\_ogb\_col <- raw\_point\_all\_ogb[, c("block\_code.1", "activityapplication\_id", "name", "date\_start", "date\_end", "duration", "status", "point\_location")]

###OGB DATA

#add in extra columns for binding later

raw\_ogb\_all$polygon\_location <- NA

raw\_ogb\_all$point\_location <- NA

#extract the columns we need

raw\_ogb\_all\_col <- raw\_ogb\_all[, c("block\_code.1", "activityapplication\_id", "name", "date\_start", "date\_end", "duration", "status", "point\_location", "polygon\_location")]

##### bind the three datasets together

planned\_raw <- bind\_rows(raw\_ogb\_all\_col@data, raw\_point\_all\_ogb\_col@data, raw\_poly\_all\_ogb\_col@data)

###rename the columns

colnames(planned\_raw) <- c("block\_code", "activity\_application\_number", "activity\_type", "start\_date", "end\_date", "estimated\_duration", "status", "point\_location", "polygon\_location")

```

###Save outputs

Save the raw planned output as \*Planned\_noise\_YYYY\_CYYYY-MM-DD.csv\*

Where the first 'YYYY' is the calendar year of planned noise and the date is the date of the table creation.

```{r save raw data table outputs}

write.csv(planned\_raw, paste0(wd, "Outputs\\Tabular\\Planned\_noise", Year\_planned\_noise, "\_C", Sys.Date(), ".csv"), row.names = FALSE)

```

```{r tidyup}

dbDisconnect(conn)

```

#APPENDIX

##Example maps

###Pulse-block-day example map

The code for this plot is in section 2.4 [Pulse-block-day map]

![](Inputs\\Example\_figures\\5.1\_PBD\_MSFD\_C20170216\_v1.21.png)

###Pulse-block-day maps per activity type example

The code for this plot is in section 2.7 [Pulse-block-day maps per activity type]

![](Inputs\\Example\_figures\\facetwrap\_test23.png)

###Activity type block contribution map plot example

The code for this plot is in section 2.8.2 [Activity type block contribution map plot].

![](Inputs\\Example\_figures\\AllActivityMap\_C20170217.png)

###Plot planned raw noise events example

The code for this plot is in section 4.4.2 [Plot planned raw noise events]

![](Inputs\\Example\_figures\\5.7\_Proposed\_activity\_C20170623\_v1.3.png)

###Plot planned noise event map (oil and gas blocks) example

The code for this plot is in section 4.3.2 [Plot planned noise event map (oil and gas blocks)]

![](Inputs\\Example\_figures\\5.8\_Raw\_OGB\_activity\_C20170620\_v1.10.png)

#BUILD STATUS

|Version | Date | Author | Reason/Comments |

|--------|-----------|--------|-------------------------------------------------------------------|

|0.1 | 07/07/2017| RP | Initial draft |

|1.0 | 20/03/2018| RP | Layers altered so the PBD layer is topmost for closed data |

| | | | Updated the block reassignment code |

| | | | Added in a section of code to identify: |

| | | | Polygons/points on land (shapefiles & csv) |

| | | | Added code to XML section to remove data on land |

|1.1 | 29/08/2018| RP | Fixed bugs in Planned noise code |

|1.2 | 31/08/2018| RP | Automated activity type contribution map string concat |

|1.3 | 19/02/2019| LM, AJ | Added support for SSL database connection & spatial formatting |

|1.31 | 07/12/2020| GF | Updated script to work with 2019 Marine Noise Registry Data |

#DISTRIBUTION:

|Copy | Version | Issue Date | Issued to |

|-----|-----------|------------|------------|

|Link | 0.1 | 07/07/2017 | CF |