Oct 28, 2024

ARMS-MBON 18S rRNA and COI gene metabarcoding: scanning for non-indigenous species

DOI
dx.doi.org/10.17504/protocols.io.n92ldmmmnl5b/v1

Nauras Daraghmeh[1]

[1]Department of Marine Sciences, University of Gothenburg, Sweden

SWEDNA Swedish eDNA …

Nauras Daraghmeh
Department of Marine Sciences, University of Gothenburg, Swe...

OPEN ACCESS

DOI: dx.doi.org/10.17504/protocols.io.n92ldmmmnl5b/v1

Protocol Citation: Nauras Daraghmeh 2024. ARMS-MBON 18S rRNA and COI gene metabarcoding: scanning for non-indigenous species . **protocols.io https://dx.doi.org/10.17504/protocols.io.n92ldmmmnl5b/v1**

Protocol status: Working
We use this protocol and it's working

Created: August 16, 2023

Last Modified: October 28, 2024

Protocol Integer ID: 86559

Funders Acknowledgement:
MARCO BOLO
Grant ID: 101082021

Abstract

This workflow details how COI and 18S rRNA gene raw amplicon sequencing data from the European ARMS programme (ARMS-MBON) can be processed bioinformatically to generate read count and taxonomy tables of molecular operational taxonomic units (mOTUs) for the identification of (marine) non-indigenuous species (NIS). However, the end products may also be used for any other diversity analyses that suit the user. The pipeline may also be adjusted to work with amplicon sequence variants (ASVs) instead of mOTU by omitting and ajdusting certain steps.

The data used here comprise all publicly available COI and 18S sequencing data from ARMS-MBON as of February 2024.

Processes described in this pipeline were executed on Unix and Windows OS. Certain steps (especially software installations etc.) may differ when run on different operating systems. Some computationally intensive steps were run on a high-performance computing cluster. This is noted in the respective section of this workflow.

Note that in this workflow, separate directories were created for each marker gene. Make sure that the input files required (i.e., the files produced in the correspoding preceding step) are in the respective directory.

References to all data, software, packages and databases used in this workflow (please cite any of the tools used in your analysis):

ARMS-MBON (Obst *et al.*, 2020)

R v4.1.0 and v4.3.1 (R Core Team, 2021, 2023) (v4.3.1 was used for *dada2* processing and COI numt-removal)

RStudio 2022.07.1 (RStudio Team, 2022)

cutadapt v4.5 (Martin, 2011)

Git v2.37.3 (Chacon & Straub, 2014)

Python v3.11.4 (Van Rossum & Drake, 2009)

MACSE v2.05 (Ranwez *et al.*, 2018)

swarm v3.0.0 (Mahé et al., 2015)

NCBI BLAST (Johnson *et al.*, 2008)

BLAST+ release 2.11.0 (Camacho *et al.*, 2009)

BOLD *(*Ratnasingham & Hebert, 2007)

BOLDigger-commandline v2.2.1 (Buchner & Leese, 2020)

SeqKit v2.5.1 (Shen *et al.*, 2016)

MIDORI2 (Leray *et al.*, 2022)

MIDORI2 webserver (Leray *et al.*, 2018)

GenBank release 257 (Benson *et al.*, 2012)

RDP classifier (Wang *et al.*, 2007)

Silva taxonomic training data formatted for DADA2 (Callahan, 2018) (**Silva v132**; Quast *et al.*, 2013)

SILVA v128 and v132 dada2 formatted 18s 'train sets' (Morien & Parfrey, 2018) (**Silva v128 and v132**; Quast *et al.*, 2013)

Protist Ribosomal Reference database (PR2) v5.0.0 (Guillou *et al.*, 2013)

World Register of Marine Species (WoRMS) (Ahyong *et al.*, 2023)

World Register of Introduced Marine Species (WRiMS) (Rius *et al.*, 2023)

Microsoft Excel *2016* (Microsoft Corporation, 2016)

R packages:

argparse v2.2.2 (Davis, 2023)

dada2 v1.28.0 (Callahan *et al.*, 2016)

ShortRead v1.58.0 (Morgan *et al.*, 2009)

Biostrings v2.68.1 (Pagés *et al.*, 2020)

ggplot2 v3.4.2 and v3.4.3 (Wickham, 2016) (v3.4.3 was used in the *dada2* workflow to plot read quality profiles)

ensembleTax v1.2.2 (Catlett *et al.*, 2023)

tidyr v1.3.0 (Wickham *et al.*, 2023)

dplyr v1.0.9 and v1.1.3 (Wickham *et al.*, 2022, 2023) (v1.1.3 was used during COI numt-removal)

stringr v1.5.0 (Wickham, 2022)

devtools v2.4.3 (Wickham *et al.*, 2021)

hiReadsProcessor v1.29.1 and v1.36.0 (Malani, 2021) (v1.36.0 was used during COI numt-removal)

seqinr v4.2.30 (Charif & Lobry, 2007)

remotes v2.4.2 (Csárdi *et al.*, 2021)

LULU v0.1.0 (Frøslev *et al.*, 2017)

readxl v1.4.0 (Wickham & Bryan, 2022)

phyloseq v1.36.0 (McMurdie & Holmes, 2013)

vegan v2.6.2 (Oksanen *et al.*, 2023)

ggpubr v0.4.0 (Kassambara, 2020)

data.table v1.14.2 (Dowle & Srinivasan, 2021)

xlsx v0.6.5 (Dragulescu & Arendt, 2020)

plyr v1.8.7 (Wickham, 2011)

geosphere v1.5.18 (Hijmans 2022)

Bibliography:

Ahyong, S., Boyko, C. B., Bailly, N., Bernot, J., Bieler, R., Brandão, S. N., Daly, M., De Grave, S., Gofas, S., Hernandez, F., Hughes, L., Neubauer, T. A., Paulay, G., Boydens, B., Decock, W., Dekeyzer, S., Vandepitte, L., Vanhoorne, B., Adlard, R., … Zullini, A. (2023). *World Register of Marine Species (WoRMS)*. WoRMS Editorial Board. https://www.marinespecies.org

Benson, D. A., Cavanaugh, M., Clark, K., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J., & Sayers, E. W. (2012). GenBank. *Nucleic Acids Research*, *41*(D1), D36–D42. https://doi.org/10.1093/nar/gks1195

Buchner, D., & Leese, F. (2020). BOLDigger – a Python package to identify and organise sequences with the Barcode of Life Data systems. *Metabarcoding and Metagenomics 4: E53535*, *4*, e53535-. https://doi.org/10.3897/MBMG.4.53535

Callahan, B. (2018). *Silva taxonomic training data formatted for DADA2 (Silva version 132)*. Zenodo. https://doi.org/10.5281/zenodo.1172783

Callahan, B. J., McMurdie, P. J., Rosen, M. J., Han, A. W., Johnson, A. J. A., & Holmes, S. P. (2016). DADA2: High-resolution sample inference from Illumina amplicon data. *Nature Methods 2016 13:7*, *13*(7), 581–583. https://doi.org/10.1038/nmeth.3869

Camacho, C., Coulouris, G., Avagyan, V., Ma, N., Papadopoulos, J., Bealer, K., & Madden, T. L. (2009). BLAST+: Architecture and applications. *BMC Bioinformatics*, *10*(1), 1–9. https://doi.org/10.1186/1471-2105-10-421/FIGURES/4

Catlett, D., Son, K., & Liang, C. (2023). *ensembleTax: Ensemble Taxonomic Assignments of Amplicon Sequencing Data*.

Chacon, S., & Straub, B. (2014). *Pro git*. Apress.

Charif, D., & Lobry, J. R. (2007). SeqinR 1.0-2: a contributed package to the R project for statistical computing devoted to biological sequences retrieval and analysis. In U. Bastolla, M. Porto, H. E. Roman, & M. Vendruscolo (Eds.), *Structural approaches to sequence evolution: Molecules, networks, populations* (pp. 207–232). Springer Verlag.

Csárdi, G., Hester, J., Wickham, H., Chang, W., Morgan, M., & Tenenbaum, D. (2021). *remotes: R Package Installation from Remote Repositories, Including "GitHub."* https://cran.r-project.org/package=remotes

Davis, T. L. (2023). *argparse: Command Line Optional and Positional Argument Parser*. https://cran.r-project.org/package=argparse

Dowle, M., & Srinivasan, A. (2021). *data.table: Extension of `data.frame`*. https://cran.r-project.org/package=data.table

Dragulescu, A., & Arendt, C. (2020). *xlsx: Read, Write, Format Excel 2007 and Excel 97/2000/XP/2003 Files*. https://cran.r-project.org/package=xlsx

Frøslev, T. G., Kjøller, R., Bruun, H. H., Ejrnæs, R., Brunbjerg, A. K., Pietroni, C., & Hansen, A. J. (2017). Algorithm for post-clustering curation of DNA amplicon data yields reliable biodiversity estimates. *Nature Communications 2017 8:1*, *8*(1), 1–11. https://doi.org/10.1038/s41467-017-01312-x

Guillou, L., Bachar, D., Audic, S., Bass, D., Berney, C., Bittner, L., Boutte, C., Burgaud, G., De Vargas, C., Decelle, J., Del Campo, J., Dolan, J. R., Dunthorn, M., Edvardsen, B., Holzmann, M., Kooistra, W. H. C. F., Lara, E., Le Bescot, N., Logares, R., … Christen, R. (2013). The Protist Ribosomal Reference database (PR2): a catalog of unicellular eukaryote Small Sub-Unit rRNA sequences with curated taxonomy. *Nucleic Acids Research*, *41*(D1), D597–D604. https://doi.org/10.1093/NAR/GKS1160

Hijmans, R. J. (2022). *geosphere: Spherical Trigonometry*. https://cran.r-project.org/package=geosphere

Johnson, M., Zaretskaya, I., Raytselis, Y., Merezhuk, Y., McGinnis, S., & Madden, T. L. (2008). NCBI BLAST: a better web interface. *Nucleic Acids Research*, *36*(Web Server), W5–W9. https://doi.org/10.1093/nar/gkn201

Kassambara, A. (2020). *ggpubr:*

"ggplot2" Based Publication Ready Plots. R package version 0.4.0. https://cran.r-project.org/package=ggpubr

Leray, M., Ho, S. L., Lin, I. J., & Machida, R. J. (2018). MIDORI server: a webserver for taxonomic assignment of unknown metazoan mitochondrial-encoded sequences using a curated database. *Bioinformatics*, *34*(21), 3753–3754. https://doi.org/10.1093/BIOINFORMATICS/BTY454

Leray, M., Knowlton, N., & Machida, R. J. (2022). MIDORI2: A collection of quality controlled, preformatted, and regularly updated reference databases for taxonomic assignment of eukaryotic mitochondrial sequences. *Environmental DNA*, *4*(4), 894–907. https://doi.org/10.1002/EDN3.303

Mahé, F., Rognes, T., Quince, C., de Vargas, C., & Dunthorn, M. (2015). Swarmv2: Highly-scalable and high-resolution amplicon clustering. *PeerJ*, *2015*(12), e1420. https://doi.org/10.7717/PEERJ.1420/SUPP-1

Malani, N. V. (2021). *hiReadsProcessor: Functions to process LM-PCR reads from 454/Illumina data*.

Martin, M. (2011). Cutadapt removes adapter sequences from high-throughput sequencing reads. *EMBnet.Journal*, *17*(1), 10–12. https://doi.org/10.14806/ej.17.1.200

McMurdie, P. J., & Holmes, S. (2013). phyloseq: An R Package for Reproducible Interactive Analysis and Graphics of Microbiome Census Data. *PLOS ONE*, *8*(4), e61217. https://doi.org/10.1371/JOURNAL.PONE.0061217

Microsoft Corporation (2016). *Microsoft Excel*, Available at: **https://office.microsoft.com/excel**

Morgan, M., Anders, S., Lawrence, M., Aboyoun, P., Pagès, H., & Gentleman, R. (2009). ShortRead: a Bioconductor package for input, quality assessment and exploration of high-throughput sequence data. *Bioinformatics*, *25*, 2607–2608. https://doi.org/10.1093/bioinformatics/btp450

Morien, E., & Parfrey, L. W. (2018). *SILVA v128 and v132 dada2 formatted 18s "train sets."* Zenodo. https://doi.org/10.5281/zenodo.1447330

Obst, M., Exter, K., Allcock, A. L., Arvanitidis, C., Axberg, A., Bustamante, M., Cancio, I., Carreira-Flores, D., Chatzinikolaou, E., Chatzigeorgiou, G., Chrismas, N., Clark, M. S., Comtet, T., Dailianis, T., Davies, N., Deneudt, K., de Cerio, O. D., Fortič, A., Gerovasileiou, V., … Pavloudi, C. (2020). A Marine Biodiversity Observation Network for Genetic Monitoring of Hard-Bottom Communities (ARMS-MBON). *Frontiers in Marine Science*, *7*, 1031. https://doi.org/10.3389/FMARS.2020.572680/BIBTEX

Pagès, H., Aboyoun, P., Gentleman, R., & DebRoy, S. (2020). *Biostrings: Efficient manipulation of biological strings*. https://bioconductor.org/packages/Biostrings

Quast, C., Pruesse, E., Yilmaz, P., Gerken, J., Schweer, T., Yarza, P., Peplies, J., & Glöckner, F. O. (2013). The SILVA ribosomal RNA gene database project: improved data processing and web-based tools. *Nucleic Acids Research*, *41*(D1), D590–D596. https://doi.org/10.1093/NAR/GKS1219

R Core Team. (2020, 2021). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.r-project.org/

Ranwez, V., Douzery, E. J. P., Cambon, C., Chantret, N., & Delsuc, F. (2018). MACSE v2: Toolkit for the Alignment of Coding Sequences Accounting for Frameshifts and Stop Codons. *Molecular Biology and Evolution*, *35*(10), 2582–2584. https://doi.org/10.1093/MOLBEV/MSY159

Ratnasingham, S., & Hebert, P. D. N. (2007). bold: The Barcode of Life Data System (http://www.barcodinglife.org). *Molecular Ecology Notes*, *7*(3), 355–364. https://doi.org/10.1111/J.1471-8286.2007.01678.X

Rius, M., Ahyong, S., Bieler, R., Boudouresque, C., Costello, M. J., Downey, R., Galil, B. S., Gollasch, S., Hutchings, P., Kamburska, L., Katsanevakis, S., Kupriyanova, E., Lejeusne, C., Marchini, A., Occhipinti, A., Pagad, S., Panov, V. E., Poore, G. C. B., Robinson, T. B., … Zhan, A. (2023). *World Register of Introduced Marine Species (WRiMS)*. WoRMS Editorial Board. https://www.marinespecies.org/introduced

RStudio Team. (2022). *RStudio: Integrated Development Environment for R*. RStudio, PBC. http://www.rstudio.com/

Shen, W., Le, S., Li, Y., & Hu, F. (2016). SeqKit: A Cross-Platform and Ultrafast Toolkit for FASTA/Q File Manipulation. *PLOS ONE*, *11*(10), e0163962. https://doi.org/10.1371/journal.pone.0163962

Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace.

Wang, Q., Garrity, G. M., Tiedje, J. M., & Cole, J. R. (2007). Naïve Bayesian classifier for rapid assignment of rRNA sequences into the new bacterial taxonomy. *Applied and Environmental Microbiology*, *73*(16), 5261–5267. https://doi.org/10.1128/AEM.00062-07/SUPPL_FILE/SUMMARY_BYHIERARCHY.ZIP

Wickham, H. (2011). The Split-Apply-Combine Strategy for Data Analysis. *Journal of Statistical Software*, *40*(1), 1–29. https://www.jstatsoft.org/v40/i01/

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org

Wickham, H. (2022). *stringr: Simple, Consistent Wrappers for Common String Operations*. https://cran.r-project.org/package=stringr

Wickham, H., & Bryan, J. (2022). *readxl: Read Excel Files*. https://cran.r-project.org/package=readxl

Wickham, H., François, R., Henry, L., & Müller, K. (2022, 2023). *dplyr: A Grammar of Data Manipulation*. R package version 1.1.2. https://cran.r-project.org/package=dplyr

Wickham, H., Hester, J., Chang, W., & Bryan, J. (2021). *devtools: Tools to Make Developing R Packages Easier*. https://cran.r-project.org/package=devtools

Wickham, H., Vaughan, D., & Girlich, M. (2023). *tidyr: Tidy Messy Data*. https://cran.r-project.org/package=tidyr

Obtain ARMS-MBON 18S and COI metabarcoding data

1 Information on all 18S and COI sequencing data of ARMS MBON available as of February 2024 were obtained from the respective files of the ARMS MBON data_workspace GitHub repo:

https://github.com/arms-mbon/data_workspace/blob/main/qualitycontrolled_data/combined/combined_OmicsData.csv
https://github.com/arms-mbon/data_workspace/blob/main/qualitycontrolled_data/combined/combined_SamplingEventData.csv
https://github.com/arms-mbon/data_workspace/blob/main/qualitycontrolled_data/combined/demultiplexing_details_OmicsData.csv

The accession numbers for genetic data deposited on the European Nucleotide Archive (ENA) are found in the *Gene_COI* and *Gene_18S* columns of the *combined_Omicsdata.csv* file. Accession numbers of negative control samples for each sequencing run are found in columns *Gene_COI_negative_control* and *Gene_18S_negative_control.* Info on which accession stems from which sequencing run and how samples where demultiplexed post-sequencing are found in *demultiplexing_details_OmicsData.csv.* Based on these files, separate files for COI and 18S were generated containing all accession numbers of each marker gene, separated by sequencing runs. The COI and 18S files containing the respective ENA numbers are found below. As ENA accession numbers of Run_7 (i.e., the sequencing batch of August 2023) contained eight digits instead of seven digits as was the case for all previous sequencing runs, two separate files were generated for each gene:

COI_ENA_accessions.txt 3KB
COI_ENA_accessions_8digits.txt 1KB
18S_ENA_accessions.txt 3KB
18S_ENA_accessions_8digits.txt 1KB

All fastq.gz files were downloaded from ENA in February 2024. For runs listed in the XX_ENA_accessions.txt file (i.e., for accessions with seven digits), the *R* script below (ENADownload.R) was executed via command line using the .txt files as input :

Command

ENADownload.R

```
#!/usr/bin/env Rscript

# List of packages you will need
required_packages <- c('argparse')

# Determine already installed packages
installed_packages <- installed.packages()[, 'Package']

# Loop through required packages
for (package in required_packages) {

  if (!(package %in% installed_packages)) {  # If package not
installed...
    options(repos = "https://cran.rstudio.com/")  # ...set the CRAN
mirror...
    install.packages(package)                     # ...and download
the package.
  }

  suppressPackageStartupMessages(library(package, character.only =
TRUE))  # Load package silently.
}


########################################
## PARSING THE COMMAND LINE ARGUMENTS ##
########################################

# Initialize command line argument parser
parser <- ArgumentParser(description = 'DOWNLOAD ENA ACCESSIONS')

# All of your command line arguments
parser$add_argument('-f', '--file', metavar = 'fileName', type =
'character', required = TRUE, help = 'Specify the file that contains
the ENA accessions.')
parser$add_argument('-d', '--directory', metavar = 'directory', type
= 'character', required = TRUE, help = 'Specify the folder that
should be downloaded into.')

# Parse the arguments
```

```
args <- parser$parse_args()

# Access the arguments
ENAFile <- args$file
directory <- args$directory


##########################
## FILE EXISTENCE CHECK ##
##########################

# Check if your file exists
if(!file.exists(ENAFile)){ # If the file does not exist...
  stop(paste("Error:", ENAFile, "not found.")) # ... stop the script.
}


############################
## FILE CONTENTS HANDLING ##
############################

# Read the contents of the ENA file
lines <- readLines(ENAFile, warn = F)

# Make an empty variable that, later on, will contain information
about the sequencing runs and their samples
runs_samples <- list()

# Make an empty variable that, later on, will help us remember what
sequencing run we are currently handling
current_run <- NULL

# Loop over all of the ENA file lines
for(line in lines){

  # Remove all of the leading and trailing white space characters
(spaces, tab and newline)
  line <- trimws(line)

  # Remove all of the other white space characters (spaces)
  line <- gsub(pattern = ' ', replacement = '', line)

  # Determine what the first character of the line is
  firstCharacter = substr(line,1,1)

  if(firstCharacter == '>'){                                  # If the
```

```
first character is a >...
    current_run <- sub(pattern = '>', replacement = '', line) # ...
this line is the name of the current sequencing run...
    runs_samples[[current_run]] <- c()                        # ...
this line is the key of a vector that will, later on, contain its
samples.
  }

  else if(line == ''){ # If the line contains no information...
    next               # ... skip this line.
  }

  else{
  # If none of the conditions is valid...
    runs_samples[[current_run]] = c(runs_samples[[current_run]],
line)   # ... the line is a sample name that is added to its
corresponding sequencing run
  }
}



####################################
## FETCHING ONLINE ENA INFORMATION ##
####################################

# Loop over all sequencing runs
for(run in names(runs_samples)){

  # Create the path into which the samples of the current sequencing
run should be downloaded
  path.download <- file.path(directory, run)

  if(!dir.exists(path.download)){ # If the directory does not exist
yet...
    dir.create(path.download)     # ... create the directory.
  }

  # Loop over all the samples belonging to the current sequencing run.
  for(sample in runs_samples[[run]]){

    # Imagine that our current sample is ERR4914118
    # The link for this ENA sample is
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR491/008/ERR4914118/ERR4914118_1.
fastq.gz
    # You can see that there is a 6 letter code in this link (ERR491)

    # We extract this 6 character code from the sample name
```

```
    # We extract this 6 character code from the sample name
    six_letter_code <- substr(sample, start = 1, stop = 6)

    # You can also see that there is a 1 character code in this link,
preceded by '00' (008)

    # We extact this 1 number code
    one_letter_code <- substr(sample, start = nchar(sample), stop =
nchar(sample))

    # We can see the forward read filename in the link
(ERR4914118_1.fastq.gz), so we construct this one
    fwd_file_name = paste0(sample, '_1.fastq.gz')

    # There is also a reverse read filename in the link for the
reverse read file, so we construct this one
    rev_file_name = paste0(sample, '_2.fastq.gz')

    # Construct the entire link for the forward read
    url_fwd = paste0('ftp://ftp.sra.ebi.ac.uk/vol1/fastq/',
six_letter_code, '/', '00', one_letter_code, '/', sample, '/',
fwd_file_name)

    # Construct the entire link for the reverse read
    url_rev = paste0('ftp://ftp.sra.ebi.ac.uk/vol1/fastq/',
six_letter_code, '/', '00', one_letter_code, '/', sample, '/',
rev_file_name)

    # Define the location and filename for the files that will be
downloaded from ENA
    dest_fwd <- file.path(path.download, fwd_file_name)
    dest_rev <- file.path(path.download, rev_file_name)

    # Define how many times we can attempt to download an ENA
accession
    max_retries <- 3

    # Define a variable that defines the number of current retries
    retry_count <- 0

    # Define a variable that will specify if the download of the ENA
file was succesful or not
    download_success <- FALSE

    while (!download_success && retry_count < max_retries) { # Keep
on trying as long as downloading did not succeed (download_succes ==
FALSE) and the maximum amount of retries is lower than 3
```

```
      retry_count <- retry_count + 1                          # Add 1
to the amount of current retries
      tryCatch({
        download.file(url_fwd, dest_fwd)                      #
Download the information from the url to the defined destination
        download_success <- TRUE                              # Mark
the download as a succes
      }, error = function(e) {                                # If
something within the tryCatch statement gave an error...
        message(paste("Error downloading file, retrying (",
retry_count, "/", max_retries, ")")) # ... print an error message...
        Sys.sleep(10)
                            # ... and wait 10 seconds.
      })
    }

    if (!download_success) { # If the download was not a succes
(download_succes == FALSE)...
      stop("Failed to download file after", max_retries, "attempts")
# stop the script and mention what accession did not work.
    }


    # The exact same thing as above, but for the reverse reads
    max_retries <- 3
    retry_count <- 0
    download_success <- FALSE

    while (!download_success && retry_count < max_retries) {
      retry_count <- retry_count + 1
      tryCatch({
        download.file(url_rev, dest_rev)
        download_success <- TRUE
      }, error = function(e) {
        message(paste("Error downloading file, retrying (",
retry_count, "/", max_retries, ")"))
        Sys.sleep(10)
      })
    }

    if (!download_success) {
      stop("Failed to download file after", max_retries, "attempts")
    }
  }
}
```

For sequencing runs listed in the XX_ENA_accessions_8digits.txt file (i.e., for accessions with eight digits), the *R* script below (ENADownload_8digits.R) was executed via command line using the *_8digits.txt* files as input :

Command

ENADownload_8digits.R script

```
#!/usr/bin/env Rscript

# List of packages you will need
required_packages <- c('argparse')

# Determine already installed packages
installed_packages <- installed.packages()[, 'Package']

# Loop through required packages
for (package in required_packages) {

  if (!(package %in% installed_packages)) {  # If package not
installed...
    options(repos = "https://cran.rstudio.com/")  # ...set the CRAN
mirror...
    install.packages(package)                     # ...and download
the package.
  }

  suppressPackageStartupMessages(library(package, character.only =
TRUE))  # Load package silently.
}


########################################
## PARSING THE COMMAND LINE ARGUMENTS ##
########################################

# Initialize command line argument parser
parser <- ArgumentParser(description = 'DOWNLOAD ENA ACCESSIONS')

# All of your command line arguments
parser$add_argument('-f', '--file', metavar = 'fileName', type =
'character', required = TRUE, help = 'Specify the file that contains
the ENA accessions.')
parser$add_argument('-d', '--directory', metavar = 'directory', type
= 'character', required = TRUE, help = 'Specify the folder that
should be downloaded into.')

# Parse the arguments
```

```
args <- parser$parse_args()

# Access the arguments
ENAFile <- args$file
directory <- args$directory


##########################
## FILE EXISTENCE CHECK ##
##########################

# Check if your file exists
if(!file.exists(ENAFile)){ # If the file does not exist...
  stop(paste("Error:", ENAFile, "not found.")) # ... stop the script.
}


############################
## FILE CONTENTS HANDLING ##
############################

# Read the contents of the ENA file
lines <- readLines(ENAFile, warn = F)

# Make an empty variable that, later on, will contain information
about the sequencing runs and their samples
runs_samples <- list()

# Make an empty variable that, later on, will help us remember what
sequencing run we are currently handling
current_run <- NULL

# Loop over all of the ENA file lines
for(line in lines){

  # Remove all of the leading and trailing white space characters
(spaces, tab and newline)
  line <- trimws(line)

  # Remove all of the other white space characters (spaces)
  line <- gsub(pattern = ' ', replacement = '', line)

  # Determine what the first character of the line is
  firstCharacter = substr(line,1,1)

  if(firstCharacter == '>'){                                  # If the
```

```
first character is a >...
    current_run <- sub(pattern = '>', replacement = '', line) # ...
this line is the name of the current sequencing run...
    runs_samples[[current_run]] <- c()                        # ...
this line is the key of a vector that will, later on, contain its
samples.
  }

  else if(line == ''){ # If the line contains no information...
    next               # ... skip this line.
  }

  else{
  # If none of the conditions is valid...
    runs_samples[[current_run]] = c(runs_samples[[current_run]],
line)   # ... the line is a sample name that is added to its
corresponding sequencing run
  }
}



#####################################
## FETCHING ONLINE ENA INFORMATION ##
#####################################

# Loop over all sequencing runs
for(run in names(runs_samples)){

  # Create the path into which the samples of the current sequencing
run should be downloaded
  path.download <- file.path(directory, run)

  if(!dir.exists(path.download)){ # If the directory does not exist
yet...
    dir.create(path.download)     # ... create the directory.
  }

  # Loop over all the samples belonging to the current sequencing run.
  for(sample in runs_samples[[run]]){

    # Imagine that our current sample is ERR12541385
    # The link for this ENA sample is
ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR125/085/ERR12541385/ERR12541385_
1.fastq.gz
    # You can see that there is a 6 letter code in this link (ERR125)

    # We extract this 6 character code from the sample name
```

```
    # We extract this 6 character code from the sample name
    six_letter_code <- substr(sample, start = 1, stop = 6)

    # You can also see that there is a 2 character code in this link,
preceded by '0' (085)

    # We extact this 2 number code
    two_letter_code <- substr(sample, start = nchar(sample)-1, stop =
nchar(sample))

    # We can see the forward read filename in the link
(ERR12541385_1.fastq.gz), so we construct this one
    fwd_file_name = paste0(sample, '_1.fastq.gz')

    # There is also a reverse read filename in the link for the
reverse read file, so we construct this one
    rev_file_name = paste0(sample, '_2.fastq.gz')

    # Construct the entire link for the forward read
    url_fwd = paste0('ftp://ftp.sra.ebi.ac.uk/vol1/fastq/',
six_letter_code, '/', '0', two_letter_code, '/', sample, '/',
fwd_file_name)

    # Construct the entire link for the reverse read
    url_rev = paste0('ftp://ftp.sra.ebi.ac.uk/vol1/fastq/',
six_letter_code, '/', '0', two_letter_code, '/', sample, '/',
rev_file_name)

    # Define the location and filename for the files that will be
downloaded from ENA
    dest_fwd <- file.path(path.download, fwd_file_name)
    dest_rev <- file.path(path.download, rev_file_name)

    # Define how many times we can attempt to download an ENA
accession
    max_retries <- 3

    # Define a variable that defines the number of current retries
    retry_count <- 0

    # Define a variable that will specify if the download of the ENA
file was succesful or not
    download_success <- FALSE

    while (!download_success && retry_count < max_retries) { # Keep
on trying as long as downloading did not succeed (download_succes ==
FALSE) and the maximum amount of retries is lower than 3
```

```
      retry_count <- retry_count + 1                          # Add 1
to the amount of current retries
      tryCatch({
        download.file(url_fwd, dest_fwd)                      #
Download the information from the url to the defined destination
        download_success <- TRUE                              # Mark
the download as a succes
      }, error = function(e) {                                # If
something within the tryCatch statement gave an error...
        message(paste("Error downloading file, retrying (",
retry_count, "/", max_retries, ")")) # ... print an error message...
        Sys.sleep(10)
                            # ... and wait 10 seconds.
      })
    }

    if (!download_success) { # If the download was not a succes
(download_succes == FALSE)...
      stop("Failed to download file after", max_retries, "attempts")
# stop the script and mention what accession did not work.
    }


    # The exact same thing as above, but for the reverse reads
    max_retries <- 3
    retry_count <- 0
    download_success <- FALSE

    while (!download_success && retry_count < max_retries) {
      retry_count <- retry_count + 1
      tryCatch({
        download.file(url_rev, dest_rev)
        download_success <- TRUE
      }, error = function(e) {
        message(paste("Error downloading file, retrying (",
retry_count, "/", max_retries, ")"))
        Sys.sleep(10)
      })
    }

    if (!download_success) {
      stop("Failed to download file after", max_retries, "attempts")
    }
  }
}
```

When running the download scripts via command line, *-f* points to file containing the ENA accession numbers separated by sequencing run and *-d* sets the directory path the fastq files will be downloaded to (create the *fastq_files* directories prior to this; the sub-directories for the fastq files of each sequencing run will be created automatically within the *fastq_files* directories):

Command

Download fastq files from ENA

```
# Make sure Rscript.exe is in the environment variable PATH or call
it directly with its path as shown below

# COI
cd ~/COI
"C:/Program Files/R/R-4.3.1/bin/Rscript.exe" ~/ENADownload.R -f
COI_ENA_accessions.txt -d ~/COI/fastq_files
"C:/Program Files/R/R-4.3.1/bin/Rscript.exe" ~/ENADownload_8digits.R -
f COI_ENA_accessions_8digits.txt -d ~/COI/fastq_files

# 18S
cd ~/18S
"C:/Program Files/R/R-4.3.1/bin/Rscript.exe" ~/ENADownload.R -f
18S_ENA_accessions.txt -d ~/18S/fastq_files
"C:/Program Files/R/R-4.3.1/bin/Rscript.exe" ~/ENADownload_8digits.R -
f 18S_ENA_accessions_8digits.txt -d ~/18S/fastq_files
```

In case of problems, the download script will terminate with an error message. This was also case here. Sometimes, this is caused by issues when files on ENA are generated based on the originally submitted ones (see the different files available for each accession number under *Submitted files: FTP* and *Generated FASTQ files: FTP*). For us, this was the case for the 18S accession ERR7125542 (problem persisted as of February 2024, and it is not clear when ENA will fix this). There were no files under *Generated FASTQ files: FTP* for this accession. We removed this accession from 18S_ENA_accessions.txt (this accession is still included in the file provided above) and ran the download script again for 18S. Then, we manually downloaded the forward and reverse read files provided under *Submitted files: FTP* of this accession

(https://www.ebi.ac.uk/ena/browser/view/ERR7125542). The downloaded files were renamed to ERR7125542_1.fastq.gz and ERR7125542_2.fastq.gz and placed in the respective directory with the other downloaded files of this sequencing run.
There also seemed to be an incorrect read pairing in all of the 18S *Generated FASTQ files* (as of February 2024) of Run_1 (sequencing run July 2019). This became clear later on in the pipeline during primer removal using *cutadapt* (see below). We removed these files, manually downloaded all files of this run provided under *Submitted files: FTP* and manually renamed them with their accession numbers as described above for the other problematic 18S accession. The *submitted* files did not show the read-pairing issue.

Primer trimming and amplicon sequence variant (ASV) inference using *cutadapt* and *dada2*

2 The downloaded fastq.gz files contain reads which were demultiplexed with two different strategies after *MiSeq* sequencing. The combined_OmicsData.csv file provided above holds information on this in the Gene_COI_demultiplexed and Gene_18S_demultiplexed columns. Sequencing reads of some runs were demultiplexed based on the *Illumina MiSeq* library indices, while others were demultiplexed based on these indices as well as with *cutadapt* based on the respective PCR primers sequences. In the former case, reads still contain the PCR primer sequences, while in the latter case, reads are already devoid of these sequences. See the table below for information on which demultiplexing strategy was applied on the reads of each sequencing run. Note *: according to the combined_OmicsData.csv file, 18S reads of Run_1 and Run_3 were demultiplexed based on *MiSeq* indices as well as PCR primer sequences. However, after checking these reads, they still contained primer sequences in certain orientations. So reads of those runs were considered as being demultiplexed based on *MiSeq* indices only.

Run	COI	18S
Run_1	MiSeq (indices)	MiSeq (indices)*
Run_2	MiSeq (indices)	MiSeq (indices)
Run_3	MiSeq (indices) / cutadapt (primers)	MiSeq (indices)*
Run_4	MiSeq (indices) / cutadapt (primers)	MiSeq (indices) / cutadapt (primers)
Run_5	MiSeq (indices)	MiSeq (indices)
Run_6	MiSeq (indices)	MiSeq (indices)
Run_7	MiSeq (indices) / cutadapt (primers)	MiSeq (indices) / cutadapt (primers)

Demultiplexing strategies applied on the reads of the different sequencing runs of each marker gene. For *: see explanation above.

Subsequently, the directories containing the downloaded fastq.gz files of each marker gene were placed in separate directories based on the demultiplexing applied (i.e., fastqs_normal and fastqs_cutadapt).

We ran a pipeline using *cutadapt* and *dada2* in *R* on an HPC cluster for primer trimming (or length-filtering of reads previously demultiplexed with *cutadapt*) and read filtering, denoising, merging, chimera & singleton removal and taxonomy assignment (for 18S ASVs, COI ASVs were not classified using *dada2*). It should also be feasible to run the pipeline on a personal machine with a good amount of RAM, although multithreading for certain steps is not enabled in *dada2* for Windows OS and the pipeline will take a while to run. The 18S and COI data sets were processed separately. We mainly followed the publicly available *dada2* workflows, with some alterations. See here:

https://benjjneb.github.io/dada2/tutorial.html
https://benjjneb.github.io/dada2/ITS_workflow.html (for a workflow incorporating *cutadapt*)

Cutadapt needs to be installed on your system prior running the *R* workflow, see *cutadapt* documentation https://cutadapt.readthedocs.io/en/stable/index.html.

To avoid potentially resulting biases during *dada2*'s error model estimation, sequence reads were processed separately per sequencing run up until (and including) the merging of paired reads. Subsequently, inferred amplicon sequence variants (ASVs) from all sequencing runs were merged prior to chimera removal.

For reads which still contained primer sequences, *cutadapt* was first applied with maximum mismatch of 1 and 2 bp (i.e., a maximum error rate of e = 0.05 and e = 0.1) for 18S and COI reads, respectively. A higher mismatch was allowed for COI reads due to the longer primer sequences compared to 18S and the fact that COI is a more variable protein-coding gene. Given the expected length of the COI amplicons (313 bp), primers were only trimmed from the forward and reverse reads when being present at the 5'-end of the reads in their forward orientation. Given the length variability of the 18S amplicons and potential read-through during sequencing, primers were trimmed from the forward and reverse reads when being present in their forward orientation, as well as when being present in their reverse-complement orientation. Untrimmed reads were discarded. For reads which were already demultiplexed with *cutadapt* and therefore did not contain any primer sequences, *cutadapt* was only applied to filter out reads with a length of zero bp (can occur during demultiplexing when a read was only made up of its primer sequence and was fully trimmed to zero bp).

Quality profiles were generated for a maximum of (randomly chosen) 20 samples of each sequencing run. For *dada2*'s *filterAndTrim* function, *maxEE* was set to 2 for forward and 4 for reverse reads for all sequencing datasets. Based on the quality profiles, COI reads were trimmed with *truncLen* to a length of 200 and 130 bp (forward and reverse reads, respectively) to on average retain base calls with a minimum Phred quality score of 30. Note that the

truncLen needs to be set to values so that the minimum overlap during the merging of paired reads further down the pipleine is still possible. No truncating was applied for 18S due to the length variation of amplicons of this marker gene.

10^8 bp were used for error model calculation (default value, fewer bp were used automatically if a dataset did consist of fewer than 10^8 bp in total). An overlap of 10 bp with a maximum mismatch of 1 bp was applied during merging of paired reads. The amount of reads which "survived" each step of the pipleine was recorded for each sequencing run. For all randomized steps, *R*'s base function *set.seed* was applied to allow for reproducibility (see scripts).

Below are several example scripts. For both COI and 18S, there are examples for the processing of sequencing runs with "regular" reads which still have primer sequences present (Run_1 example for 18S and COI). This script was applied for all sequencing runs with this kind of reads (see table above). When running this script for other runs, just change the name of the respective runs in the script accordingly. For COI and 18S, there also examples below for the processing of sequencing runs which are already devoid of primer sequences (see example Run_3 for COI and example Run_4 for 18S). This script was applied for all sequencing runs with this kind of reads (see table above). When running this script for other runs, just change the name of the respective runs in the script accordingly. To summarise, the following table shows which example script was applied for each run:

Run	COI	18S
Run_1	COI example Run_1	18S example Run_1
Run_2	COI example Run_1	18S example Run_1
Run_3	COI example Run_3	18S example Run_1
Run_4	COI example Run_3	18S example Run_4
Run_5	COI example Run_1	18S example Run_1
Run_6	COI example Run_1	18S example Run_1
Run_7	COI example Run_3	18S example Run_4

This table shows which of each marker gene's run was processed with what kind of example script provided below.

Command

COI: Run_1 example - primer trimming and ASV inference with cutadapt and dada2 in R until read merging

```
### dada2 COI workflow with cutadapt primer removal ###

# load / install necessary packages

if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

BiocManager::install("dada2") # if this does not work, try to install
via devtools (requires prior installation of devtools)
BiocManager::install("ShortRead")
BiocManager::install("Biostrings")

library(dada2)
library(ShortRead)
library(Biostrings)
library(ggplot2)

# directory containing the fastq.gz files

path <- "~/COI/fastqs_normal/Run_1"

list.files(path)

# generate matched lists of the forward and reverse read files, as
well as parsing out the sample name

fnFs <- sort(list.files(path, pattern = "_1.fastq.gz", full.names =
TRUE))
fnRs <- sort(list.files(path, pattern = "_2.fastq.gz", full.names =
TRUE))

# Designate sequences [including ambiguous nucleotides (base = N, Y,
W, etc.) if present) of the primers used
# The reverse COI primer jgHCO2198 contains Inosine nucleotides.
# These "I" bases are not part of IUPAC convention and are not
recognized by the packages used here. Change "I"s to "N"s.

FWD <- "GGWACWGGWTGAACWGTWTAYCCYCC"  ## forward primer sequence
REV <- "TANACYTCNGGRTGNCCRAARAAYCA"  ## reverse primer sequence
```

```
REV <- "TANACTTCNGGRTGNCCRAARAATCA"  ## reverse primer sequence

# Verify the presence and orientation of these primers in the data

allOrients <- function(primer) {
  # Create all orientations of the input sequence
  require(Biostrings)
  dna <- DNAString(primer)  # The Biostrings works w/ DNAString
objects rather than character vectors
  orients <- c(Forward = dna, Complement = complement(dna), Reverse =
reverse(dna),
               RevComp = reverseComplement(dna))
  return(sapply(orients, toString))  # Convert back to character
vector
}
FWD.orients <- allOrients(FWD)
REV.orients <- allOrients(REV)
FWD.orients
REV.orients

# Calculate number of reads containing forward and reverse primer
sequences (considering all possible primer orientations. Only exact
matches are found.).
# Only one set of paired end fastq.gz files will be checked (second
sample in this case).
# This is is sufficient, assuming all the files were created using
the same library preparation.

primerHits <- function(primer, fn) {
  # Counts number of reads in which the primer is found
  nhits <- vcountPattern(primer, sread(readFastq(fn)), fixed = FALSE)
  return(sum(nhits > 0))
}
rbind(FWD.ForwardReads = sapply(FWD.orients, primerHits, fn =
fnFs[[2]]),
      REV.ReverseReads = sapply(REV.orients, primerHits, fn =
fnRs[[2]]))

# Output:
# FWD primer should mainly be found in the forward reads in its
forward orientation.
# REV primer should mainly be found in the reverse reads in its
forward orientation.

# Use cutadapt for primer removal (prior installation of cutadapt on
your machine via python, anaconda, etc. required)
# Tell R the path to cutadapt.
```

```
# Check installed version of cutadapt.

cutadapt <- "/sw/bioinfo/cutadapt/4.5/rackham/bin/cutadapt" # CHANGE
ME to the cutadapt path on your machine
system2(cutadapt, args = "--version") # see if R recognizes cutadapt
and shows its version

# Create output filenames for the cutadapt-ed files.
# Define the parameters for the cutadapt command.
# See here for a detailed explanation of paramter settings:
https://cutadapt.readthedocs.io/en/stable/guide.html#

path.cut <- file.path(path, "cutadapt")
if(!dir.exists(path.cut)) dir.create(path.cut)
fnFs.cut <- file.path(path.cut, basename(fnFs))
fnRs.cut <- file.path(path.cut, basename(fnRs))

# Trim FWD off of R1 (forward reads) -
R1.flags <- paste0("-g", " ^", FWD)
# Trim REV off of R2 (reverse reads)
R2.flags <- paste0("-G", " ^", REV)
# Run Cutadapt
for(i in seq_along(fnFs)) {
  system2(cutadapt, args = c("-e 0.1 --discard-untrimmed", R1.flags,
R2.flags,
                             "-o", fnFs.cut[i], "-p", fnRs.cut[i], #
output files
                             fnFs[i], fnRs[i])) # input files
}

# see here for a detailed explanation of the output:
# https://cutadapt.readthedocs.io/en/stable/guide.html#cutadapt-s-
output
# Sometimes, you will see this: "WARNING: One or more of your adapter
sequences may be incomplete. Please see the detailed output above."
# This usually refers to: "WARNING: The adapter is preceded by "T"
(or any other base) extremely often. The provided adapter sequence
could be incomplete at its 3' end."
# The amplified regions and primer binding sites are usually highly
conserved, so primer sequences are often preceded by the same base.
# Cutadapt just warns us that this is the case and tells us to check
if the preceding base is indeed not part of the primer.

# Count the presence of primers in the first cutadapt-ed sample to
check if cutadapt worked as intended:
```

```
rbind(FWD.ForwardReads = sapply(FWD.orients, primerHits, fn =
fnFs.cut[[2]]),
      REV.ReverseReads = sapply(REV.orients, primerHits, fn =
fnRs.cut[[2]]))

# The primer-free sequence read files are now ready to be analyzed.
# Similar to the earlier steps of reading in FASTQ files, read in the
names of the cutadapt-ed FASTQ files.
# Apply some string manipulation to get the matched lists of forward
and reverse fastq files.

# Forward and reverse fastq filenames have the format:
cutFs <- sort(list.files(path.cut, pattern = "_1.fastq.gz",
full.names = TRUE))
cutRs <- sort(list.files(path.cut, pattern = "_2.fastq.gz",
full.names = TRUE))

# Check if forward and reverse files match:

if(length(cutFs) == length(cutRs)) print("Forward and reverse files
match. Go forth and explore")
if (length(cutFs) != length(cutRs)) stop("Forward and reverse files
do not match. Better go back and have a check")

# Extract sample names, assuming filenames have format:
get.sample.name <- function(fname) strsplit(basename(fname), "_")[[1]]
[1]
sample.names <- unname(sapply(cutFs, get.sample.name))
head(sample.names)

# Inspect read quality profiles.
# If there are more than 20 samples, grab 20 randomly

set.seed(1)

if(length(cutFs) <= 20) {
  fwd_qual_plots<-plotQualityProfile(cutFs) +
    scale_x_continuous(breaks=seq(0,300,20)) +
    scale_y_continuous(breaks=seq(0,40,5)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))+
    geom_hline(yintercept = 30)
  rev_qual_plots<-plotQualityProfile(cutRs) +
    scale_x_continuous(breaks=seq(0,300,20)) +
    scale_y_continuous(breaks=seq(0,40,5)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))+
    geom_hline(yintercept = 30)
} else {
```

```
} else {
  rand_samples <- sample(size = 20, 1:length(cutFs)) # grab 20 random
samples to plot
  fwd_qual_plots <- plotQualityProfile(cutFs[rand_samples]) +
    scale_x_continuous(breaks=seq(0,300,20)) +
    scale_y_continuous(breaks=seq(0,40,5)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))+
    geom_hline(yintercept = 30)
  rev_qual_plots <- plotQualityProfile(cutRs[rand_samples]) +
    scale_x_continuous(breaks=seq(0,300,20)) +
    scale_y_continuous(breaks=seq(0,40,5)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))+
    geom_hline(yintercept = 30)
}
fwd_qual_plots
rev_qual_plots

# Print out the forward quality plot

setwd("~/COI")

jpeg(file="COI_Run1_quality_forward.jpg",res=300, width=15, height=8,
units="in")
fwd_qual_plots
dev.off()

# Print out the reverse quality plot

jpeg(file="COI_Run1_quality_reverse.jpg",res=300, width=15, height=8,
units="in")
rev_qual_plots
dev.off()

## Filter and trim ##

# Assign filenames to the fastq.gz files of filtered and trimmed
reads.

filtFs <- file.path(path.cut, "filtered", basename(cutFs))
filtRs <- file.path(path.cut, "filtered", basename(cutRs))

# Set filter and trim parameters.

out <- filterAndTrim(cutFs, filtFs, cutRs, filtRs, truncLen
=c(200,130),maxN = 0, maxEE = c(2,4),
                     truncQ = 2, minLen = 50, rm.phix = TRUE,
compress = TRUE, multithread = T)
```

```
# Save this output as RDS file for the read tracking table created
downstream:
saveRDS(out, "filter_and_trim_out_Run1.rds")

# check how many reads remain after filtering

out

# Check if file names match

sample.names <- sapply(strsplit(basename(filtFs), "_"), `[`, 1) #
Assumes filename = samplename_XXX.fastq.gz
sample.namesR <- sapply(strsplit(basename(filtRs), "_"), `[`, 1) #
Assumes filename = samplename_XXX.fastq.gz
if(identical(sample.names, sample.namesR)) {print("Files are still
matching.....congratulations")
} else {stop("Forward and reverse files do not match.")}
names(filtFs) <- sample.names
names(filtRs) <- sample.namesR

# Estimate error models of the amplicon dataset.

set.seed(100) # set seed to ensure that randomized steps are
replicatable
errF <- learnErrors(filtFs, multithread=T)
errR <- learnErrors(filtRs, multithread=T)

# save error calculation as RDS files:

saveRDS(errF, "errF_Run1.rds")
saveRDS(errR, "errR_Run1.rds")

# As a sanity check, visualize the estimated error rates and write to
file:

plot_err_F<-plotErrors(errF, nominalQ = TRUE)
plot_err_R<-plotErrors(errR, nominalQ = TRUE)

jpeg(file="COI_Run1_error_forward.jpg")
plot_err_F
dev.off()

jpeg(file="COI_Run1_error_reverse.jpg")
plot_err_R
dev.off()
```

```
### The dada2 tutorial implements a dereplication step at this point.
### This does not seem to be necessary any more with the newer dada2
versions, according to what the developers stated in the dada2 github
forum.

# Apply the dada2's core sequence-variant inference algorithm:

# Set pool = pseudo", see https://benjjneb.github.io/dada2/pool.html

dadaFs <- dada(filtFs, err=errF, multithread=T,pool="pseudo")
dadaRs <- dada(filtRs, err=errR, multithread=T,pool="pseudo")

# Apply the sample names extracted earlier (see above) to remove the
long fastq.gz file names
names(dadaFs) <- sample.names
names(dadaRs) <- sample.names

# Save sequence-variant inference output as RDS files:

saveRDS(dadaFs, "dadaFs_Run1.rds")
saveRDS(dadaRs, "dadaRs_Run1.rds")

# Merge the forward and reverse reads.
# Adjust the minimum overlap (default = 12) and maximum mismatch
allowed if necessary.

mergers <- mergePairs(dadaFs, filtFs, dadaRs,filtRs,minOverlap =
10,maxMismatch = 1,verbose=TRUE)

saveRDS(mergers,"mergers_Run1.rds")

# Construct an amplicon sequence variant table (ASV) table
# If maxMismatch > 0 has been allowed in the mergePairs step,
# "Duplicate sequences detected and merged" may appear as output
during the sequence table creation
# This is not a problem, just ignore it.

seqtab <- makeSequenceTable(mergers)

# How many sequence variants were inferred?
dim(seqtab)

# Save sequence table

saveRDS(seqtab, "seqtab_Run1.rds")
```

```
## Track reads throughout the pipeline ##

# Get number of reads in files prior to cutadapt application

input<-countFastq(path,pattern=".gz") # get statistics from input
files
input$Sample<-rownames(input)
input$Sample<-gsub("_.*","",input$Sample) # Remove all characters
after _ (incl. _) in file names
input<-aggregate(.~Sample,input,FUN="mean") # Aggregate forward and
reverse read files

# Get number of reads from each step of dada2 pipeline

getN <- function(x) sum(getUniques(x))
track <- cbind(out, sapply(dadaFs, getN), sapply(dadaRs, getN),
sapply(mergers,getN))
colnames(track) <- c("cutadapt", "filtered", "denoisedF",
"denoisedR", "merged")
rownames(track) <- sample.names

# Combine with read numbers from input files

input<-input[order(match(input[,1],rownames(track))),]
track<-cbind(input$records,track)
colnames(track)[1]<-"input"

# Save to file

write.table(track,"track_Run1.txt",sep="\t",col.names = NA)
```

Command

18S: Run_1 example - primer trimming and ASV inference with cutadapt and dada2 in R until read merging

```
### dada2 18S workflow with cutadapt primer removal ###

# load / install necessary packages

if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

BiocManager::install("dada2") # if this does not work, try to install
via devtools (requires prior installation of devtools)
BiocManager::install("ShortRead")
BiocManager::install("Biostrings")

library(dada2)
library(ShortRead)
library(Biostrings)
library(ggplot2)

# directory containing the fastq.gz files

path <- "~/18S/fastqs_normal/Run_1"

list.files(path)

# generate matched lists of the forward and reverse read files, as
well as parsing out the sample name

fnFs <- sort(list.files(path, pattern = "_1.fastq.gz", full.names =
TRUE))
fnRs <- sort(list.files(path, pattern = "_2.fastq.gz", full.names =
TRUE))

# Designate sequences [including ambiguous nucleotides (base = N, Y,
W, etc.) if present) of the primers used

FWD <- "TGGTGCATGGCCGTTCTTAGT"  ## forward primer sequence
REV <- "CATCTAAGGGCATCACAGACC"  ## reverse primer sequence

# Verify the presence and orientation of these primers in the data
```

```
allOrients <- function(primer) {
  # Create all orientations of the input sequence
  require(Biostrings)
  dna <- DNAString(primer)  # The Biostrings works w/ DNAString
objects rather than character vectors
  orients <- c(Forward = dna, Complement = complement(dna), Reverse =
reverse(dna),
               RevComp = reverseComplement(dna))
  return(sapply(orients, toString))  # Convert back to character
vector
}
FWD.orients <- allOrients(FWD)
REV.orients <- allOrients(REV)
FWD.orients
REV.orients

# Calculate number of reads containing forward and reverse primer
sequences (considering all possible primer orientations. Only exact
matches are found.).
# Only one set of paired end fastq.gz files will be checked
(firstsample in this case).
# This is is sufficient, assuming all the files were created using
the same library preparation.

primerHits <- function(primer, fn) {
  # Counts number of reads in which the primer is found
  nhits <- vcountPattern(primer, sread(readFastq(fn)), fixed = FALSE)
  return(sum(nhits > 0))
}
rbind(FWD.ForwardReads = sapply(FWD.orients, primerHits, fn =
fnFs[[1]]),
      FWD.ReverseReads = sapply(FWD.orients, primerHits, fn =
fnRs[[1]]),
      REV.ForwardReads = sapply(REV.orients, primerHits, fn =
fnFs[[1]]),
      REV.ReverseReads = sapply(REV.orients, primerHits, fn =
fnRs[[1]]))

# Output:
# FWD primer should mainly be found in the forward reads in its
forward orientation.
# FWD primer may also be found in some of the reverse reads in its
reverse-complement orientation (due to read-through when amplicons
are short).
# REV primer may also be found in the forward reads in its reverse
complement orientation (due to read-through when amplicons are short).
```

```
# REV primer should mainly be found in the reverse reads in its
forward orientation.

# Use cutadapt for primer removal (prior installation of cutadapt on
your machine via python, anaconda, etc. required)
# Tell R the path to cutadapt.
# Check installed version of cutadapt.

cutadapt <- "/sw/bioinfo/cutadapt/4.5/rackham/bin/cutadapt" # CHANGE
ME to the cutadapt path on your machine
system2(cutadapt, args = "--version") # see if R recognizes cutadapt
and shows its version

# Create output filenames for the cutadapt-ed files.
# Define the parameters for the cutadapt command.
# See here for a detailed explanation of paramter settings:
https://cutadapt.readthedocs.io/en/stable/guide.html#

path.cut <- file.path(path, "cutadapt")
if(!dir.exists(path.cut)) dir.create(path.cut)
fnFs.cut <- file.path(path.cut, basename(fnFs))
fnRs.cut <- file.path(path.cut, basename(fnRs))

FWD.RC <- dada2:::rc(FWD)
REV.RC <- dada2:::rc(REV)
# Trim FWD and the reverse-complement of REV off of R1 (forward reads)
R1.flags <- paste("-g", FWD, "-a", REV.RC)
# Trim REV and the reverse-complement of FWD off of R2 (reverse reads)
R2.flags <- paste("-G", REV, "-A", FWD.RC)
# Run Cutadapt
for(i in seq_along(fnFs)) {
  system2(cutadapt, args = c("-e 0.05 --discard-untrimmed",R1.flags,
R2.flags, "-m",1, # -e sets the allowed error, -m 1 discards
sequences of length zero after cutadapting
                             "-n", 2, # -n 2 required to remove FWD
and REV from reads
                             "-o", fnFs.cut[i], "-p", fnRs.cut[i], #
output files
                             fnFs[i], fnRs[i])) # input files
}

# see here for a detailed explanation of the output:
# https://cutadapt.readthedocs.io/en/stable/guide.html#cutadapt-s-
output
# Often, you will see this: "WARNING: One or more of your adapter
sequences may be incomplete. Please see the detailed output above."
```

```
# This usually refers to: "WARNING: The adapter is preceded by "T"
(or any other base) extremely often. The provided adapter sequence
could be incomplete at its 3' end."
# The amplified regions and primer binding sites are usually highly
conserved, so primer sequences are often preceded by the same base.
# Cutadapt just warns us that this is the case and tells us to check
if the preceding base is indeed not part of the primer. Ignore the
warning, this is not the case.

# Count the presence of primers in the first cutadapt-ed sample as a
check if cutadapt worked:

rbind(FWD.ForwardReads = sapply(FWD.orients, primerHits, fn =
fnFs.cut[[1]]),
      FWD.ReverseReads = sapply(FWD.orients, primerHits, fn =
fnRs.cut[[1]]),
      REV.ForwardReads = sapply(REV.orients, primerHits, fn =
fnFs.cut[[1]]),
      REV.ReverseReads = sapply(REV.orients, primerHits, fn =
fnRs.cut[[1]]))


# The primer-free sequence read files are now ready to be analyzed.
# Similar to the earlier steps of reading in FASTQ files, read in the
names of the cutadapt-ed FASTQ files.
# Apply some string manipulation to get the matched lists of forward
and reverse fastq files.

# Forward and reverse fastq filenames have the format:
cutFs <- sort(list.files(path.cut, pattern = "_1.fastq.gz",
full.names = TRUE))
cutRs <- sort(list.files(path.cut, pattern = "_2.fastq.gz",
full.names = TRUE))

# Check if forward and reverse files match:

if(length(cutFs) == length(cutRs)) print("Forward and reverse files
match. Go forth and explore")
if (length(cutFs) != length(cutRs)) stop("Forward and reverse files
do not match. Better go back and have a check")

# Extract sample names, assuming filenames have format:
get.sample.name <- function(fname) strsplit(basename(fname), "_")[[1]]
[1]
sample.names <- unname(sapply(cutFs, get.sample.name))
head(sample.names)
```

```
# Inspect read quality profiles.
# If there are more than 20 samples, grab 20 randomly

set.seed(1)

if(length(cutFs) <= 20) {
  fwd_qual_plots<-plotQualityProfile(cutFs) +
    scale_x_continuous(breaks=seq(0,300,20)) +
    scale_y_continuous(breaks=seq(0,40,5)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))+
    geom_hline(yintercept = 30)
  rev_qual_plots<-plotQualityProfile(cutRs) +
    scale_x_continuous(breaks=seq(0,300,20)) +
    scale_y_continuous(breaks=seq(0,40,5)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))+
    geom_hline(yintercept = 30)
} else {
  rand_samples <- sample(size = 20, 1:length(cutFs)) # grab 20 random
samples to plot
  fwd_qual_plots <- plotQualityProfile(cutFs[rand_samples]) +
    scale_x_continuous(breaks=seq(0,300,20)) +
    scale_y_continuous(breaks=seq(0,40,5)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))+
    geom_hline(yintercept = 30)
  rev_qual_plots <- plotQualityProfile(cutRs[rand_samples]) +
    scale_x_continuous(breaks=seq(0,300,20)) +
    scale_y_continuous(breaks=seq(0,40,5)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))+
    geom_hline(yintercept = 30)
}
fwd_qual_plots
rev_qual_plots

# Print out the forward quality plot

setwd("~/18S")

jpeg(file="18S_Run1_quality_forward.jpg",res=300, width=15, height=8,
units="in")
fwd_qual_plots
dev.off()

# Print out the reverse quality plot

jpeg(file="18S_Run1_quality_reverse.jpg",res=300, width=15, height=8,
units="in")
```

```
rev_qual_plots
dev.off()

## Filter and trim ##

# Assign filenames to the fastq.gz files of filtered and trimmed
reads.

filtFs <- file.path(path.cut, "filtered", basename(cutFs))
filtRs <- file.path(path.cut, "filtered", basename(cutRs))

# Set filter and trim parameters.

out <- filterAndTrim(cutFs, filtFs, cutRs, filtRs,maxN = 0, maxEE =
c(2,4),
                     truncQ = 2, minLen = 50, rm.phix = TRUE,
compress = TRUE, multithread = T)

# Save this output as RDS file for the read tracking table created
downstream:
saveRDS(out, "filter_and_trim_out_Run1.rds")

# check how many reads remain after filtering

out

# Check if file names match

sample.names <- sapply(strsplit(basename(filtFs), "_"), `[`, 1) #
Assumes filename = samplename_XXX.fastq.gz
sample.namesR <- sapply(strsplit(basename(filtRs), "_"), `[`, 1) #
Assumes filename = samplename_XXX.fastq.gz
if(identical(sample.names, sample.namesR)) {print("Files are still
matching.....congratulations")
} else {stop("Forward and reverse files do not match.")}
names(filtFs) <- sample.names
names(filtRs) <- sample.namesR

# Estimate error models of the amplicon dataset.

set.seed(100) # set seed to ensure that randomized steps are
replicatable
errF <- learnErrors(filtFs, multithread=T)
errR <- learnErrors(filtRs, multithread=T)

# save error calculation as RDS files:
```

```
saveRDS(errF, "errF_Run1.rds")
saveRDS(errR, "errR_Run1.rds")

# As a sanity check, visualize the estimated error rates and write to
file:

plot_err_F<-plotErrors(errF, nominalQ = TRUE)
plot_err_R<-plotErrors(errR, nominalQ = TRUE)

jpeg(file="18S_Run1_error_forward.jpg")
plot_err_F
dev.off()

jpeg(file="18S_Run1_error_reverse.jpg")
plot_err_R
dev.off()

### The dada2 tutorial implements a dereplication step at this point.
### This does not seem to be necessary any more with the newer dada2
versions, according to what the developers stated in the dada2 github
forum.

# Apply the dada2's core sequence-variant inference algorithm:
# Set pool = pseudo", see https://benjjneb.github.io/dada2/pool.html

dadaFs <- dada(filtFs, err=errF, multithread=T,pool="pseudo")
dadaRs <- dada(filtRs, err=errR, multithread=T,pool="pseudo")

# Apply the sample names extracted earlier (see above) to remove the
long fastq.gz file names
names(dadaFs) <- sample.names
names(dadaRs) <- sample.names

# Save sequence-variant inference output as RDS files:

saveRDS(dadaFs, "dadaFs_Run1.rds")
saveRDS(dadaRs, "dadaRs_Run1.rds")

# Merge the forward and reverse reads.
# Adjust the minimum overlap (default = 12) and maximum mismatch
allowed if necessary.

mergers <- mergePairs(dadaFs, filtFs, dadaRs,filtRs,minOverlap =
10,maxMismatch = 1,verbose=TRUE)

saveRDS(mergers,"mergers_Run1.rds")
```

```
# Construct an amplicon sequence variant table (ASV) table
# If maxMismatch > 0 has been allowed in the mergePairs step,
# "Duplicate sequences detected and merged" may appear as output
during the sequence table creation
# This is not a problem, just ignore it.

seqtab <- makeSequenceTable(mergers)

# How many sequence variants were inferred?
dim(seqtab)

# Save sequence table

saveRDS(seqtab, "seqtab_Run1.rds")

## Track reads throughout the pipeline ##

# Get number of reads in files prior to cutadapt application

input<-countFastq(path,pattern=".gz") # get statistics from input
files
input$Sample<-rownames(input)
input$Sample<-gsub("_.*","",input$Sample) # Remove all characters
after _ (incl. _) in file names
input<-aggregate(.~Sample,input,FUN="mean") # Aggregate forward and
reverse read files

# Get number of reads from each step of dada2 pipeline

getN <- function(x) sum(getUniques(x))
track <- cbind(out, sapply(dadaFs, getN), sapply(dadaRs, getN),
sapply(mergers,getN))
colnames(track) <- c("cutadapt", "filtered", "denoisedF",
"denoisedR", "merged")
rownames(track) <- sample.names

# Combine with read numbers from input files

input<-input[order(match(input[,1],rownames(track))),]
track<-cbind(input$records,track)
colnames(track)[1]<-"input"

# Save to file

write.table(track,"track_Run1.txt",sep="\t",col.names = NA)
```

Command

COI: Run_3 example - length filtering and ASV inference with cutadapt and dada2 in R until read merging

```
### dada2 COI workflow without cutadapt primer removal ###

# load / install necessary packages

if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

BiocManager::install("dada2") # if this does not work, try to install
via devtools (requires prior installation of devtools)
BiocManager::install("ShortRead")
BiocManager::install("Biostrings")

library(dada2)
library(ShortRead)
library(Biostrings)
library(ggplot2)

# directory containing the fastq.gz files

path <- "~/COI/fastqs_cutadapt/Run_3"

list.files(path)

# generate matched lists of the forward and reverse read files, as
well as parsing out the sample name

fnFs <- sort(list.files(path, pattern = "_1.fastq.gz", full.names =
TRUE))
fnRs <- sort(list.files(path, pattern = "_2.fastq.gz", full.names =
TRUE))

# Filter reads only for length

cutadapt <- "/sw/bioinfo/cutadapt/4.5/rackham/bin/cutadapt" # CHANGE
ME to the cutadapt path on your machine
system2(cutadapt, args = "--version") # see if R recognizes cutadapt
and shows its version

# Create output filenames for the cutadapt-ed files
```

```
# Create output filenames for the cutadapt-ed files.
# Define the parameters for the cutadapt command.
# See here for a detailed explanation of paramter settings:
https://cutadapt.readthedocs.io/en/stable/guide.html#

path.cut <- file.path(path, "cutadapt")
if(!dir.exists(path.cut)) dir.create(path.cut)
fnFs.cut <- file.path(path.cut, basename(fnFs))
fnRs.cut <- file.path(path.cut, basename(fnRs))

# Run Cutadapt just for length filtering
for(i in seq_along(fnFs)) {
  system2(cutadapt, args = c("-m 1",
                             "-o", fnFs.cut[i], "-p", fnRs.cut[i], #
output files
                             fnFs[i], fnRs[i])) # input files
}

# see here for a detailed explanation of the output:
# https://cutadapt.readthedocs.io/en/stable/guide.html#cutadapt-s-
output

# The length-filtered sequence read files are now ready to be
analyzed.
# Similar to the earlier steps of reading in FASTQ files, read in the
names of the cutadapt-ed FASTQ files.
# Apply some string manipulation to get the matched lists of forward
and reverse fastq files.

# Forward and reverse fastq filenames have the format:
cutFs <- sort(list.files(path.cut, pattern = "_1.fastq.gz",
full.names = TRUE))
cutRs <- sort(list.files(path.cut, pattern = "_2.fastq.gz",
full.names = TRUE))

# Check if forward and reverse files match:

if(length(cutFs) == length(cutRs)) print("Forward and reverse files
match. Go forth and explore")
if (length(cutFs) != length(cutRs)) stop("Forward and reverse files
do not match. Better go back and have a check")

# Extract sample names, assuming filenames have format:
get.sample.name <- function(fname) strsplit(basename(fname), "_")[[1]]
[1]
sample.names <- unname(sapply(cutFs, get.sample.name))
head(sample.names)
```

```
# Inspect read quality profiles.
# If there are more than 20 samples, grab 20 randomly

set.seed(1)

if(length(cutFs) <= 20) {
  fwd_qual_plots<-plotQualityProfile(cutFs) +
    scale_x_continuous(breaks=seq(0,300,20)) +
    scale_y_continuous(breaks=seq(0,40,5)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))+
    geom_hline(yintercept = 30)
  rev_qual_plots<-plotQualityProfile(cutRs) +
    scale_x_continuous(breaks=seq(0,300,20)) +
    scale_y_continuous(breaks=seq(0,40,5)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))+
    geom_hline(yintercept = 30)
} else {
  rand_samples <- sample(size = 20, 1:length(cutFs)) # grab 20 random
samples to plot
  fwd_qual_plots <- plotQualityProfile(cutFs[rand_samples]) +
    scale_x_continuous(breaks=seq(0,300,20)) +
    scale_y_continuous(breaks=seq(0,40,5)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))+
    geom_hline(yintercept = 30)
  rev_qual_plots <- plotQualityProfile(cutRs[rand_samples]) +
    scale_x_continuous(breaks=seq(0,300,20)) +
    scale_y_continuous(breaks=seq(0,40,5)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))+
    geom_hline(yintercept = 30)
}
fwd_qual_plots
rev_qual_plots

# Print out the forward quality plot

setwd("~/COI")

jpeg(file="COI_Run3_quality_forward.jpg",res=300, width=15, height=8,
units="in")
fwd_qual_plots
dev.off()

# Print out the reverse quality plot

jpeg(file="COI_Run3_quality_reverse.jpg",res=300, width=15, height=8,
```

```
units="in")
rev_qual_plots
dev.off()

## Filter and trim ##

# Assign filenames to the fastq.gz files of filtered and trimmed
reads.

filtFs <- file.path(path.cut, "filtered", basename(cutFs))
filtRs <- file.path(path.cut, "filtered", basename(cutRs))

# Set filter and trim parameters.

out <- filterAndTrim(cutFs, filtFs, cutRs, filtRs, truncLen
=c(200,130),maxN = 0, maxEE = c(2,4),
                     truncQ = 2, minLen = 50, rm.phix = TRUE,
compress = TRUE, multithread = T)

# Save this output as RDS file for the read tracking table created
downstream:
saveRDS(out, "filter_and_trim_out_Run3.rds")

# check how many reads remain after filtering

out

# Check if file names match

sample.names <- sapply(strsplit(basename(filtFs), "_"), `[`, 1) #
Assumes filename = samplename_XXX.fastq.gz
sample.namesR <- sapply(strsplit(basename(filtRs), "_"), `[`, 1) #
Assumes filename = samplename_XXX.fastq.gz
if(identical(sample.names, sample.namesR)) {print("Files are still
matching.....congratulations")
} else {stop("Forward and reverse files do not match.")}
names(filtFs) <- sample.names
names(filtRs) <- sample.namesR

# Estimate error models of the amplicon dataset.

set.seed(100) # set seed to ensure that randomized steps are
replicatable
errF <- learnErrors(filtFs, multithread=T)
errR <- learnErrors(filtRs, multithread=T)

# save error calculation as RDS files:
```

```
# Save error calculation as RDS files:

saveRDS(errF, "errF_Run3.rds")
saveRDS(errR, "errR_Run3.rds")

# As a sanity check, visualize the estimated error rates and write to
file:

plot_err_F<-plotErrors(errF, nominalQ = TRUE)
plot_err_R<-plotErrors(errR, nominalQ = TRUE)

jpeg(file="COI_Run3_error_forward.jpg")
plot_err_F
dev.off()

jpeg(file="COI_Run3_error_reverse.jpg")
plot_err_R
dev.off()

### The dada2 tutorial implements a dereplication step at this point.
### This does not seem to be necessary any more with the newer dada2
versions, according to what the developers stated in the dada2 github
forum.

# Apply the dada2's core sequence-variant inference algorithm:

# Set pool = pseudo", see https://benjjneb.github.io/dada2/pool.html

dadaFs <- dada(filtFs, err=errF, multithread=T,pool="pseudo")
dadaRs <- dada(filtRs, err=errR, multithread=T,pool="pseudo")

# Apply the sample names extracted earlier (see above) to remove the
long fastq.gz file names
names(dadaFs) <- sample.names
names(dadaRs) <- sample.names

# Save sequence-variant inference output as RDS files:

saveRDS(dadaFs, "dadaFs_Run3.rds")
saveRDS(dadaRs, "dadaRs_Run3.rds")

# Merge the forward and reverse reads.
# Adjust the minimum overlap (default = 12) and maximum mismatch
allowed if necessary.

mergers <- mergePairs(dadaFs, filtFs, dadaRs,filtRs,minOverlap =
10,maxMismatch = 1,verbose=TRUE)
```

```
saveRDS(mergers,"mergers_Run3.rds")

# Construct an amplicon sequence variant table (ASV) table
# If maxMismatch > 0 has been allowed in the mergePairs step,
# "Duplicate sequences detected and merged" may appear as output
during the sequence table creation
# This is not a problem, just ignore it.

seqtab <- makeSequenceTable(mergers)

# How many sequence variants were inferred?
dim(seqtab)

# Save sequence table

saveRDS(seqtab, "seqtab_Run3.rds")

## Track reads throughout the pipeline ##

# Get number of reads in files prior to cutadapt application

input<-countFastq(path,pattern=".gz") # get statistics from input
files
input$Sample<-rownames(input)
input$Sample<-gsub("_.*","",input$Sample) # Remove all characters
after _ (incl. _) in file names
input<-aggregate(.~Sample,input,FUN="mean") # Aggregate forward and
reverse read files

# Get number of reads from each step of dada2 pipeline

getN <- function(x) sum(getUniques(x))
track <- cbind(out, sapply(dadaFs, getN), sapply(dadaRs, getN),
sapply(mergers,getN))
colnames(track) <- c("cutadapt", "filtered", "denoisedF",
"denoisedR", "merged")
rownames(track) <- sample.names

# Combine with read numbers from input files

input<-input[order(match(input[,1],rownames(track))),]
track<-cbind(input$records,track)
colnames(track)[1]<-"input"

# Save to file
```

```
write.table(track,"track_Run3.txt",sep="\t",col.names = NA)
```

Command

18S: Run_4 example - length filtering and ASV inference with cutadapt and dada2 in R until read merging

```
### dada2 18S workflow without cutadapt primer removal ###

# load / install necessary packages

if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

BiocManager::install("dada2") # if this does not work, try to install
via devtools (requires prior installation of devtools)
BiocManager::install("ShortRead")
BiocManager::install("Biostrings")

library(dada2)
library(ShortRead)
library(Biostrings)
library(ggplot2)

# directory containing the fastq.gz files

path <- "~/18S/fastqs_cutadapt/Run_4"

list.files(path)

# generate matched lists of the forward and reverse read files, as
well as parsing out the sample name

fnFs <- sort(list.files(path, pattern = "_1.fastq.gz", full.names =
TRUE))
fnRs <- sort(list.files(path, pattern = "_2.fastq.gz", full.names =
TRUE))

# Filter reads only for length

cutadapt <- "/sw/bioinfo/cutadapt/4.5/rackham/bin/cutadapt" # CHANGE
ME to the cutadapt path on your machine
system2(cutadapt, args = "--version") # see if R recognizes cutadapt
and shows its version

# Create output filenames for the cutadapt-ed files
```

```
# Create output filenames for the cutadapt-ed files.
# Define the parameters for the cutadapt command.
# See here for a detailed explanation of paramter settings:
https://cutadapt.readthedocs.io/en/stable/guide.html#

path.cut <- file.path(path, "cutadapt")
if(!dir.exists(path.cut)) dir.create(path.cut)
fnFs.cut <- file.path(path.cut, basename(fnFs))
fnRs.cut <- file.path(path.cut, basename(fnRs))

# Run Cutadapt just for length filtering
for(i in seq_along(fnFs)) {
  system2(cutadapt, args = c("-m 1",
                             "-o", fnFs.cut[i], "-p", fnRs.cut[i], #
output files
                             fnFs[i], fnRs[i])) # input files
}

# see here for a detailed explanation of the output:
# https://cutadapt.readthedocs.io/en/stable/guide.html#cutadapt-s-
output

# The primer-free sequence read files are now ready to be analyzed.
# Similar to the earlier steps of reading in FASTQ files, read in the
names of the cutadapt-ed FASTQ files.
# Apply some string manipulation to get the matched lists of forward
and reverse fastq files.

# Forward and reverse fastq filenames have the format:
cutFs <- sort(list.files(path.cut, pattern = "_1.fastq.gz",
full.names = TRUE))
cutRs <- sort(list.files(path.cut, pattern = "_2.fastq.gz",
full.names = TRUE))

# Check if forward and reverse files match:

if(length(cutFs) == length(cutRs)) print("Forward and reverse files
match. Go forth and explore")
if (length(cutFs) != length(cutRs)) stop("Forward and reverse files
do not match. Better go back and have a check")

# Extract sample names, assuming filenames have format:
get.sample.name <- function(fname) strsplit(basename(fname), "_")[[1]]
[1]
sample.names <- unname(sapply(cutFs, get.sample.name))
head(sample.names)
```

```
# Inspect read quality profiles.
# If there are more than 20 samples, grab 20 randomly

set.seed(1)

if(length(cutFs) <= 20) {
  fwd_qual_plots<-plotQualityProfile(cutFs) +
    scale_x_continuous(breaks=seq(0,300,20)) +
    scale_y_continuous(breaks=seq(0,40,5)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))+
    geom_hline(yintercept = 30)
  rev_qual_plots<-plotQualityProfile(cutRs) +
    scale_x_continuous(breaks=seq(0,300,20)) +
    scale_y_continuous(breaks=seq(0,40,5)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))+
    geom_hline(yintercept = 30)
} else {
  rand_samples <- sample(size = 20, 1:length(cutFs)) # grab 20 random
samples to plot
  fwd_qual_plots <- plotQualityProfile(cutFs[rand_samples]) +
    scale_x_continuous(breaks=seq(0,300,20)) +
    scale_y_continuous(breaks=seq(0,40,5)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))+
    geom_hline(yintercept = 30)
  rev_qual_plots <- plotQualityProfile(cutRs[rand_samples]) +
    scale_x_continuous(breaks=seq(0,300,20)) +
    scale_y_continuous(breaks=seq(0,40,5)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))+
    geom_hline(yintercept = 30)
}
fwd_qual_plots
rev_qual_plots

# Print out the forward quality plot

setwd("~/18S")

jpeg(file="18S_Run4_quality_forward.jpg",res=300, width=15, height=8,
units="in")
fwd_qual_plots
dev.off()

# Print out the reverse quality plot

jpeg(file="18S_Run4_quality_reverse.jpg",res=300, width=15, height=8,
units="in")
```

```
rev_qual_plots
dev.off()

## Filter and trim ##

# Assign filenames to the fastq.gz files of filtered and trimmed
reads.

filtFs <- file.path(path.cut, "filtered", basename(cutFs))
filtRs <- file.path(path.cut, "filtered", basename(cutRs))

# Set filter and trim parameters.

out <- filterAndTrim(cutFs, filtFs, cutRs, filtRs,maxN = 0, maxEE =
c(2,4),
                     truncQ = 2, minLen = 50, rm.phix = TRUE,
compress = TRUE, multithread = T)

# Save this output as RDS file for the read tracking table created
downstream:
saveRDS(out, "filter_and_trim_out_Run4.rds")

# check how many reads remain after filtering

out

# Check if file names match

sample.names <- sapply(strsplit(basename(filtFs), "_"), `[`, 1) #
Assumes filename = samplename_XXX.fastq.gz
sample.namesR <- sapply(strsplit(basename(filtRs), "_"), `[`, 1) #
Assumes filename = samplename_XXX.fastq.gz
if(identical(sample.names, sample.namesR)) {print("Files are still
matching.....congratulations")
} else {stop("Forward and reverse files do not match.")}
names(filtFs) <- sample.names
names(filtRs) <- sample.namesR

# Estimate error models of the amplicon dataset.

set.seed(100) # set seed to ensure that randomized steps are
replicatable
errF <- learnErrors(filtFs, multithread=T)
errR <- learnErrors(filtRs, multithread=T)

# save error calculation as RDS files:
```

```
saveRDS(errF, "errF_Run4.rds")
saveRDS(errR, "errR_Run4.rds")

# As a sanity check, visualize the estimated error rates and write to
file:

plot_err_F<-plotErrors(errF, nominalQ = TRUE)
plot_err_R<-plotErrors(errR, nominalQ = TRUE)

jpeg(file="18S_Run4_error_forward.jpg")
plot_err_F
dev.off()

jpeg(file="18S_Run4_error_reverse.jpg")
plot_err_R
dev.off()

### The dada2 tutorial implements a dereplication step at this point.
### This does not seem to be necessary any more with the newer dada2
versions, according to what the developers stated in the dada2 github
forum.

# Apply the dada2's core sequence-variant inference algorithm:

# Set pool = pseudo", see https://benjjneb.github.io/dada2/pool.html

dadaFs <- dada(filtFs, err=errF, multithread=T,pool="pseudo")
dadaRs <- dada(filtRs, err=errR, multithread=T,pool="pseudo")

# Apply the sample names extracted earlier (see above) to remove the
long fastq.gz file names
names(dadaFs) <- sample.names
names(dadaRs) <- sample.names

# Save sequence-variant inference output as RDS files:

saveRDS(dadaFs, "dadaFs_Run4.rds")
saveRDS(dadaRs, "dadaRs_Run4.rds")

# Merge the forward and reverse reads.
# Adjust the minimum overlap (default = 12) and maximum mismatch
allowed if necessary.

mergers <- mergePairs(dadaFs, filtFs, dadaRs,filtRs,minOverlap =
10,maxMismatch = 1,verbose=TRUE)
```

```
saveRDS(mergers,"mergers_Run4.rds")

# Construct an amplicon sequence variant table (ASV) table
# If maxMismatch > 0 has been allowed in the mergePairs step,
# "Duplicate sequences detected and merged" may appear as output
during the sequence table creation
# This is not a problem, just ignore it.

seqtab <- makeSequenceTable(mergers)

# How many sequence variants were inferred?
dim(seqtab)

# Save sequence table

saveRDS(seqtab, "seqtab_Run4.rds")

## Track reads throughout the pipeline ##

# Get number of reads in files prior to cutadapt application

input<-countFastq(path,pattern=".gz") # get statistics from input
files
input$Sample<-rownames(input)
input$Sample<-gsub("_.*","",input$Sample) # Remove all characters
after _ (incl. _) in file names
input<-aggregate(.~Sample,input,FUN="mean") # Aggregate forward and
reverse read files

# Get number of reads from each step of dada2 pipeline

getN <- function(x) sum(getUniques(x))
track <- cbind(out, sapply(dadaFs, getN), sapply(dadaRs, getN),
sapply(mergers,getN))
colnames(track) <- c("cutadapt", "filtered", "denoisedF",
"denoisedR", "merged")
rownames(track) <- sample.names

# Combine with read numbers from input files

input<-input[order(match(input[,1],rownames(track))),]
track<-cbind(input$records,track)
colnames(track)[1]<-"input"

# Save to file
```

```
write.table(track,"track_Run4.txt",sep="\t",col.names = NA)
```

Subsequently, individual ASV tables of each sequencing run were merged (COI ASVs were additionally subjected to a stringent length filtering to only retain sequences with a length of of 310, 313 or 316 bp (so the expected length of 313 bp ± one codon triplet); this was not done for 18S due to the length variation of amplicons of this marker gene) and chimeric and singleton sequences were removed. For COI, a .fasta file containing the non-singleton ASV sequences with corresponding headers, as well as an ASV count table (read abundances per sample) were generated. Taxonomy was assigned later on in the workflow (see sections below). For 18S, ASVs were classified with *dada2*'s *assignTaxonomy* function using three different reference sets (with varying taxonomic ranks) formatted for use in *dada2*: a) official *Silva* v132 containing prokayrote and eukaryote sequences (silva_nr_v132_train_set.fa.gz, https://zenodo.org/record/1172783); b) a subset of *Silva* v132 containing eukaryote sequences clustered at 99% similarity only (silva_132.18s.99_rep_set.dada2.fa.gz, https://zenodo.org/record/1447330); and c) *PR2* v5.0.0 (pr2_version_5.0.0_SSU_dada2.fasta.gz, https://github.com/pr2database/pr2database/releases). A minimum bootstrap threshold of 70 was applied. An ensemble taxonomy, ASV fasta file and ASV count table for 18S ASVs were generated in the next section of the workflow (see below). For both COI and 18S, a final read tracking table was generated showing the amount of reads (total and as percentage compared to initial input) "surviving" each step of the *cutadapt-dada2*-pipeline. The following scripts were executed in *R*:

Command

COI chimera & singleton removal with dada2 in R

```
library(dada2)

setwd("~/COI")

# Load the sequence table of the different sequence runs

Run1<-readRDS("seqtab_Run1.rds")
Run2<-readRDS("seqtab_Run2.rds")
Run3<-readRDS("seqtab_Run3.rds")
Run4<-readRDS("seqtab_Run4.rds")
Run5<-readRDS("seqtab_Run5.rds")
Run6<-readRDS("seqtab_Run6.rds")
Run7<-readRDS("seqtab_Run7.rds")

# Merge sequence tables

merged<-mergeSequenceTables(Run1,Run2,Run3,Run4,Run5,Run6,Run7)

saveRDS(merged,"merged_seqtab_coi.rds")

# Keep sequence reads with a length of 310, 313 or 316 bp only.

seqtab.filtered <- merged[,nchar(colnames(merged)) %in%
c(310,313,316)]

saveRDS(seqtab.filtered,"seqtab_filtered_coi.rds")

# Remove chimeras #

seqtab.nochim <- removeBimeraDenovo(seqtab.filtered, multithread=T,
verbose=TRUE)

# Save sequence table with the non-chimeric sequences as RDS file:

saveRDS(seqtab.nochim, "seqtab_nochim_coi.rds")

# It is possible that a large fraction of the total number of UNIQUE
SEQUENCES will be chimeras.
# However, this is usually not the case for the majority of the READS.
# Calculate percentage of the reads that were non-chimeric.
```

```
sum(seqtab.nochim)/sum(merged)

# Remove singletons from the non-chimeric ASVs

#Transform counts to numeric (as they will most likely be integers)
mode(seqtab.nochim) = "numeric"

# Subset columns with counts of > 1 and save to file
seqtab.nochim.nosingle<-seqtab.nochim[,colSums(seqtab.nochim) > 1]
saveRDS(seqtab.nochim.nosingle,"seqtab_nochim_nosingle_coi.rds")

# Write a fasta file of the final, non-chimeric , non-singleton
sequences with short >ASV... type headers

asv_seqs <- colnames(seqtab.nochim.nosingle)
asv_headers <- vector(dim(seqtab.nochim.nosingle)[2],
mode="character")
for (i in 1:dim(seqtab.nochim.nosingle)[2]) {
  asv_headers[i] <- paste(">ASV", i, sep="")
}

asv_fasta <- c(rbind(asv_headers, asv_seqs))
write(asv_fasta, "COI_nochim_nosingle_ASVs.fa")

# Write an ASV count table of the final, non-chimeric, non-singleton
sequences with short >ASV... type names

colnames(seqtab.nochim.nosingle) <- paste0("ASV",
seq(ncol(seqtab.nochim.nosingle)))

ASV_counts<-t(seqtab.nochim.nosingle) # transposing table

write.table(ASV_counts,file="COI_ASV_counts_nosingle.txt",sep="\t",
quote=F,col.names=NA)

## Track reads through the entire dada2 pipeline ##

# Track reads through the chimera and singleton removal step.

# Read non-chimeric and non-singleton table again, as it has been
modified
seqtab.nochim.nosingle<-readRDS("seqtab_nochim_nosingle_coi.rds")

track_nochim_nosingle<-
cbind(rowSums(seqtab.filtered),rowSums(seqtab.nochim),rowSums(seqtab.n
```

```
ochim.nosingle))
colnames(track_nochim_nosingle) <-
c("length_filt","nonchim","nosingle")

# Read tracking tables of the single runs and combine

track1<-read.table("track_Run1.txt",sep="\t",header=T,row.names = 1)
track2<-read.table("track_Run2.txt",sep="\t",header=T,row.names = 1)
track3<-read.table("track_Run3.txt",sep="\t",header=T,row.names = 1)
track4<-read.table("track_Run4.txt",sep="\t",header=T,row.names = 1)
track5<-read.table("track_Run5.txt",sep="\t",header=T,row.names = 1)
track6<-read.table("track_Run6.txt",sep="\t",header=T,row.names = 1)
track7<-read.table("track_Run7.txt",sep="\t",header=T,row.names = 1)

tracks<-rbind(track1,track2,track3,track4,track5,track6,track7)

# Combine all tracking tables

tracks<-
tracks[order(match(rownames(tracks),rownames(track_nochim_nosingle))),
]
track_coi<-cbind(tracks,track_nochim_nosingle)

# Calculate percentages for each step compared to input

track_coi$cutadapt_perc<-(track_coi$cutadapt / track_coi$input)*100
track_coi$filtered_perc<-(track_coi$filtered / track_coi$input)*100
track_coi$denoisedF_perc<-(track_coi$denoisedF / track_coi$input)*100
track_coi$denoisedR_perc<-(track_coi$denoisedR / track_coi$input)*100
track_coi$merged_perc<-(track_coi$merged / track_coi$input)*100
track_coi$length_filt_perc<-(track_coi$length / track_coi$input)*100
track_coi$nonchim_perc<-(track_coi$nonchim / track_coi$input)*100
track_coi$nosingle_perc<-(track_coi$nosingle / track_coi$input)*100

# Save final read tracking table to file

write.table(track_coi,"track_COI.txt",sep="\t",col.names = NA)
```

Command

18S chimera & singleton removal and taxonomic classification with dada2 in R

```
library(dada2)

setwd("~/18S")

# Load the sequence table of the different sequence runs

Run1<-readRDS("seqtab_Run1.rds")
Run2<-readRDS("seqtab_Run2.rds")
Run3<-readRDS("seqtab_Run3.rds")
Run4<-readRDS("seqtab_Run4.rds")
Run5<-readRDS("seqtab_Run5.rds")
Run6<-readRDS("seqtab_Run6.rds")
Run7<-readRDS("seqtab_Run7.rds")

# Merge sequence tables

merged<-mergeSequenceTables(Run1,Run2,Run3,Run4,Run5,Run6,Run7)

saveRDS(merged,"merged_seqtab_18S.rds")

# Do not apply length filtering due to length variability

# Remove chimeras #

seqtab.nochim <- removeBimeraDenovo(merged, multithread=T,
verbose=TRUE)

# Save sequence table with the non-chimeric sequences as RDS file:

saveRDS(seqtab.nochim, "seqtab_nochim_18S.rds")

# It is possible that a large fraction of the total number of UNIQUE
SEQUENCES will be chimeras.
# However, this is usually not the case for the majority of the READS.
# Calculate percentage of the reads that were non-chimeric.

sum(seqtab.nochim)/sum(merged)

# Remove singletons from the non-chimeric ASVs
```

```
# Remove singletons from the non-chimeric ASVs

#Transform counts to numeric (as they will most likely be integers)

mode(seqtab.nochim) = "numeric"

# Subset columns with counts of > 1

seqtab.nochim.nosingle<-seqtab.nochim[,colSums(seqtab.nochim) > 1]

saveRDS(seqtab.nochim.nosingle,"seqtab_nochim_nosingle_18S.rds")

## Track reads through the entire dada2 pipeline ##

# Track reads through the chimera and singleton removal step.

track_nochim_nosingle<-
cbind(rowSums(seqtab.nochim),rowSums(seqtab.nochim.nosingle))
colnames(track_nochim_nosingle) <- c("nonchim","nosingle")

# Read tracking tables of the single runs and combine

track1<-read.table("track_Run1.txt",sep="\t",header=T,row.names = 1)
track2<-read.table("track_Run2.txt",sep="\t",header=T,row.names = 1)
track3<-read.table("track_Run3.txt",sep="\t",header=T,row.names = 1)
track4<-read.table("track_Run4.txt",sep="\t",header=T,row.names = 1)
track5<-read.table("track_Run5.txt",sep="\t",header=T,row.names = 1)
track6<-read.table("track_Run6.txt",sep="\t",header=T,row.names = 1)
track7<-read.table("track_Run7.txt",sep="\t",header=T,row.names = 1)

tracks<-rbind(track1,track2,track3,track4,track5,track6,track7)

# Combine all tracking tables

tracks<-
tracks[order(match(rownames(tracks),rownames(track_nochim_nosingle))),
]
track_18S<-cbind(tracks,track_nochim_nosingle)

# Calculate percentages for each step compared to input

track_18S$cutadapt_perc<-(track_18S$cutadapt / track_18S$input)*100
track_18S$filtered_perc<-(track_18S$filtered / track_18S$input)*100
track_18S$denoisedF_perc<-(track_18S$denoisedF / track_18S$input)*100
track_18S$denoisedR_perc<-(track_18S$denoisedR / track_18S$input)*100
track_18S$merged_perc<-(track_18S$merged / track_18S$input)*100
track_18S$nonchim_perc<-(track_18S$nonchim / track_18S$input)*100
```

```
track_18S$nosingle_perc<-(track_18S$nosingle / track_18S$input)*100

# Save final read tracking table to file
```

Generate fasta file, count table and ensemble taxonomy for 18S ASVs

3 Taxonomy has been assigned to 18S ASVs in the previous step within the *dada2* pipeline in *R*. For this, contributed *dada2*-formatted reference files of *PR2* v5.0.0, *Silva* v132, and a specific *Silva* v132 Eukaryote set were used.

The *R* script below generates an ASV fasta file, an ASV count table and an ensemble taxonomy table based on the previous assignments of the three reference sets. To merge the different taxonomy assignments, we applied certain functions of the *ensembleTax* package and parts of its recommended pipeline (see https://github.com/dcat4/ensembleTax for more details).

Please note that resolving eukaryote taxonomy homogeneously according to clear rank hierarchy can be challenging. In addition, different reference sets use different taxonomic ranks and/or synonomous terms at a given rank for the same taxa. The ensemble taxonomy for 18S ASVs resulting from the script below contains the following ranks: Domain, Supergroup, Division, Subdivision, Phylum, Class_X, Class_Order_Family, Order_Family_X, Genus, Species. Especially rank assignments between Subdivision and Order_Family_X should be treated with caution. Depending on the reference set or taxa, these can be actual subdivisions, phyla, classes, order or families, or any of their super, sub, or infra groups. The names of these ranks define that the majority of assignments correspond to this rank. So for example, most assignments in Class_Order_Family represent one of these three ranks. However, in some cases, this does not hold true. For example, the assignment of Class_X may actually be a sub phylum or an order level assignment, etc.

Command

Ensemble taxonomy for 18S ASVs in R

```
library(devtools)
devtools::install_github("dcat4/ensembleTax", build_manual = FALSE,
build_vignettes = TRUE) # Installation issue when build_manual was
set to TRUE (problem with pdflatex)
library(ensembleTax)
library(tidyr)
library(dplyr)
library(Biostrings)
library(dada2)
library(stringr)

setwd("~/18S")

# Read non-chimeric, non-singleton sequence table

seqtab.nochim.nosingle<-readRDS("seqtab_nochim_nosingle_18S.rds")

# Read taxonomy objects (dada2 output)

silva<-readRDS("taxa_18S_silva.rds")
silva.euk<-readRDS("taxa_18S_silva_euk.rds")
pr2<-readRDS("taxa_18S_pr2.rds")

## Combine the two Silva taxonomies ##

# remove the Strain column of silva.euk

silva.euk$tax<-silva.euk$tax[,-8]
silva.euk$boot<-silva.euk$boot[,-8]

# Make dataframes of the tax objects of the two Silva taxonomies

silva_tax<-as.data.frame(silva$tax)
silva_euk_tax<-as.data.frame(silva.euk$tax)

# Create two vectors for Class and Order_Family level
# For ASVs where Silva classification was not NA, this information
was kept.
# If Silva classification was NA, the classification of the Silva
eukaryote reference set was kept.
```

```
# If both were NA, NA was set.

class <-
ifelse(!is.na(silva_tax$Class),silva_tax$Class,silva_euk_tax$Class)

order_family <-
ifelse(!is.na(silva_tax$Order),silva_tax$Order,silva_euk_tax$Order_Fam
ily)

# Combine columns of Silva and Silva eukaryote taxonomies

silva_taxonomy<-
cbind(silva_euk_tax[,1:4],silva_tax[,1:2],class,order_family,silva_euk
_tax[,7])
colnames(silva_taxonomy)[7:9]<-c("Class","Order_Family","Species") #
Set some column names

# One ASV was classified as Bacteria in the Silva classification.
Taxonomy will be set to NA for all ranks

silva_taxonomy[which(silva_taxonomy$Kingdom=="Bacteria"),] <- NA

# remove Kingdom column

silva_taxonomy<-silva_taxonomy[,-5]

# Split Species column strings into separate columns
# First. make rownames as a column, as separate_wider_functions
remove rownames (probably bug)

silva_taxonomy$seqs<-rownames(silva_taxonomy)
silva_taxonomy<-silva_taxonomy[,c(9,1:8)] # quick re-arranging of
columns
silva_taxonomy<-silva_taxonomy %>% separate_wider_delim(Species,
delim = "_", names_sep="",too_few = "align_start")

# Keep only first two columns of Species strings

silva_taxonomy<-silva_taxonomy[,-(11:ncol(silva_taxonomy))]

# Set second column of species strings to NA if it says sp. or cf. or
environmental

silva_taxonomy[which(silva_taxonomy[,10]=="sp." |
silva_taxonomy[,10]=="cf." |
silva_taxonomy[,10]=="environmental"),10] <- NA
```

```
# Set species strings to NA if the first species string column
```

Linearization of fasta files and problems with files in DOS CRLF format

4 For all downstream steps, please make sure all fasta files used as input are linearized, i.e., sequences are stored in a single line below a header, not in multiple lines.

If this is not the case for any of the fasta files, run the following in command line:

Command

Remove line breaks

```
awk '/^>/ { print (NR==1 ? "" : RS) $0; next } { printf "%s", $0 }
END { printf RS }' input.fasta > tmp && mv tmp output.fasta
```

In case you are working with *Windows* OS and the result of any of the downstream command line operations is an empty fasta file, one of the input files may be formatted in *Windows* DOS CRLF format. Check if this is the case by opening the file using Editor (or any other text editing tools), then check at the bottom. You may convert the respective file to Unix LF format in command line as shown below:

Command

Change file format from DOS CRLF to UNIX

```
dos2unix xy.file
```

Removing nuclear mitochondrial DNA pseudogenes (nuMTs) for COI using *MACSE*

5 Download latest release of *MACSE* from here: https://bioweb.supagro.inra.fr/macse/index.php?menu=releases. We used *MACSE* v2.05.

To identify putative numts, we aligned our COI non-singleton ASVs generated with *dada2* against a pre-aligned set of COI sequences. The developers of *MACSE* have generated alignments of COI sequences derived from the *BOLD* database for particular taxonomic groups. To create overall reference alignments for marine taxa with amino acid translations corresponding to the *NCBI* genetic codes (see https://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi), we downloaded these alignment fasta files for the reference sequence alignments (ref.Align fasta files) of most groups known to include marine taxa from here https://bioweb.supagro.inra.fr/macse/index.php?menu=download_Barcoding (save target as...). The refAlign.fasta files are curated alignments of reference sequences which represent the sequence diversity of the respective taxonomic group. The refAlign.fasta files for the following taxonomic groups were downloaded (refSeq.fasta for groups lackign a refAlign.fasta):

	Taxonomic groups	NCBI genetic code
	Actinopterygii, Chondrichthyes, Cyclostomata, Sarcopterygii	2
	Cnidaria, Porifera, Ctenophora	4
	Mollusca, Malacostraca, Bryozoa, Annelida, Sipuncula, Nemertea, Chaetognatha, Rotifera, Acanthocephala, Kinorhyncha, Nematoda, Tardigrada, Pycnogonida, Ostracoda, Brachiopoda (refSeq.fasta instead of refAlign.fasta), Gastrotricha (refSeq.fasta instead of refAlign.fasta)	5
	Echinodermata, Platyhelminthes, Hemichordata (refSeq.fasta instead of refAlign.fasta)	9
	Tunicata	13

Taxonomic groups for which reference sequence alignments were downlaoded. These alignment fasta files were then merged based on the genetic codes for amino acid translation.

Subsequently, the alignments of the various taxonomic groups were merged based on their respective genetic code. We used *MACSE*'s *alignTwoProfiles* function for this. After executing the *macse_vX.XX.jar* file downloaded as detailed above (*Java* needs to be installed on your machine to run *MACSE*), a GUI opens up. Navigate to *Programs* and choose a*lignTwoProfiles*. Under *ALL OPTIONS*, set the respective genetic code for *gc_def* (not necessary for Tunicata, genetic code 13: only one taxonomic group with one refAlign.fasta):

	gc_def setting	NCBI genetic code
	The_Vertebrate_Mitochondrial_Code	2
	The_Mold_Protozoan_and_Coelenterate_Mitochondrial_Code_and_the_Mycoplasma_Sprioplasma_Code	4
	The_Invertebrate_Mitochondrial_Code	5

	gc_def setting	NCBI genetic code
	The_Echinoderm_and_Flatworm_Mitochondrial_Code	9

Genetic code settings for gc_def when running *alignTwoProfiles* with *MACSE* (not necessary for Tunicata, genetic code 13: only one taxonomic group with one refAlig.fasta).

Then choose two of the refAlign.fasta files for *p1* and *p2*. We didn't change the other default settings. Then, execute *run alignTwoProfiles*. As an example, for *NCBI* genetic code 2 (vertebrates), we ran the procedure with the files *Actinopterygii_BOLD_COI_final_align_NT.aln* and *Chondrichthyes_BOLD_COI_final_align_NT.aln* as *p1* and *p2*. This resulted in two alignment files, one on nucleotide and one on amino acid level:

Actinopterygii_BOLD_COI_final_align_Chondrichthyes_BOLD_COI_final_align_AA.fasta
Actinopterygii_BOLD_COI_final_align_Chondrichthyes_BOLD_COI_final_align_NT.fasta

Subsequently, the next refAlign.fasta was added to the previously generated _NT.fasta file by running *alignTwoProfiles* again. This time, the _NT.fasta was used as *p1* and e.g. the *Sarcopterygii_BOLD_COI_final_align_NT.aln* file as *p2*. The next refAlign.fasta file was added again to the resulting _NT.fasta with the same procedure, and so on. This was continued until an overall alignment for the respective taxonomic group belonging to a particular genetic code was achieved (not necessary for Tunicata, genetic code 13: only one taxonomic group with one refAlig.fasta). Resulting alignment files were named *MACSE_BOLD_gcXX_marine_taxa_aligned_NT.fasta* and can be found below:

MACSE_BOLD_gc2_marine_taxa_ali… 349KB
MACSE_BOLD_gc4_marine_taxa_ali… 244KB
MACSE_BOLD_gc5_marine_taxa_ali… 1.2MB
MACSE_BOLD_gc9_marine_taxa_ali… 327KB
MACSE_BOLD_Tunicata_gc13_align… 70KB

To align our ASVs against these reference alignments using *MACSE*'s *enrichAlignment* function, we ran *MACSE* from within *R*. The script below was run on an HPC cluster. A directory called *pseudo* containing the directories *gc2, gc4, gc5, gc9* and *gc13* was created prior to running the *R* script. In this script, the input fasta containing the ASVs was first split into smaller fractions to enable alignment computation. The process was first completed for genetic code 5, as most of the ASVs most likely represented invertebrate taxa.

Command

MACSE alignment and pseudogene (numt) identification in R

```
library(Biostrings)
# install.packages("remotes")
# remotes::install_github("malnirav/hiReadsProcessor")
library(hiReadsProcessor)
library(dplyr)
library(seqinr)

### Run MACSE for genetic code 5 first

path <- "~/pseudo/gc5"
path.cut <- file.path(path, "splitseqs")
if(!dir.exists(path.cut)) dir.create(path.cut)
x <- readDNAStringSet("~/pseudo/COI_nochim_nosingle_ASVs.fa")

# Split the fasta file into 20 fractions.
# Make sure the input fasta is found in the respective directory

splitSeqsToFiles(x, 20, "fasta","splitseqs","~/pseudo/gc5/splitseqs")

# Make list of the file names of these split sequences

path <- "~/pseudo/gc5/splitseqs"
split.files <- sort(list.files(path, pattern = ".fasta", full.names =
TRUE))

# Create some output file names to use for MACSE

get.sample.name <- function(fname) strsplit(basename(fname), ".fasta")
[[1]][1]
sample.names <- unname(sapply(split.files, get.sample.name))
sample.names
outputAA <- file.path(paste0(path, sample.names, "_AA_gc5.fa"))
outputNT <- file.path(paste0(path, sample.names, "_NT_gc5.fa"))
outputstats <- file.path(paste0(path, sample.names, "_stats_gc5.csv"))

# Run MACSE with enrichAlignment for genetic code 5
# Make sure the .jar file of MACSE and the respective reference
alignment -NT.fasta are found in the repsective directories specified
in the command
# Do not allow any in-frame stop codons, frameshifts or insertions
```

```
and a maximum of 3 in-frame deletions (on amino acid level) in the
enrichAlignment procedure.

for(i in seq_along(split.files)) {
  system2("java", args = c(" -jar ~/pseudo/macse_v2.05.jar -prog
enrichAlignment -align
~/pseudo/MACSE_BOLD_gc5_marine_taxa_aligned_NT.fasta -seq ",
split.files[i], "-gc_def 5 -maxSTOP_inSeq 0 -output_only_added_seq_ON
=TRUE -fixed_alignment_ON =TRUE -maxDEL_inSeq 3 -maxFS_inSeq 0 -
maxINS_inSeq 0  -out_AA ", outputAA[i], " -out_NT ", outputNT[i], " -
out_tested_seq_info ", outputstats[i]))
}

# Make a table for pseudo and nonpseudo sequences.
# This is possible as running enrichAlignment will produce _stats.csv
files for each alignment. These files show if an ASV has been added
to the reference alignment under the specified settings regime.

path <- "~/pseudo/gc5"
split.files.res <- sort(list.files(path, pattern = "_stats_gc5.csv",
full.names = TRUE))
nonpseudo_gc5 = data.frame()
for(i in seq_along(split.files.res)){
  splitseqres<-read.table(split.files.res[i], h=T, sep=";")
  splitseqres1 <- splitseqres %>%
    filter(added == "yes")
  df <- data.frame(splitseqres1)
  nonpseudo_gc5 <- rbind(nonpseudo_gc5,df)
}
pseudo_gc5 = data.frame()
for(i in seq_along(split.files.res)){
  splitseqres<-read.table(split.files.res[i], h=T, sep=";")
  splitseqres1 <- splitseqres %>%
    filter(added == "no")
  df <- data.frame(splitseqres1)
  pseudo_gc5 <- rbind(pseudo_gc5,df)
}

# Subset those that were pseudogenes into a new fasta file

fastafile <- read.fasta("~/pseudo/COI_nochim_nosingle_ASVs.fa",
seqtype="DNA", as.string=TRUE)
fastafile1 <- fastafile[c(which(names(fastafile) %in%
pseudo_gc5$name))]
write.fasta(fastafile1, names=names(fastafile1), file.out =
"~/pseudo/gc5/gc5_pseudo.fasta")
```

```
### Run MACSE for genetic code 4

# Read the previously created fasta file with the putative
pseudogenes for genetic code 5 back into R and align these sequences
with genetic code 4 to see if some of these may not be identified as
pseudogenes with this translation table.

path <- "~/pseudo/gc4"
path.cut <- file.path(path, "splitseqs")
if(!dir.exists(path.cut)) dir.create(path.cut)
x <- readDNAStringSet("~/pseudo/gc5/gc5_pseudo.fasta")

# Split this file into smaller fractions

splitSeqsToFiles(x, 5, "split.fasta","splitseqs",
"~/pseudo/gc4/splitseqs")

path <- "~/pseudo/gc4/splitseqs"
split.files <- sort(list.files(path, pattern = ".fasta", full.names =
TRUE))

# Create some output file names to use for MACSE

get.sample.name <- function(fname) strsplit(basename(fname), ".fasta")
[[1]][1]
sample.names <- unname(sapply(split.files, get.sample.name))
sample.names
outputAA <- file.path(paste0(path, sample.names, "_AA_gc4.fa"))
outputNT <- file.path(paste0(path, sample.names, "_NT_gc4.fa"))
outputstats <- file.path(paste0(path, sample.names, "_stats_gc4.csv"))

# Run MACSE with enrichAlignment for genetic code 4
# Make sure the .jar file of MACSE and the respective reference
alignment -NT.fasta are found in the repsective directories specified
in the command
# Do not allow any in-frame stop codons, frameshifts or insertions
and a maximum of 3 in-frame deletions (on amino acid level) in the
enrichAlignment procedure.

for(i in seq_along(split.files)) {
  system2("java", args = c(" -jar ~/pseudo/macse_v2.05.jar -prog
enrichAlignment -align
~//pseudo/MACSE_BOLD_gc4_marine_taxa_aligned_NT.fasta -seq ",
split.files[i], "-gc_def 4 -maxSTOP_inSeq 0 -output_only_added_seq_ON
=TRUE -fixed_alignment_ON =TRUE -maxDEL_inSeq 3 -maxFS_inSeq 0 -
maxINS_inSeq 0 -out_AA ", outputAA[i], " -out_NT ", outputNT[i], " -
```

```
maxINS_inSeq 0 -out_AA ", outputAA[i], " -out_NT ", outputNT[i], " -
out_tested_seq_info ", outputstats[i]))
}

# Make a table for pseudo and nonpseudo sequences.
# This is possible as running enrichAlignment will produce _stats.csv
files for each alignment. These files show if an ASV has been added
to the reference alignment under the specified settings regime.

path <- "~/pseudo/gc4"
split.files.res <- sort(list.files(path, pattern = "_stats_gc4.csv",
full.names = TRUE))
nonpseudo_gc4 = data.frame()
for(i in seq_along(split.files.res)){
  splitseqres<-read.table(split.files.res[i], h=T, sep=";")
  splitseqres1 <- splitseqres %>%
    filter(added == "yes")
  df <- data.frame(splitseqres1)
  nonpseudo_gc4 <- rbind(nonpseudo_gc4,df)
}
pseudo_gc4 = data.frame()
for(i in seq_along(split.files.res)){
  splitseqres<-read.table(split.files.res[i], h=T, sep=";")
  splitseqres1 <- splitseqres %>%
    filter(added == "no")
  df <- data.frame(splitseqres1)
  pseudo_gc4 <- rbind(pseudo_gc4,df)
}

# Subset those that were pseudogenes into a new fasta file

fastafile <- read.fasta("~/pseudo/COI_nochim_nosingle_ASVs.fa",
seqtype="DNA", as.string=TRUE)
fastafile1 <- fastafile[c(which(names(fastafile) %in%
pseudo_gc4$name))]
write.fasta(fastafile1, names=names(fastafile1), file.out =
"~/pseudo/gc4/gc4_pseudo.fasta")

### Run MACSE for genetic code 2

# Read the previously created fasta file with the putative
pseudogenes for genetic code 4 back into R and align these sequences
with genetic code 2 to see if some of these may not be identified as
pseudogenes with this translation table.

path <- "~/pseudo/gc2"
path.cut <- file.path(path, "splitseqs")
```

```
if(!dir.exists(path.cut)) dir.create(path.cut)
x <- readDNAStringSet("~/pseudo/gc4/gc4_pseudo.fasta")

# Split it into smaller fractions

splitSeqsToFiles(x, 2, "split.fasta","splitseqs",
"~/pseudo/gc2/splitseqs")

path <- "~/pseudo/gc2/splitseqs"
split.files <- sort(list.files(path, pattern = ".fasta", full.names =
TRUE))

# Create some output file names to use for MACSE

get.sample.name <- function(fname) strsplit(basename(fname), ".fasta")
[[1]][1]
sample.names <- unname(sapply(split.files, get.sample.name))
sample.names
outputAA <- file.path(paste0(path, sample.names, "_AA_gc2.fa"))
outputNT <- file.path(paste0(path, sample.names, "_NT_gc2.fa"))
outputstats <- file.path(paste0(path, sample.names, "_stats_gc2.csv"))

# Run MACSE with enrichAlignment for genetic code 2
# Make sure the .jar file of MACSE and the respective reference
alignment -NT.fasta are found in the repsective directories specified
in the command
# Do not allow any in-frame stop codons, frameshifts or insertions
and a maximum of 3 in-frame deletions (on amino acid level) in the
enrichAlignment procedure.


for(i in seq_along(split.files)) {
  system2("java", args = c(" -jar ~/pseudo/macse_v2.05.jar -prog
enrichAlignment -align
~//pseudo/MACSE_BOLD_gc2_marine_taxa_aligned_NT.fasta -seq ",
split.files[i], "-gc_def 2 -maxSTOP_inSeq 0 -output_only_added_seq_ON
=TRUE -fixed_alignment_ON =TRUE -maxDEL_inSeq 3 -maxFS_inSeq 0 -
maxINS_inSeq 0  -out_AA ", outputAA[i], " -out_NT ", outputNT[i], " -
out_tested_seq_info ", outputstats[i]))
}

# Make a table for pseudo and nonpseudo sequences.
# This is possible as running enrichAlignment will produce _stats.csv
files for each alignment. These files show if an ASV has been added
to the reference alignment under the specified settings regime.
```

```
path <- "~/pseudo/gc2"
split.files.res <- sort(list.files(path, pattern = "_stats_gc2.csv",
full.names = TRUE))
nonpseudo_gc2 = data.frame()
for(i in seq_along(split.files.res)){
  splitseqres<-read.table(split.files.res[i], h=T, sep=";")
  splitseqres1 <- splitseqres %>%
    filter(added == "yes")
  df <- data.frame(splitseqres1)
  nonpseudo_gc2 <- rbind(nonpseudo_gc2,df)
}
pseudo_gc2 = data.frame()
for(i in seq_along(split.files.res)){
  splitseqres<-read.table(split.files.res[i], h=T, sep=";")
  splitseqres1 <- splitseqres %>%
    filter(added == "no")
  df <- data.frame(splitseqres1)
  pseudo_gc2 <- rbind(pseudo_gc2,df)
}

# Subset those that were pseudogenes into a new fasta file

fastafile <- read.fasta("~/pseudo/COI_nochim_nosingle_ASVs.fa",
seqtype="DNA", as.string=TRUE)
fastafile1 <- fastafile[c(which(names(fastafile) %in%
pseudo_gc2$name))]
write.fasta(fastafile1, names=names(fastafile1), file.out =
"~/pseudo/gc2/gc2_pseudo.fasta")

### Run MACSE for genetic code 9

# Read the previously created fasta file with the putative
pseudogenes for genetic code 2 back into R and align these sequences
with genetic code 9 to see if some of these may not be identified as
pseudogenes with this translation table.

# Make list of the file names of these split sequences

path <- "~/pseudo/gc9"
path.cut <- file.path(path, "splitseqs")
if(!dir.exists(path.cut)) dir.create(path.cut)
x <- readDNAStringSet("~/pseudo/gc2/gc2_pseudo.fasta")

# Split it into smaller fractions

splitSeqsToFiles(x, 2, "split.fasta","splitseqs",
"~/pseudo/gc9/splitseqs")
```

```
~/pseudo/gc9/splitseqs")

path <- "~/pseudo/gc9/splitseqs"
split.files <- sort(list.files(path, pattern = ".fasta", full.names =
TRUE))

# Create some output file names to use for MACSE

get.sample.name <- function(fname) strsplit(basename(fname), ".fasta")
[[1]][1]
sample.names <- unname(sapply(split.files, get.sample.name))
sample.names
outputAA <- file.path(paste0(path, sample.names, "_AA_gc9.fa"))
outputNT <- file.path(paste0(path, sample.names, "_NT_gc9.fa"))
outputstats <- file.path(paste0(path, sample.names, "_stats_gc9.csv"))

# Run MACSE with enrichAlignment for genetic code 9
# Make sure the .jar file of MACSE and the respective reference
alignment -NT.fasta are found in the repsective directories specified
in the command
# Do not allow any in-frame stop codons, frameshifts or insertions
and a maximum of 3 in-frame deletions (on amino acid level) in the
enrichAlignment procedure.

for(i in seq_along(split.files)) {
  system2("java", args = c(" -jar ~/pseudo/macse_v2.05.jar -prog
enrichAlignment -align
~//pseudo/MACSE_BOLD_gc9_marine_taxa_aligned_NT.fasta -seq ",
split.files[i], "-gc_def 9 -maxSTOP_inSeq 0 -output_only_added_seq_ON
=TRUE -fixed_alignment_ON =TRUE -maxDEL_inSeq 3 -maxFS_inSeq 0 -
maxINS_inSeq 0  -out_AA ", outputAA[i], " -out_NT ", outputNT[i], " -
out_tested_seq_info ", outputstats[i]))
}

# Make a table for pseudo and nonpseudo sequences.
# This is possible as running enrichAlignment will produce _stats.csv
files for each alignment. These files show if an ASV has been added
to the reference alignment under the specified settings regime.

path <- "~/pseudo/gc9"
split.files.res <- sort(list.files(path, pattern = "_stats_gc9.csv",
full.names = TRUE))
nonpseudo_gc9 = data.frame()
for(i in seq_along(split.files.res)){
  splitseqres<-read.table(split.files.res[i], h=T, sep=";")
  splitseqres1 <- splitseqres %>%
    filter(added == "yes")
```

```
  df <- data.frame(splitseqres1)
  nonpseudo_gc9 <- rbind(nonpseudo_gc9,df)
}
pseudo_gc9 = data.frame()
for(i in seq_along(split.files.res)){
  splitseqres<-read.table(split.files.res[i], h=T, sep=";")
  splitseqres1 <- splitseqres %>%
    filter(added == "no")
  df <- data.frame(splitseqres1)
  pseudo_gc9 <- rbind(pseudo_gc9,df)
}

# Subset those that were pseudogenes into a new fasta file

fastafile <- read.fasta("~/pseudo/COI_nochim_nosingle_ASVs.fa",
seqtype="DNA", as.string=TRUE)
fastafile1 <- fastafile[c(which(names(fastafile) %in%
pseudo_gc9$name))]
write.fasta(fastafile1, names=names(fastafile1), file.out =
"~/pseudo/gc9/gc9_pseudo.fasta")

### Run MACSE for genetic code 13

# Read the previously created fasta file with the putative
pseudogenes for genetic code 9 back into R and align these sequences
with genetic code 13 to see if some of these may not be identified as
pseudogenes with this translation table.

path <- "~/pseudo/gc13"
path.cut <- file.path(path, "splitseqs")
if(!dir.exists(path.cut)) dir.create(path.cut)
x <- readDNAStringSet("~/pseudo/gc9/gc9_pseudo.fasta")

# Split it into smaller fractions

splitSeqsToFiles(x, 2, "split.fasta","splitseqs",
"~/pseudo/gc13/splitseqs")

path <- "~/pseudo/gc13/splitseqs"
split.files <- sort(list.files(path, pattern = ".fasta", full.names =
TRUE))

# Create some output file names to use for MACSE

get.sample.name <- function(fname) strsplit(basename(fname), ".fasta")
[[1]][1]
```

```
sample.names <- unname(sapply(split.files, get.sample.name))
sample.names
outputAA <- file.path(paste0(path, sample.names, "_AA_gc13.fa"))
outputNT <- file.path(paste0(path, sample.names, "_NT_gc13.fa"))
outputstats <- file.path(paste0(path, sample.names,
"_stats_gc13.csv"))

# Run MACSE with enrichAlignment for genetic code 13
# Make sure the .jar file of MACSE and the respective reference
alignment -NT.fasta are found in the repsective directories specified
in the command
# Do not allow any in-frame stop codons, frameshifts or insertions
and a maximum of 3 in-frame deletions (on amino acid level) in the
enrichAlignment procedure.

for(i in seq_along(split.files)) {
  system2("java", args = c(" -jar ~/pseudo/macse_v2.05.jar -prog
enrichAlignment -align
~//pseudo/MACSE_BOLD_Tunicata_gc13_aligned_NT.fasta -seq ",
split.files[i], "-gc_def 13 -maxSTOP_inSeq 0 -
output_only_added_seq_ON =TRUE -fixed_alignment_ON =TRUE -
maxDEL_inSeq 3 -maxFS_inSeq 0 -maxINS_inSeq 0  -out_AA ",
outputAA[i], " -out_NT ", outputNT[i], " -out_tested_seq_info ",
outputstats[i]))
}

# Make a table for pseudo and nonpseudo sequences

path <- "~/pseudo/gc13"
split.files.res <- sort(list.files(path, pattern = "_stats_gc13.csv",
full.names = TRUE))
nonpseudo_gc13 = data.frame()
for(i in seq_along(split.files.res)){
  splitseqres<-read.table(split.files.res[i], h=T, sep=";")
  splitseqres1 <- splitseqres %>%
    filter(added == "yes")
  df <- data.frame(splitseqres1)
  nonpseudo_gc13 <- rbind(nonpseudo_gc13,df)
}
pseudo_gc13 = data.frame()
for(i in seq_along(split.files.res)){
  splitseqres<-read.table(split.files.res[i], h=T, sep=";")
  splitseqres1 <- splitseqres %>%
    filter(added == "no")
  df <- data.frame(splitseqres1)
  pseudo_gc13 <- rbind(pseudo_gc13,df)
}
```

```
)

# Make a table for pseudo and nonpseudo sequences.
# This is possible as running enrichAlignment will produce _stats.csv
files for each alignment. These files show if an ASV has been added
to the reference alignment under the specified settings regime.

fastafile <- read.fasta("~/pseudo/COI_nochim_nosingle_ASVs.fa",
seqtype="DNA", as.string=TRUE)
fastafile1 <- fastafile[c(which(names(fastafile) %in%
pseudo_gc13$name))]
write.fasta(fastafile1, names=names(fastafile1), file.out =
"~/pseudo/gc13/gc13_pseudo.fasta")

# Output those ASVs that are potential pseudos and the ones that are
not

pseudo.combined <- rbind(pseudo_gc5,
pseudo_gc4,pseudo_gc2,pseudo_gc9,pseudo_gc13)
pseudo.combined.names <- as.character(unique(pseudo.combined$name))
pseudo.combined.names<-paste0(">",pseudo.combined.names)
nonpseudo.combined <- rbind(nonpseudo_gc5,
nonpseudo_gc4,nonpseudo_gc2,nonpseudo_gc9,nonpseudo_gc13)
nonpseudo.combined.names <-
as.character(unique(nonpseudo.combined$name))
nonpseudo.combined.names<-paste0(">",nonpseudo.combined.names)

write.table(pseudo.combined.names,
"~/pseudo/pseudo.combined.names.txt",row.names = F,col.names =
F,quote = F)
write.table(nonpseudo.combined.names,
"~/pseudo/nonpseudo.combined.names.txt",row.names = F,col.names =
F,quote = F)
```

The final *gc13_pseudo.fasta* contains the overall putative numt-ASV set after running *MACSE* with five translations (one for each genetic code). The resulting nonpseudo.combined.names.txt file contains the non-numt ASVs (in ">ASVxy" format).

Non-numt ASVs were then subset from the *COI_nochim_nosingle.fa* file (resulting from the *dada2* workflow, see above) by running the following in command line:

Command

Subset non-numt ASVs

```
grep -w -A 1 -f nonpseudo.combined.names.txt
COI_nochim_nosingle_ASVs.fa > COI_nochim_nosingle_nopseudo.fa --no-
group-separator
```

Negative control correction

6 After nuMT filtering for COI, we removed 18S and COI ASVs for which the read count in negative control samples exceeded 10% of their total read count. This was done in *R* / *RStudio* using the following script:

Command

Blank correction in R

```
### COI ###

setwd("~/COI")

# Read ASV count table (output from dada2)

asv_table<-read.table("COI_ASV_counts_nosingle.txt",sep="\t",header =
T,row.names = 1,as.is=T,check.names=F)

# Read list of ASVs which remained after NUMT removal

asv_list<-read.table("nonpseudo.combined.names.txt",sep="\t")

# Remove ">" in asv_list

asv_list[,1]<-gsub(">","",asv_list[,1])

# Subset asv_table to non-numt ASVs

asv_table<-subset(asv_table, rownames(asv_table) %in% asv_list[,1])

# Subset rows where the ASV read count in the blank / negative
samples  exceeds 10 % of an ASVs total read count

blank1<-
subset(asv_table,asv_table[,colnames(asv_table)=="ERR3460471"] >
0.1*rowSums(asv_table))
blank2<-
subset(asv_table,asv_table[,colnames(asv_table)=="ERR4018471"] >
0.1*rowSums(asv_table))
blank3<-
subset(asv_table,asv_table[,colnames(asv_table)=="ERR4018737"] >
0.1*rowSums(asv_table))
blank4<-
subset(asv_table,asv_table[,colnames(asv_table)=="ERR4914160"] >
0.1*rowSums(asv_table))
blank5<-
subset(asv_table,asv_table[,colnames(asv_table)=="ERR7125534"] >
0.1*rowSums(asv_table))
blank6<-
```

```
subset(asv_table,asv_table[,colnames(asv_table)=="ERR7125582"] >
0.1*rowSums(asv_table))
blank7<-
subset(asv_table,asv_table[,colnames(asv_table)=="ERR9632065"] >
0.1*rowSums(asv_table))
blank8<-
subset(asv_table,asv_table[,colnames(asv_table)=="ERR12541481"] >
0.1*rowSums(asv_table))
blank9<-
subset(asv_table,asv_table[,colnames(asv_table)=="ERR12541482"] >
0.1*rowSums(asv_table))

# before combining the tables using rbind, add a column with ASV IDs.
This is necessary to stop R from introducing new rownames adding
zeros to ASV names if duplicate ASVs exist in the newly created blank
dataframes

blank1 <- cbind(ASV_ID = rownames(blank1), blank1)
blank2 <- cbind(ASV_ID = rownames(blank2), blank2)
blank3 <- cbind(ASV_ID = rownames(blank3), blank3)
blank4 <- cbind(ASV_ID = rownames(blank4), blank4)
blank5 <- cbind(ASV_ID = rownames(blank5), blank5)
blank6 <- cbind(ASV_ID = rownames(blank6), blank6)
blank7 <- cbind(ASV_ID = rownames(blank7), blank7)
blank8 <- cbind(ASV_ID = rownames(blank8), blank8)
blank9 <- cbind(ASV_ID = rownames(blank9), blank9)

# Combine blank dataframes

blanks<-
rbind(blank1,blank2,blank3,blank4,blank5,blank6,blank7,blank8,blank9)

# Remove ASVs if there are duplicates in the "blanks" table (in case
an ASV's read count exceeded 10 % of the total read count in more
than one blank sample)

blanks <- blanks[!duplicated(blanks$ASV_ID), ]

# Remove the potential contaminant ASVs from the ASV table

asv_table<- cbind(ASV_ID = rownames(asv_table), asv_table)
asv_no_contams<-asv_table[!(asv_table$ASV_ID %in% blanks$ASV_ID),]

# Write table of potential contaminant ASVs and ASV count table
devoid of contaminants
```

```
write.table(blanks,"asv_contaminants_COI.txt",sep="\t",row.names = F)

write.table(asv_no_contams,"asv_no_contaminants_COI.txt",sep="\t",row.
names = F)

# Write clean fasta file
no_contam_headers<-paste0(">",asv_no_contams$ASV_ID,
"\n",asv_no_contams$ASV)
write.table(no_contam_headers,"no_contam_headers_COI.txt",sep="\t",row
.names = F,quote=F,col.names = F)
# COI
cd ~/COI
### 18S ###
grep -w -A 1 -f no_contam_headers_COI.txt
COI_nochim_nosingle_nopseudo.fa --no-group-separator >
COI_nochim_nosingle_nopseudo_nocontam.fa

# Read ASV count table (output from dada2)
# 18S
cd ~/18S
asv_table<-read.table("18S_ASV_counts_nosingle.txt",sep="\t",header =
T,row.names = 1,as.is=T,check.names=F)
grep -w -A 1 -f no_contam_headers_18S.txt 18S_nochim_nosingle_ASVs.fa
--no-group-separator > 18S_nochim_nosingle_nocontam.fa

# Subset rows where the ASV read count in the blank / negative
samples  exceeds 10 % of an ASVs total read count


blank1<-
subset(asv_table,asv_table[,colnames(asv_table)=="ERR4018472"] >
```

We then subset the ASVs remaining after correction in command line:

Command

Generate fastas with ASVs remaining after blank correction

Clustering ASVs into Molecular Operational Taxonomic Units (MOTUs) using *swarm*

7 Install the most recent version of *swarm* on your system. See here: https://github.com/torognes/swarm. We used *swarm* v3.0.0. A note on this part: downloading the installation folder of a corresponding release will provide you with a bin folder containing a swarm.exe file (at least this was the case for the Windows version). This made a more complicated installation via compilation unnecessary. According to the developers, this file is provided for convenience and can just be executed as is.

The previously generated fasta files containing the blank-corrected ASVs needs to be dereplicated (total read abundances neet to be added to the ASV headers as _XXX) to be used for clustering with *swarm*. Run the following script in *R* / *RStudio* to generate a file with the appropriate ASV header format:

Command

Dereplication of ASV headers in R

```
# For clustering ASVs into MOTUs using swarm, an ASV fasta with
headers containing the total abundance of each is required.
# Here, we generate these new headers for the fasta files which will
be used as input for swarm.

### COI ###

setwd("~/COI")

# Read the non-contaminant COI ASV count table

ASV_counts<-
read.table(file="asv_no_contaminants_COI.txt",sep="\t",row.names=1,hea
der=T)

# Count total read abundances per ASV

ASV_sums<-rowSums(ASV_counts)

# Create headers containing read counts

seqnames<-paste0(">",paste(rownames(ASV_counts),ASV_sums,sep="_"))

write.table(seqnames,
"ASV_dereplicated.txt",sep="\t",col.names=F,row.names = F,quote=F)

### 18S ###

setwd("~/18S")

# Read the non-contaminant 18S ASV count table

ASV_counts<-
read.table(file="asv_no_contaminants_18S.txt",sep="\t",row.names=1,hea
der=T)

# Count total read abundances per ASV

ASV_sums<-rowSums(ASV_counts)
```

```
# Create headers containing read counts

seqnames<-paste0(">",paste(rownames(ASV_counts),ASV_sums,sep="_"))

write.table(seqnames,
"ASV_dereplicated.txt",sep="\t",col.names=F,row.names = F,quote=F)
```

Replace the headers in the fasta files containing the blank-corrected ASVs with the dereplicated headers. Run the following in command line:

Command

Replace headers with dereplicated header names

```
# COI
cd ~/COI
awk 'NR%2==0' COI_nochim_nosingle_nopseudo_nocontam.fa | paste -d'\n'
ASV_dereplicated.txt - > COI_dereplicated_ASVs.fa

# 18S
cd ~/18S
awk 'NR%2==0' 18S_nochim_nosingle_nocontam.fa | paste -d'\n'
ASV_dereplicated.txt - > 18S_dereplicated_ASVs.fa
```

Then run the swarm.exe file. Something like the following will pop up, *swarm* is now waiting for data input.

Command

Swarm

```
Swarm 3.0.0
Copyright (C) 2012-2019 Torbjorn Rognes and Frederic Mahe
https://github.com/torognes/swarm

Mahe F, Rognes T, Quince C, de Vargas C, Dunthorn M (2014)
Swarm: robust and fast clustering method for amplicon-based studies
PeerJ 2:e593 https://doi.org/10.7717/peerj.593

Mahe F, Rognes T, Quince C, de Vargas C, Dunthorn M (2015)
Swarm v2: highly-scalable and high-resolution amplicon clustering
PeerJ 3:e1420 https://doi.org/10.7717/peerj.1420

CPU features:      mmx sse sse2 sse3 ssse3 sse4.1 sse4.2 popcnt avx
avx2
Database file:     -
Output file:       -
Resolution (d):    1
Threads:           1
Break OTUs:        Yes
Fastidious:        No

Waiting for data... (Hit Ctrl-C and run swarm -h if you meant to read
data from a file.)
```

This window should not be closed, *swarm* is now waiting for data input. We then ran the actual *swarm* algorithm from comand line. -d sets the most important parameter, the number of differences allowed between ASVs to be grouped into the same molecular operational taxonomic unit (MOTU). -i, -o, -s and -u allow us to write certain information and statistics to files specified by the corresponding file names. -w and the following specified fasta file name determine that a fasta file with the representative sequences of each MOTU is written (headers show which ASV was determined as representative and the total number of reads of the corresponding MOTU), the ultimate file name specifies the input fasta file for the clustering procedure. For COI, d = 13 was applied, while for 18S, d was set to 1 and the fastidious option was enabled (-f, only available for d = 1):

Command

swarm clustering execution

```
# COI
cd ~/COI
swarm -d 13 -i internal.txt -o output.txt -s statistics.txt -u
uclust.txt -w COI_cluster_reps.fa COI_dereplicated_ASVs.fa

# 18S
cd ~/18S
swarm -d 1 -f -i internal.txt -o output.txt -s statistics.txt -u
uclust.txt -w 18S_cluster_reps.fa 18S_dereplicated_ASVs.fa
```

LULU curation

8 The clustered MOTUs were curated using *LULU* v0.1.0 (Frøslev *et al.*, 2017) in *R / RStudio*. For details on *LULU* curation, see https://github.com/tobiasgf/lulu.

First, we generated MOTU tables based on the ASV read count tables resulting from blank correction (see above) and the *swarm* output files. See the following *R* script:

Command

MOTU tables for LULU using R

```
### COI ###

setwd("~/COI")

# Read the uclust.txt table (from swarm output)

uclust<-read.table("uclust.txt",sep="\t",stringsAsFactors = F)

# Delete the rows with value "C" in first column (to remove one of
the S / C duplicate ASV names)

uclust2<-subset(uclust, uclust[,1]!="C")

# Rename the 10th column if first column = S to give the most
abundant ASv in a cluster a MOTU ID

uclust2[,10] <-ifelse(uclust2[,1]=="S",uclust2[,9],uclust2[,10])

# Subset the columns we need

motu_asv_list<-uclust2[,9:10]

# Remove all characters after _ (incl. _)

motu_asv_list[] <- lapply(motu_asv_list, function(y) gsub("_.*","",
y))

# Rename columns

colnames(motu_asv_list)<-c("ASV","MOTU")

# Read the ASV count table (output from blank correction)

asv_counts<-read.table("asv_no_contaminants_COI.txt",sep="\t",header
= T,stringsAsFactors=F)

# Sort motu_asv_list based on order in asv_counts

motu_asv_list<-
motu_asv_list[order(match(motu_asv_list[,1],asv_counts[,1])),]
```

```
# Bind corresponding MOTUs to the ASV count table

asv_counts<-cbind(motu_asv_list$MOTU,asv_counts,stringsAsFactors=F)
colnames(asv_counts)[1]<-"MOTU"

# Sum ASV counts based on MOTU
# First, delete the original column containing the ASV names as they
cannot be summed

asv_counts<-asv_counts[-2]
motu_table <- aggregate(. ~ MOTU, data=asv_counts, FUN=sum)

# Write MOTU table to file

write.table(motu_table,"motu_table_COI.txt",sep="\t",row.names=F,quote
=F)

### 18S ###

setwd("~/18S")

# Read the uclust.txt table (from swarm output)

uclust<-read.table("uclust.txt",sep="\t",stringsAsFactors = F)

# Delete the rows with value "C" in first column (to remove one of
the S / C duplicate ASV names)

uclust2<-subset(uclust, uclust[,1]!="C")

# Rename the 10th column if first column = S to give the most
abundant ASV in a cluster a MOTU ID

uclust2[,10] <-ifelse(uclust2[,1]=="S",uclust2[,9],uclust2[,10])

# Subset the columns we need

motu_asv_list<-uclust2[,9:10]

# Remove all characters after _ (incl. _)

motu_asv_list[] <- lapply(motu_asv_list, function(y) gsub("_.*","",
y))

# Rename columns
```

```
colnames(motu_asv_list)<-c("ASV","MOTU")

# Read the ASV count table (output from blank correction)

asv_counts<-read.table("asv_no_contaminants_18S.txt",sep="\t",header
= T,stringsAsFactors=F)

# Sort motu_asv_list based on order in asv_counts

motu_asv_list<-
motu_asv_list[order(match(motu_asv_list[,1],asv_counts[,1])),]

# Bind corresponding MOTUs to the ASV count table

asv_counts<-cbind(motu_asv_list$MOTU,asv_counts,stringsAsFactors=F)
colnames(asv_counts)[1]<-"MOTU"

# Sum ASV counts based on MOTU
# First, delete the original column containing the ASV names as they
cannot be summed

asv_counts<-asv_counts[-2]
motu_table <- aggregate(. ~ MOTU, data=asv_counts, FUN=sum)

# Write MOTU table to file

write.table(motu_table,"motu_table_18S.txt",sep="\t",row.names=F,quote
=F)
```

To obtain identical MOTU IDs in the MOTU tables and the fasta files with cluster representative sequences obtained from *swarm*, we removed the MOTU read abundance strings from the sequence headers in the latter from command line:

Command

Adjust sequence headers in the fasta files for LULU processing

```
# COI
cd ~/COI
awk -F'_' '{print $1}' COI_cluster_reps.fa >
COI_cluster_reps_lulu_ready.fa

# 18S
cd ~/18S
awk -F'_' '{print $1}' 18S_cluster_reps.fa >
18S_cluster_reps_lulu_ready.fa
```

Match lists meeting *LULU*s requirements were generated using *BLASTn.* For this*, BLAST+* was installed as standalone version for Windows by downloading the corresponding win64.exe file from ftp://ftp.ncbi.nlm.nih.gov/blast/executables/LATEST (release 2.11.0 in this case). Please note that downloading via *Mozilla* lead to files being corrupted and not executable for some reason, this was not the case when using *Chrome* as browser. For each marker gene, a blastdatabase was then produced, followed by blasting the MOTUs against this database. We executed the following in command line (note that for the perc_identity parameter, different values were applied *f*or COI and 18S):

Command

Produce match lists with BLASTn

```
# COI
cd ~/COI

makeblastdb -in COI_cluster_reps_lulu_ready.fa -parse_seqids -dbtype
nucl

blastn -db COI_cluster_reps_lulu_ready.fa -outfmt '6 qseqid sseqid
pident' -out match_list.txt -qcov_hsp_perc 80 -perc_identity 84 -
query COI_cluster_reps_lulu_ready.fa

# 18S
cd ~/18S

makeblastdb -in 18S_cluster_reps_lulu_ready.fa -parse_seqids -dbtype
nucl

blastn -db 18S_cluster_reps_lulu_ready.fa -outfmt '6 qseqid sseqid
pident' -out match_list.txt -qcov_hsp_perc 80 -perc_identity 90 -
query 18S_cluster_reps_lulu_ready.fa
```

We then ran the actual *LULU* curation in *R* / RStudio with a minimium sequence similarity threshold of 0.84 for COI and 0.90 for 18S (this value has to be equal or above the perc_identity parameter used during the match list creation) and minimum co-occurrence rate of 0.95. For 18S, we changed the minimum_ratio to 100, from the default of 1. Curated MOTU tables as well as mapping files were generated, see the following script:

Command

LULU curation in R

```
# library(devtools)
# install_github("tobiasgf/lulu")

library(lulu)

### COI ###

setwd("~/COI")

# Read MOTU table

motu_table<-read.table("motu_table_COI.txt",sep="\t",header =
T,row.names = 1,as.is=T)

# Read match list

matchlist<-read.table("match_list.txt",sep="\t",header=F,as.is =
T,stringsAsFactors = F)

# Run curation

curated_result <- lulu(motu_table, matchlist,minimum_match =
0.84,minimum_relative_cooccurence = 0.9)

# Write curated MOTU table to file

write.table(curated_result$curated_table,"lulu_motu_table_COI.txt",sep
="\t",quote=F,col.names = NA)

# Write data on how MOTUs where mapped to file

write.table(curated_result$otu_map,"motu_map_lulu_COI.txt",sep="\t",qu
ote=F,col.names = NA)

# Write headers of curated MOTUs to file to subset the corresponding
representative sequences of the swarm cluster fasta file later on

lulu_curated_headers<-
paste0(">",row.names(curated_result$curated_table))
write.table(lulu curated headers,"lulu curated headers.txt",sep="\t",r
```

```
ow.names = F,quote=F,col.names = F)

### 18S ###

setwd("~/18S")

# Read MOTU table

motu_table<-read.table("motu_table_18S.txt",sep="\t",header =
T,row.names = 1,as.is=T)

# Read match list

matchlist<-read.table("match_list.txt",sep="\t",header=F,as.is =
T,stringsAsFactors = F)

# Run curation

curated_result <- lulu(motu_table, matchlist,minimum_match =
0.9,minimum_ratio=100,minimum_relative_cooccurence = 0.95)

# Write curated MOTU table to file

write.table(curated_result$curated_table,"lulu_motu_table_18S.txt",sep
="\t",quote=F,col.names = NA)

# Write data on how MOTUs where mapped to file

write.table(curated_result$otu_map,"motu_map_lulu_18S.txt",sep="\t",qu
ote=F,col.names = NA)

# Write headers of curated MOTUs to file to subset the corresponding
representative sequences of the swarm cluster fasta file later on

lulu_curated_headers<-
paste0(">",row.names(curated_result$curated_table))
write.table(lulu_curated_headers,"lulu_curated_headers.txt",sep="\t",r
ow.names = F,quote=F,col.names = F)
```

We then subset the representative sequences of MOTUs remaining after *LULU* curation from command line:

Command

Generate fasta files with MOTUs remaining after LULU curation

```
# COI
cd ~/COI
grep -w -A 1 -f lulu_curated_headers.txt
COI_cluster_reps_lulu_ready.fa --no-group-separator >
COI_cluster_reps_lulu_curated.fa

# 18S
cd ~/18S
grep -w -A 1 -f lulu_curated_headers.txt
18S_cluster_reps_lulu_ready.fa --no-group-separator >
18S_cluster_reps_lulu_curated.fa
```

Subset taxonomy of 18S MOTUs and taxonomy assignment for COI MOTUs

9 Taxonomy has previously been assigned to 18S ASVs (see above). With the *R* script below, the respective taxonomy of the representative ASV sequences of *LULU*-curated MOTUs will be subset from the initial taxonomy table:

Command

Subset 18S taxonomy in R

```
library(dplyr)

setwd("~/18S")

# Read the ensemble taxonomy table generated after the dada2 pipeline

tax_table<-read.table("18S_tax_table.txt",sep="\t",header=T)

# Read the headers of LULU curated MOTUs

headers<-read.table("lulu_curated_headers.txt",sep="\t")
headers[,1]<-gsub(">","",headers[,1]) # Remove the ">" from the MOTU
names

# Subset the taxonomy assignments of ASVs which are the
representative ASV sequences of LULU curated MOTUs

taxa_lulu<-tax_table %>% filter(ASV %in% headers[,1])
colnames(taxa_lulu)[1]<-"MOTU"

# Write to file

write.table(taxa_lulu,"18S_final_tax_table.txt",sep="\t",row.names =
F)
```

Taxonomy was assigned to COI MOTUs using two assignment tools and reference databases. The *Barcode Of Life Data System* (i.e., *BOLD)* database was used via the P*ython* tool *BOLDigger-commandline* (see https://github.com/DominikBuchner/BOLDigger-commandline and https://github.com/DominikBuchner/BOLDigger for details). This tool requires a *BOLDSYSTEMS* user account (**https://v3.boldsystems.org/**) and *Python* v3.6 or higher. In addition, taxonomy was assigned to the COI dataset using the *MIDORI2* database via its web server tool (**https://reference-midori.info/server.php**; based on *GenBank* release 257 at the time of usage). Results from both databases were then compared and final classification determined as described below.

For *MIDORI2*, we uploaded the *LULU*-curated fasta file to the web server (http://reference-midori.info/server.php). Because the web server allows a maximum of 10,000 sequences to be classified in one job and our *LULU*-curated fasta file contained more sequences, we simply split the fasta file into two files in *R/RStudio* using the commands below:

Command

Split fasta file in R

```
library(Biostrings)
library(hiReadsProcessor)

setwd("~/COI")

# read LULU-curated fasta file

fasta<-readDNAStringSet("COI_cluster_reps_lulu_curated.fa ")

# Split fasta into two files and save them

splitSeqsToFiles(fasta, 2, "fasta","COI_cluster_reps_lulu_curated")
```

Note: It seems that if the number of sequences in the input fasta file is odd-numbered, splitting into two files will create two new equally-sized fasta files containing the same even number of sequence) and a third fasta file with the final remaining sequence. you can just manually add this final sequence to the second fasta file and remove the third file.

We then simply ran two jobs on the server using the two fasta files as input. As *Program*, we used the "RDP classifier". As *Database*, we chose "Unique" and "COI". We applied a confidence cut-off of 0.7 with the *output format* "allrank" in the *Parameters* section. While there is an *output format* "filterbyconf" which automatically provides classification for each taxonomic level when the confidence threshold is met (and otherwise sets it as "unclassified"), it does not provide species level classification for some reason. Therefore, we ran classification with aforementioned "allrank" format and subsequently filtered the taxonomic classification to remove information with a classification confidence of below 0.7. Classification via the *MIDORI2* webserver results in two files being provided via e-mail: ...hier_outfile.txt and ...usga_classified.txt. We used the ...usga_classified.txt files of the two jobs (re-named to midori2_GB_257_1.txt and midori2_GB_257_2.txt with 257 being the respective *GenBank* version which *MIDORI2* applies at the time of usage) for threshold application. In a few cases, information is missing on some taxonomic levels while it is available on lower levels, which

shifts the entries within the results table. This made it necessary to format the result files and subsequently merge both files. This was done in *R / RStudio* using the commands below:

Command

Format MIDORI2 results in R for confidence threshold application

```
library(dplyr)

setwd("~/COI")

### Format tax table from Midori2 web server to remove information
with confidence below 0.7

# Load the ...usga_classified.txt files (output of the Midori2 web
server)
# Files have been renamed to midori_GB_257_1.txt and
midori_GB_257_2.txt after downloading them

midori1<-read.table("midori2_GB_257_1.txt",sep="\t",header=F,fill=T)
midori2<-read.table("midori2_GB_257_2.txt",sep="\t",header=F,fill=T)

# remove unnecessary columns

midori1<-midori1[,-c(2:5,7,8)]
midori2<-midori2[,-c(2:5,7,8)]

# For some reason, there are rows where information on taxonomic
levels is missing even though it is present on lower taxonomic levels.
# In these cases, the entries got shifted to the left and need to be
shifted back to the correct position.

# see where species level information got shifted and correct it

# Identifying which rows to shift
rows_to_change1 <- midori1$V22 %in% "species"
rows_to_change2 <- midori2$V22 %in% "species"

# Moving columns one space to the right
midori1[rows_to_change1,18:20] <- midori1[rows_to_change1,15:17]
midori2[rows_to_change2,18:20] <- midori2[rows_to_change2,15:17]

# Deleting wrong values
midori1[rows_to_change1,15:17] <- NA
midori2[rows_to_change2,15:17] <- NA

# Identifying which rows to shift
```

```
rows_to_change1 <- midori1$V19 %in% "species"
rows_to_change2 <- midori2$V19 %in% "species"

# Moving columns one space to the right
midori1[rows_to_change1,18:20] <- midori1[rows_to_change1,12:14]
midori2[rows_to_change2,18:20] <- midori2[rows_to_change2,12:14]

# Deleting wrong values
midori1[rows_to_change1,12:14] <- NA
midori2[rows_to_change2,12:14] <- NA

# Identifying which rows to shift
rows_to_change1 <- midori1$V16 %in% "species"
rows_to_change2 <- midori2$V16 %in% "species"

# Moving columns one space to the right
midori1[rows_to_change1,18:20] <- midori1[rows_to_change1,9:11]
midori2[rows_to_change2,18:20] <- midori2[rows_to_change2,9:11]

# Deleting wrong values
midori1[rows_to_change1,9:11] <- NA
midori2[rows_to_change2,9:11] <- NA

# Identifying which rows to shift
rows_to_change1 <- midori1$V13 %in% "species"
rows_to_change2 <- midori2$V13 %in% "species"

# Moving columns one space to the right
midori1[rows_to_change1,18:20] <- midori1[rows_to_change1,6:8]
midori2[rows_to_change2,18:20] <- midori2[rows_to_change2,6:8]

# Deleting wrong values
midori1[rows_to_change1,6:8] <- NA
midori2[rows_to_change2,6:8] <- NA

# see where genus level information got shifted and correct it

# Identifying which rows to shift
rows_to_change1 <- midori1$V19 %in% "genus"
rows_to_change2 <- midori2$V19 %in% "genus"

# Moving columns one space to the right
midori1[rows_to_change1,15:17] <- midori1[rows_to_change1,12:14]
midori2[rows_to_change2,15:17] <- midori2[rows_to_change2,12:14]

# Deleting wrong values
```

```
midori1[rows_to_change1,12:14] <- NA
midori2[rows_to_change2,12:14] <- NA

# Identifying which rows to shift
rows_to_change1 <- midori1$V16 %in% "genus"
rows_to_change2 <- midori2$V16 %in% "genus"

# Moving columns one space to the right
midori1[rows_to_change1,15:17] <- midori1[rows_to_change1,9:11]
midori2[rows_to_change2,15:17] <- midori2[rows_to_change2,9:11]

# Deleting wrong values
midori1[rows_to_change1,9:11] <- NA
midori2[rows_to_change2,9:11] <- NA

# Identifying which rows to shift
rows_to_change1 <- midori1$V13 %in% "genus"
rows_to_change2 <- midori2$V13 %in% "genus"

# Moving columns one space to the right
midori1[rows_to_change1,15:17] <- midori1[rows_to_change1,6:8]
midori2[rows_to_change2,15:17] <- midori2[rows_to_change2,6:8]

# Deleting wrong values
midori1[rows_to_change1,6:8] <- NA
midori2[rows_to_change2,6:8] <- NA

# see where family level information got shifted and correct it

# Identifying which rows to shift
rows_to_change1 <- midori1$V16 %in% "family"
rows_to_change2 <- midori2$V16 %in% "family"

# Moving columns one space to the right
midori1[rows_to_change1,12:14] <- midori1[rows_to_change1,9:11]
midori2[rows_to_change2,12:14] <- midori2[rows_to_change2,9:11]

# Deleting wrong values
midori1[rows_to_change1,9:11] <- NA
midori2[rows_to_change2,9:11] <- NA

# Identifying which rows to shift
rows_to_change1 <- midori1$V13 %in% "family"
rows_to_change2 <- midori2$V13 %in% "family"

# Moving columns one space to the right
midori1[rows_to_change1,12:14] <- midori1[rows_to_change1,6:8]
```

```
midori1[rows_to_change1,12:14] <- midori1[rows_to_change1,6:8]
midori2[rows_to_change2,12:14] <- midori2[rows_to_change2,6:8]

# Deleting wrong values
midori1[rows_to_change1,6:8] <- NA
midori2[rows_to_change2,6:8] <- NA

# see where order level information got shifted and correct it

# Identifying which rows to shift
rows_to_change1 <- midori1$V13 %in% "order"
rows_to_change2 <- midori2$V13 %in% "order"

# Moving columns one space to the right
midori1[rows_to_change1,9:11] <- midori1[rows_to_change1,6:8]
midori2[rows_to_change2,9:11] <- midori2[rows_to_change2,6:8]

# Deleting wrong values
midori1[rows_to_change1,6:8] <- NA
midori2[rows_to_change2,6:8] <- NA

# Check if class level info got shifted

dim(midori1[!midori1$V10 %in% "phylum",]) # zero rows
dim(midori2[!midori2$V10 %in% "phylum",]) # zero rows

# Information for class level and above did not get shifted #

# Combine the two tables

midori<-rbind(midori1,midori2)

# Some empty "" cells remain after processing, set them as NA

midori[midori==""] <- NA

# Name the taxonomy columns

colnames(midori)[c(1,2,3,6,9,12,15,18)]<-
c("MOTU","Kingdom","Phylum","Class","Order","Family","Genus","Species"
)

# Write table to file

write.table(midori,"midori_pre_70.txt",sep="\t",row.names = F)
```

The final filtering to remove taxonomic information with a classification confidence of below 0.7 at the respective level from the formatted table (*midori_pre_70.txt*) was done in *Excel.* In order to do this, he column giving the confidence values for phylum level assignments was first sorted from highest to lowest. All assignments with a confidence value below our chosen threshold of 0.7 were set to NA (This was done for the phylum level confidence column, the two columns left of it and all columns right of it. The procedure was then repeated for all confidence value columns down to species level). The resulting file was saved as *midori_70.txt* (70 = confidence threshold of 0.7).

For *BOLDigger*, we ran a four-step protocol via its command line tool with the commands below (the non-commandline tool has more extensive documentation: https://github.com/DominikBuchner/BOLDigger):

1. Taxonomy assignment with the fasta file containing the curated MOTUs using the *BOLD identification engine* with the COI database and a batch size of 50. It may happen that reaching *BOLD* fails with high batch sizes, try batch sizes of 5 or 10 in this case. Log in details for *BOLD* user account need to be provided (account can be created via *BOLD*'s website prior to this).
2. *Searching for additional data* with the BOLDresults file generated in step 1.
3. *Add a list of top hits* with *BOLDigger* method.
4. *BOLD API verification*. "This option scans the BOLDigger top hits for hits with high similarity (>= 98%) and missing species-level assignment. This can happen, if the top 20 hits are populated with high similarity hits and missing species-level assignment "hide" away better hits." (from: https://github.com/DominikBuchner/BOLDigger)

Command

Install and run BOLDigger commandline tool

```
pip install boldigger_cline

cd ~/COI

# Run classification with default batch size of 50
boldigger-cline ie_coi username password
COI_cluster_reps_lulu_curated.fa ~/COI 50

# Download additional data from BOLD
boldigger-cline add_metadata
BOLDResults_COI_cluster_reps_lulu_curated_part_1.xlsx

# Get top hit with BOLDigger method
boldigger-cline digger_hit
BOLDResults_COI_cluster_reps_lulu_curated_part_1.xlsx

# Perform API correction
boldigger-cline api_verification
BOLDResults_COI_cluster_reps_lulu_curated_part_1.xlsx
COI_cluster_reps_lulu_curated_done.fa
```

The classification step of *BOLDigger-commandline* may terminate with an error for sequences where entries in BOLD produce a database error on the BOLD website (see https://github.com/DominikBuchner/BOLDigger/issues/24 and https://github.com/DominikBuchner/BOLDigger/issues/14, for example). This happened twice with our fatsa file (for ASV858 and ASV7885). When this happened, we executed the classification step again with a batch_size of 1 until the script got terminated again. We then manually removed the sequence following the last classified sequence from the fasta file and executed the command again with a batch_size of 50 until the next error occurred. After the *BOLDigger-commandline* procedure, we manually added these two sequences to the *BOLDigger hit - API corrected* sheet of the resulting .xlsx file and set all rank assignments for these two cases as *No Match*.

We compared taxonomy resulting from *MIDORI2* and *BOLDigger* and created a final taxonomy table. For MOTUs where *MIDORI2* and *BOLDigger* agreed at the respective level, this

information was kept. For levels of a MOTU where the two disagreed, the information of "BOLDigger hit - API corrected" was kept at Phylum, Class or Order level if the similarity was above 85, at Family level if similarity was above 90, at Genus level if similarity was above 95, and at Species level if similarity was above 98. Where *BOLDigger* had NA entries and *MIDORI2* non-NA entries, *MIDORI2* classification was kept for the respective level of a MOTU. Where the two disagreed but BOLDigger did not meet the threshold for the respective level of a MOTU, entries were set as NA. These steps were performed in *R* / *RStudio* with the following commands:

Command

COI taxonomy table based on BOLDigger and MIDORI2 in R

```
library(readxl)
library(tidyr)

# Set working directory

setwd("~/COI")

### Compare Midori2 and BOLDigger taxonomy and keep agreeing taxonomy

# Read files

# Midori file
# After removing the entries with confidence below 0.7, the file was
saved as "midori_70.txt"
midori<-read.table("midori_70.txt",sep="\t",header=T)
midori<-midori[,c(1:3,6,9,12,15,18)] # Keep only taxonomy level
columns

# Boldigger file
# Read the API corrected sheetand remove the ">" in MOTU IDs
bold<-
as.data.frame(read_xlsx("BOLDResults_COI_cluster_reps_lulu_curated_par
t_1.xlsx",sheet="BOLDigger hit - API corrected"))
bold[,1]<-gsub(">","",bold[,1])
colnames(bold)[1]<-"MOTU"

# In case the midori table hasn't been reordered in Excel, order
entries based on order in BOLDigger table
midori<-midori[order(match(midori[,1],bold[,1])),]
write.table(midori,"midori_70.txt",sep="\t",row.names = F)

# Separate the Species column of midori table with separate_wider-
delim with names_sep="".
# As many columns as needed will be created, because some species
name have a CMCxy suffix etc and we can remove them through this
procedure.

midori<-midori %>% separate_wider_delim(Species, delim = " ",
names_sep="",too_few = "align_start")
midori<-as.data.frame(midori[,c(1:7,9)]) # remove unnecessary columns
```

```
# Create column with genus and species level in one string
midori$Species<-paste(midori$Genus,midori$Species2,sep=" ")
midori$Species<-gsub(".*( NA|NA ).*", NA, midori$Species) # Replace
entries that now contain the " NA" or "NA " string with NA
midori<-midori[,-8] # Remove Species2 column

bold$Species<-paste(bold$Genus,bold$Species,sep=" ")
bold$Species<-gsub(".*( NA|NA ).*", NA, bold$Species) # Replace
entries that now contain the " NA" or "NA " string with NA

# To simplify downstream code, set NA entries in the Midori table as
character string (we chose "unknown")
midori[is.na(midori)] <- "unknown"

## Generate final taxonomy
# For MOTUs where MIDORI2 and BOLDigger agreed at the respective
level, this information was kept.
# For levels of a MOTU were the two disagreed, the information of
"BOLDigger hit - API corrected" was kept at Phylum, Class or Order
level if the similarity was above 85, at Family level if similarity
was above 90, at Genus level if similarity was above 95, and at
Species level if similarity as above 98.
# Where BOLDigger had NA entries and MIDORI2 non-NA entries, MIDORI2
classification was kept for the respective level of a MOTU.
# Where the two disagreed but BOLDigger did not meet the threshold
for the respective level of a MOTU, entries where set as NA.

Phylum <-ifelse(midori$Phylum == bold$Phylum | (bold$Similarity>85 &
bold$Phylum!=midori$Phylum),bold$Phylum,ifelse((is.na(bold$Phylum) |
bold$Phylum == "No Match") &
!is.na(midori$Phylum),midori$Phylum,"NA"))
Class <-ifelse(midori$Class == bold$Class | (bold$Similarity>85 &
bold$Class!=midori$Class),bold$Class,ifelse((is.na(bold$Class) |
bold$Class == "No Match") & !is.na(midori$Class),midori$Class,"NA"))
Order <-ifelse(midori$Order == bold$Order | (bold$Similarity>85 &
bold$Order!=midori$Order),bold$Order,ifelse((is.na(bold$Order) |
bold$Order == "No Match") & !is.na(midori$Order),midori$Order,"NA"))
Family <-ifelse(midori$Family == bold$Family | (bold$Similarity>90 &
bold$Family!=midori$Family),bold$Family,ifelse((is.na(bold$Family) |
bold$Family == "No Match") &
!is.na(midori$Family),midori$Family,"NA"))
Genus <-ifelse(midori$Genus == bold$Genus | (bold$Similarity>95 &
bold$Genus!=midori$Genus),bold$Genus,ifelse((is.na(bold$Genus) |
bold$Genus == "No Match") & !is.na(midori$Genus),midori$Genus,"NA"))
Species <-ifelse(midori$Species == bold$Species | (bold$Similarity>98
```

```
&
bold$Species!=midori$Species),bold$Species,ifelse((is.na(bold$Species)
 | bold$Species == "No Match") &
```

Merging same-species MOTUs

10 MOTUs which could be classified down to species level and which displayed the same species level assignment were merged. Here, read counts of corresponding same-species MOTUs were summed up and the representative sequence of the most abundant MOTU prior to merging was kept. The following script was executed in *R / RStudio*:

Command

Merge same-species MOTUs in R

```
library(stringr)
library(dplyr)

### COI ###

setwd("~/COI/")

# Read the Midori/ BOLDIgger taxonomy file

tax_table<-
read.table("boldigger_midori_tax_table.txt",sep="\t",header =
T,stringsAsFactors = F)

# Read LULU curated MOTU count table.

motu_tab<-
read.table("lulu_motu_table_COI.txt",header=T,check.names=F,stringsAsF
actors = F, sep="\t")

# Sort the count table based on the order in the tax table and
combine both tables

motu_tab<-motu_tab[order(match(motu_tab[,1],tax_table[,1])),]

motu_tax_tab<-
as.data.frame(cbind(tax_table,motu_tab[,-1]),stringsAsFactors=F)

# Write this MOTU table to file

write.table(motu_tax_tab,"COI_motu_table_tax_counts_species_fullname.t
xt",sep="\t",row.names = F)

## Aggregate MOTUs with same species assignment, adding read counts
of same-species MOTUs to the most abundant same-species MOTU. ##

# Sort motu_tax_tab by read abundance

motu_tax_tab <-
motu_tax_tab[order(rowSums(motu_tax_tab[,9:ncol(motu_tax_tab)]),decrea
sing=TRUE),]
```

```
# Create a mapping file for subsequent downstream analysis to track
which MOTUs are going to be merged.

motu_map<-aggregate(MOTU ~ Species, data = motu_tax_tab, paste,
collapse = ",")
write.table(motu_map,"motu_map_identical_species.txt",sep="\t",row.nam
es = F)

# Aggregate rows by species entries and sum read counts.

tax_sum<-aggregate(.~ motu_tax_tab$Species, data =
motu_tax_tab[,9:ncol(motu_tax_tab)], sum)

# Combine this table with the mapping table

motu_unique<-as.data.frame(cbind(motu_map,tax_sum[,-1]))

# Where same-species MOTUs have been merged, we only want to keep the
MOTU ID of the most abundant one.
# Where ID entries with aggregated MOTUs exist, the first entry
equals the most abundant MOTU so we delete the characters after (and
incl.) the comma.

motu_unique$MOTU<-gsub(",.*","",motu_unique$MOTU)

# Filter the previously generated motu_tax_tab table to only contain
the MOTUs now listed in the motu_unique table.

motu_tax_tab_uniq<-motu_tax_tab[motu_tax_tab$MOTU %in%
motu_unique$MOTU,]

# Order the motu_unique table to match the MOTU ID order of
motu_tax_tab_uniq and create a final MOTU table with the taxonomy
info of motu_tax_tab_uniq and the summed up read counts of
motu_unique

motu_unique<-
motu_unique[order(match(motu_unique[,2],motu_tax_tab_uniq[,1])),]
final_motu_tab<-
as.data.frame(cbind(motu_tax_tab_uniq[,1:8],motu_unique[,-c(1,2)]))

# Combine it with the entries that were not part of the aggregation
procedure, e.g. MOTUs with no species level assignment

final_motu_tab<-
```

```
rbind(final_motu_tab,motu_tax_tab[is.na(motu_tax_tab$Species),])
```

Determining sites / groups of field replicates

11 ARMS-MBON consists of observatories regularly deploying ARMS at one to seven sights each. Within an observatory, sites are locations with one or more ARMS being deplyoed with a certain distance from each other. However, during the initial phase of ARMS-MBON, there was no clear definition for what requirements define a group of ARMS units as field replicates.

For downstream analysis purposes, we consider ARMS units deployed with a distance of maximum 20 km from each other as one site = one Field_Replicate group. We took information on coordinates of each deployed ARMS unit from the https://github.com/arms-mbon/data_workspace/blob/main/qualitycontrolled_data/combined/combined_ObservatoryData.csv file from the ARMS-MBON GitHub repository.

The follwoing script calculates distances in km in *R/RStudio* for each ARMS unit combination:

Command

Calculate distances between ARMS in R

```
library(dplyr)
library(geosphere) # install.packages("geosphere")

# prepare metadata
meta<-read.csv("combined_ObservatoryData.csv",header = T) # read csv
file with info on observatories available on ARMS-MBON GitHub repo
meta<-meta %>% select(c(ObservatoryID,UnitID,Longitude,Latitude)) #
subset necessary columns
meta<-meta[order(meta$ObservatoryID),]
rownames(meta)<-meta$UnitID
meta<-meta[,-(1:2)]

# calculate distances
distances<-distm(meta)
distances<-distances/1000 # distances are given in meters and have to
be changed to km
rownames(distances)<-rownames(meta)
colnames(distances)<-rownames(meta)
write.table(distances,"distances.txt",sep="\t",col.names = NA)
```

The resulting file is a distance matrix and contains more ARMS then present in the data set processed here, as there are more observatories in ARMS-MBON than data was available for at this time. We manually checked the distances between the ARMS units which are part of our data set and manually added the column *Field_Replicate* to the *XX_sample_data.txt* files used in the next section (see below). In this column, we defined a code for each site = Field_Replicate group an ARMS unit belongs to.

```
meta_unique
```

Data processing and cleaning with *phyloseq* etc.

12 Further data processing and cleaning was done in *R / RStudio* using mainly the *phyloseq* package (and other packages, see script below). The COI and 18S data sets were processed separately. Below are the metadata files used as sample_data in the respective phyloseq objects (the information was taken from the respective files available on the ARMS-MBON

GitHub repository: https://github.com/arms-mbon/data_workspace/blob/main/qualitycontrolled_data/combined/combined_ObservatoryData.csv, https://github.com/arms-mbon/data_workspace/blob/main/qualitycontrolled_data/combined/combined_SamplingEventData.csv and https://github.com/arms-mbon/data_workspace/blob/main/qualitycontrolled_data/combined/combined_OmicsData.csv):

COI_sample_data.txt 61KB

18S_sample_data.txt 62KB

The following steps were carried out for both data sets:

1. Create phyloseq objects based on the count and taxonomy tables created during the merging of same-species MOTUs in the previous section, as well as a table containing metadata for the samples (see above).
2. The MOTUs are still named by their representative ASV sequences. They will be renamed to MOTUxy and a mapping file is created to link each MOTU name to its representative ASV sequence.
3. Check and plot sample-wise sequencing depth (i.e., read numbers) for each sequencing run.
4. Removal of certain sample types: blank / negative control samples; sediment / plankton samples which were sequenced as a trial during the initial phase of the ARMS MBON program; and any samples without any reads remaining after all previous pipeline steps. **This is now the most unfiltered data set for each marker gene which will be saved!**

Command

18S - data processing and cleaning in R

```
library(phyloseq)
library(ggplot2)
library(data.table)


setwd("~/18S")

# Read MOTU counts

MOTUcounts18S<-
read.table("18S_motu_count_table_merged_species.txt",header=T,check.na
mes=F, sep="\t")

# Read MOTU taxonomy (needs to be read as matrix, may cause problems
otherwise when phyloseq object will be created)

MOTUtaxa18S<-
as.matrix(read.table("18S_motu_tax_table_merged_species.txt",header=T,
check.names=F, sep="\t"))

# Sort count table based on order in tax table (precautionary measure)

MOTUcounts18S<-
MOTUcounts18S[order(match(MOTUcounts18S[,1],MOTUtaxa18S[,1])),]

# MOTUs are still named "ASVxy". Replace them with MOTUxy and set
them as rownames.

# First, create and write mapping file (MOTUxy = ASVyz).
mapping<-cbind(MOTUcounts18S[,1],paste0("MOTU",
seq(1:nrow(MOTUcounts18S))))
colnames(mapping)[1:2]<-c("ASV","MOTU")
write.table(mapping,"motu_asv_mapping.txt",sep="\t",row.names = F)

rownames(MOTUcounts18S) <- mapping[,2]
MOTUcounts18S[,1] <- NULL

rownames(MOTUtaxa18S) <- mapping[,2]
MOTUtaxa18S<-MOTUtaxa18S[,-1] # setting it as NULL does not work for
matrix object
```

```
# Read sample metadata
# Note that there may be more samples left with zero reads. In this
case, these samples passed the filterAndTrim step with reads, but
they ended up with zero reads during the subsequent dada2 steps.

MOTUsample18S<-
read.table("18S_sample_data.txt",header=T,check.names=F,row.names=1,se
p="\t",strip.white = T)

# Create phyloseq object

ps18S <- phyloseq(otu_table(MOTUcounts18S,taxa_are_rows = TRUE),
sample_data(MOTUsample18S), tax_table(MOTUtaxa18S))

## Get a quick overview of sequencing depth per sequencing run ##

# Make data.table for plot

read_sums<- data.table(as(sample_data(ps18S), "data.frame"),
                       TotalReads = sample_sums(ps18S), keep.rownames
= TRUE)
setnames(read_sums,"rn","SampleID")

# Violin plot based on sequencing events

reads_plot <- ggplot(read_sums,
aes(y=TotalReads,x=Sequenced,color=Sequenced)) +
  geom_violin() +
  geom_jitter(shape=16, position=position_jitter(0.2),size=1) +
  scale_y_continuous(labels = scales::comma)+
  scale_x_discrete(limits=c("Jul-19","Jan-20","Sep-20","Apr-21","May-
21","Jan-22","Aug-23"))+
  theme_bw()+
  theme(legend.position = "none")+
  ggtitle("Sequencing Depth")

reads_plot

ggsave(reads_plot,file="18S_sequencing_depth_runs.png",height =
4,width = 6)

##

# Remove the negative controls
```

```
ps18S_cleaned <- subset_samples(ps18S,ARMS!="blank")

# Remove the sediment and plankton samples (some sediment and
plankton samples were sequenced as a trial during the initial phase
of the ARMS program)

ps18S_cleaned <- subset_samples(ps18S_cleaned,Fraction!="SED" &
Fraction!="PS")

# Remove samples with a read number of zero

ps18S_cleaned <- prune_samples(sample_sums(ps18S_cleaned) > 0,
ps18S_cleaned)

# Remove MOTUs which have a total abundance of zero after removing
samples during all of the previous steps

ps18S_cleaned<-
prune_taxa(rowSums(otu_table(ps18S_cleaned))>0,ps18S_cleaned)

# Save this phyloseq object as the most unfiltered ARMS data set

saveRDS(ps18S_cleaned,"ps18S_unfiltered_ARMS.rds")

# get number of remaining samples and MOTUs

ps18S_cleaned

# get number of reads

sum(sample_sums(ps18S_cleaned))

# Get percentage of MOTUs classified at phylum level

subset_taxa(ps18S_cleaned,!is.na(Phylum))

6465/13771

# get percentage of reads classified at phylum level

sum(sample_sums(subset_taxa(ps18S_cleaned,!is.na(Phylum))))/sum(sample
_sums(ps18S_cleaned))

# Get percentage of MOTUs classified at species level

subset_taxa(ps18S_cleaned,!is.na(Species))
```

```
1076/13771

# get percentage of reads classified at phylum level

sum(sample_sums(subset_taxa(ps18S_cleaned,!is.na(Species))))/sum(sampl
e_sums(ps18S_cleaned))
```

Command

COI - data processing and cleaning in R

```
library(phyloseq)
library(ggplot2)
library(data.table)

setwd("~/COI")

# Read MOTU counts

MOTUcountsCOI<-
read.table("COI_motu_count_table_merged_species.txt",header=T,check.na
mes=F, sep="\t")

# Read MOTU taxonomy (needs to be read as matrix, may cause problems
otherwise when phyloseq object will be created)

MOTUtaxaCOI<-
as.matrix(read.table("COI_motu_tax_table_merged_species.txt",header=T,
check.names=F, sep="\t"))

# Sort count table based on order in tax table (precautionary measure)

MOTUcountsCOI<-
MOTUcountsCOI[order(match(MOTUcountsCOI[,1],MOTUtaxaCOI[,1])),]

# MOTUs are still named "ASVxy". Replace them with MOTUxy and set
them as rownames.

# First, create and write mapping file (MOTUxy = ASVyz).
mapping<-cbind(MOTUcountsCOI[,1],paste0("MOTU",
seq(1:nrow(MOTUcountsCOI))))
colnames(mapping)[1:2]<-c("ASV","MOTU")
write.table(mapping,"motu_asv_mapping.txt",sep="\t",row.names = F)

rownames(MOTUcountsCOI) <- mapping[,2]
MOTUcountsCOI[,1] <- NULL

rownames(MOTUtaxaCOI) <- mapping[,2]
MOTUtaxaCOI<-MOTUtaxaCOI[,-1] # setting it as NULL does not work for
matrix object
```

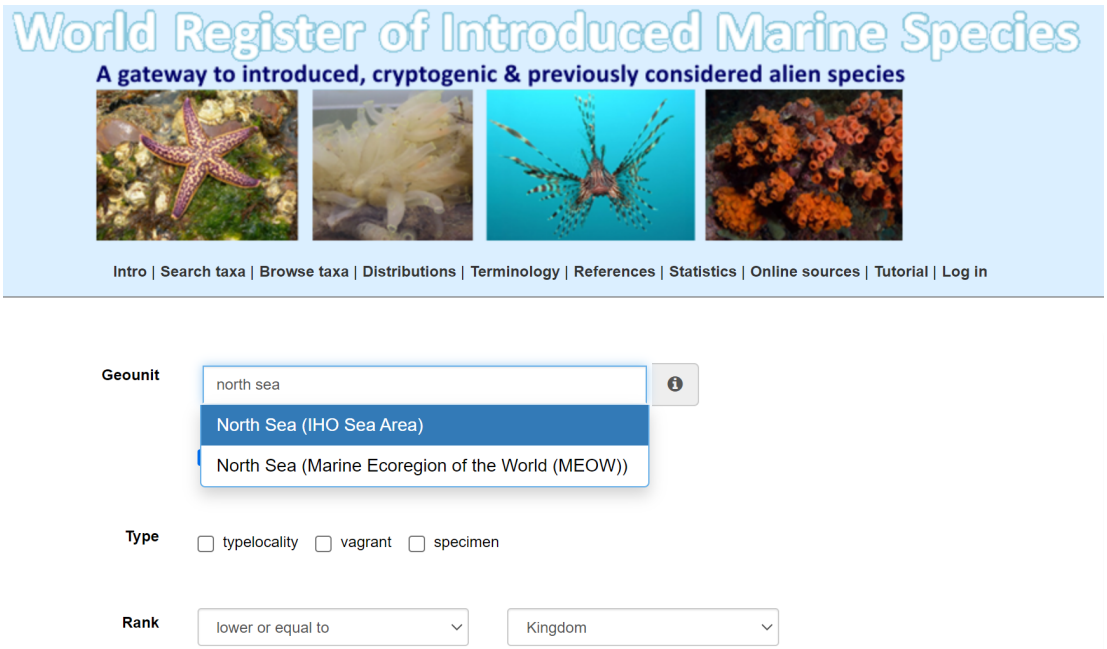
```
# Read sample metadata

MOTUsampleCOI<-
read.table("COI_sample_data.txt",header=T,row.names=1,check.names=F,
sep="\t",strip.white = T)

# Create phyloseq object

psCOI <- phyloseq(otu_table(MOTUcountsCOI,taxa_are_rows = TRUE),
sample_data(MOTUsampleCOI), tax_table(MOTUtaxaCOI))

## Get a quick overview of sequencing depth per sequencing run ##

# Make data.table for plot

read_sums<- data.table(as(sample_data(psCOI), "data.frame"),
                       TotalReads = sample_sums(psCOI), keep.rownames
= TRUE)
setnames(read_sums,"rn","SampleID")

# Violin plot based on sequencing events

reads_plot <- ggplot(read_sums,
aes(y=TotalReads,x=Sequenced,color=Sequenced)) +
  geom_violin() +
  geom_jitter(shape=16, position=position_jitter(0.2),size=1) +
  scale_x_discrete(limits=c("Jul-19","Jan-20","Sep-20","Apr-21","May-
21","Jan-22","Aug-23"))+
  theme_bw()+
  theme(legend.position = "none")+
  ggtitle("Sequencing Depth")

reads_plot

ggsave(reads_plot,file="COI_sequencing_depth_runs.png",height =
4,width = 6)

##

# Remove the negative controls

psCOI_cleaned <- subset_samples(psCOI,ARMS!="blank")

# Remove the sediment samples and plankton samples (some sediment and
plankton samples were sequenced as a trial during the initial phase
of the ARMS program)
```

```
psCOI_cleaned <- subset_samples(psCOI_cleaned,Fraction!="SED" &
Fraction!="PS")
```

Identify non-indigenous species based on WRiMS

13 Putative non-indigenous species (NIS) were identified from the MOTUs which could be classified to species level.

Species listed in the World Register of Introduced Marine Species (WRiMS) were obtained for the regions corresponding to the ARMS locations. WRiMS' *Distributions* browser was used for this (https://www.marinespecies.org/introduced/aphia.php?p=checklist). Multiple searches were performed with default settings and the respective region specified in the field *Geonuit* (see image below).



The interface of WRiMS' *Distributions* browser.

To keep the search standardized, it was carried out considering IHO Sea Areas if possible. However, there were no entries for some IHO Areas, e.g. the IHO Sea Area *Skagerrak* was not listed in WRiMS. Hence, in such cases Marine Regions were selected for the search instead. For the case of Skagerrak, this was the Marine Region *Swedish part of the Skagerrak* for example. In other cases, the IHO Areas comprised too large a geographical area (e.g., *North Atlantic Ocean*) containing regions irrelevant for this study. Therefore, the search was performed for subordinate IHO Sea Areas or, alternatively, Marine Regions within the respective region. WRiMS only allows downloads of files with a maximum of 1,000 entries. The IHO Sea Area *Mediterranean Sea - Eastern Basin* had more than 1,000 entries, so subordinate regions

had to be selected for separate search queries (*Adriatic Sea* and *Aegean Sea* in this case). The table below shows the regions and their *PlaceType* according to Marine Regions (**https://www.marineregions.org/**) for which the taxa listed in WRiMS were obtained.

	Geounit	Marine Regions PlaceType
	Adriatic Sea	IHO Sea Area
	Aegean Sea	IHO Sea Area
	Arctic Ocean	IHO Sea Area
	Baltic Sea	IHO Sea Area
	English Channel	IHO Sea Area
	Irish part of the North Atlantic Ocean	Marine Region
	North Sea	IHO Sea Area
	Norwegian part of the Norwegian Sea	Marine Region
	Red Sea	IHO Sea Area
	Spanish part of the Bay of Biscay	Marine Region
	Spanish part of the North Atlantic Ocean	Marine Region
	Swedish part of the Skagerrak	Marine Region

Regions and their *PlaceType* according to Marine Regions (**https://www.marineregions.org/**) for which the taxa listed in WRiMS were obtained.

The results for each search were downloaded as .xlsx files (with the pre-selected default columns included). See images below.

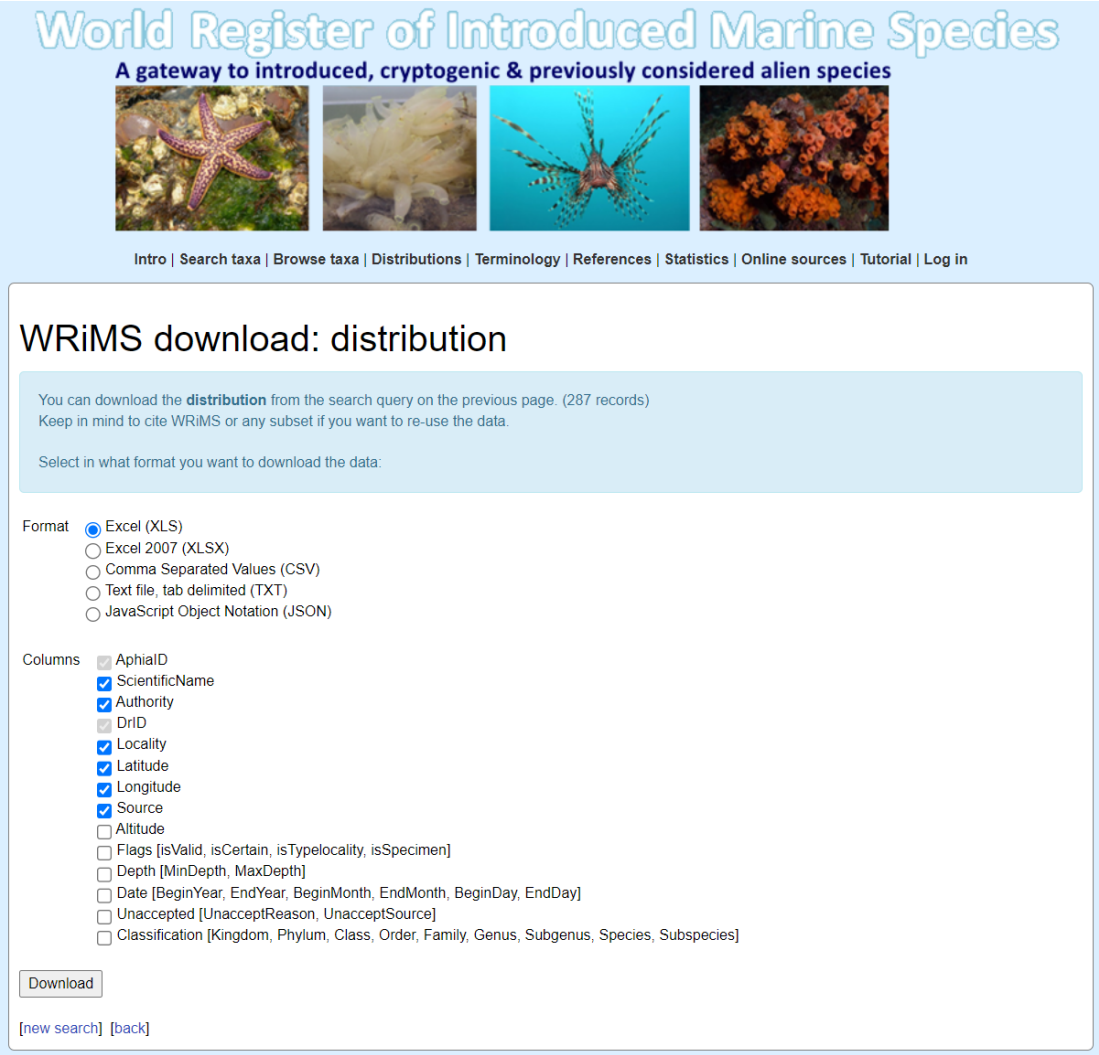
WRiMS Distribution

287 matching records, showing records 1-100.
Click on one of the taxon names listed below to check details for that taxon [new search] [direct link] [download results]

Acartia (Acanthacartia) tonsa Dana, 1849 (origin: alien)

Acartia tonsa Dana, 1849 represented as *Acartia (Acanthacartia) tonsa* Dana, 1849 (origin: alien)

Download link for the search results of each region's listed taxa.



Download interface of WRiMS' *Distributions* browser.

The downloaded results were then combined in *R / RStudio* to an overall list of taxa listed in *WRiMS* for the study regions. See the script below:

Command

Create overall WRiMS NIS list in R

```
library(xlsx)
library(tidyr)

setwd("~/NIS")

# Read the tables downloaded from WRiMS

list.files() # Briefly check the file names

arctic<-read.xlsx("Arctic Ocean IHO Sea Area.xlsx",sheetIndex = 1)
norway<-read.xlsx("Norwegian part of the Norwegian Sea Marine
Region.xlsx",sheetIndex = 1)
northsea<-read.xlsx("North Sea IHO Sea Area.xlsx",sheetIndex = 1)
skagerrak<-read.xlsx("Swedish part of the Skagerrak Marine
Region.xlsx",sheetIndex = 1)
baltic<-read.xlsx("Baltic Sea IHO Sea Area.xlsx",sheetIndex = 1)
channel<-read.xlsx("English Channel IHO Sea Area.xlsx",sheetIndex = 1)
biscaya<-read.xlsx("Spanish part of the Bay of Biscay Marine
Region.xlsx",sheetIndex = 1)
spain<-read.xlsx("Spanish part of the North Atlantic Ocean Marine
Region.xlsx",sheetIndex = 1)
adriatic<-read.xlsx("Adriatic Sea IHO Sea Area.xlsx",sheetIndex = 1)
aegean<-read.xlsx("Aegean Sea IHO Sea Area.xlsx",sheetIndex = 1)
irish<-read.xlsx("Irish part of the North Atlantic Ocean Marine
Region.xlsx",sheetIndex = 1)

# Combine all tables

all_lists<-
rbind(arctic,norway,northsea,skagerrak,baltic,channel,biscaya,spain,ad
riatic,aegean,redsea,irish)

# Remove unnecessary columns

nis<-all_lists[,-c(3,4,6:8)]

# Aggregate by species and Aphia ID to get a column with a string
containing all localities the species is considered a NIS

nis<-aggregate(Locality ~ AphiaID + ScientificName, data = nis,
```

```
paste, collapse = ",")

# Separate the strings in Locality column into single columns

nis<-separate_wider_delim(nis,Locality, delim = ",",
names_sep="",too_few = "align_start")

# Write overall NIS table to file

write.table(nis,"WRiMS_ARMS_locations_NIS_List.txt",sep="\t",row.names
 = F)
```

The result of this was the following file:

WRiMS_ARMS_locations_NIS_List.txt 202KB

The 18S and COI MOTUs which could be classified to species level were then scanned for taxa found in the WRiMS list. This was done for the *phyloseq* data sets created in the previous section. The taxa names in the reference sets used for taxonomic classification do not always equal the accepted names in WRiMS / WoRMS. Therefore, the species-level assignments in the 18S and COI data sets were first matched against the WoRMS database to obtain the respective accepted names for the taxa. *LifeWatch Belgium*'s web services were used for the taxon matching (see **https://www.lifewatch.be/data-services/**). These web services actually include a taxon match service for WRiMS, but is was not functional at the time of usage. This may have been fixed by the time of publication of this protocol. First, a .txt file fulfilling the input requirements of the web service (i.e., MOTU IDs and the respective species names) was generated in *R / RStudio* for each marker gene. See script below:

Command

Generate files in R for LifeWatch's WoRMS Taxon Match service

```
library(phyloseq)

### COI ###

setwd("~/COI")

# Load most unfiltered phyloseq object

unfiltered<-readRDS("psCOI_unfiltered_ARMS.rds")

# The unfiltered data set is the least exclusive, i.e., all MOTUs
appearing in this data set are found in all potential filtered data
sets
# Get a table of the unfiltered data set with MOTU names and Genus
and Species assignments in one column for taxa classified down to
species level

taxa<-
cbind(rownames(tax_table(subset_taxa(unfiltered,!is.na(Species)))),tax
_table(subset_taxa(unfiltered,!is.na(Species)))[,7])
colnames(taxa)<-c("MOTU","ScientificName") # The LifeWatch e-Lab
service used later on needs the Species column to be named
"ScientificName"

# Write table to file which will be checked for correct species names
in WoRMS via LifeWatch Belgium's e-Lab services
https://www.lifewatch.be/data-services/

write.table(taxa,"ARMS_COI_species_check_WoRMS.txt",sep="\t",quote=F,r
ow.names = F)

### 18S ###

setwd("~/18S")

# Load most unfiltered phyloseq object

unfiltered<-readRDS("ps18S_unfiltered_ARMS.rds")

# The unfiltered data set is the least exclusive, i.e., all MOTUs
```

```
appearing in this data set are found in all potential filtered data
sets
# Get a table of the unfiltered data set with MOTU names and Genus
and Species assignments in one column for taxa classified down to
species level

gen_spec<-subset_taxa(unfiltered,!is.na(Species))
tax_table(gen_spec)[,10]<-gsub("_"," ",tax_table(gen_spec)[,10])
taxa<-cbind(rownames(tax_table(gen_spec)),tax_table(gen_spec)[,10])
colnames(taxa)<-c("MOTU","ScientificName") # The LifeWatch e-Lab
service used later on needs the Species column to be named
"ScientificName"

# Write table to file which will be checked for correct species names
in WoRMS via LifeWatch Belgium's e-Lab services
https://www.lifewatch.be/data-services/

write.table(taxa,"ARMS_18S_species_check_WoRMS.txt",sep="\t",quote=F,r
ow.names = F)
```

To run LifeWatch Belgium's e-Lab services, an account is required. Under "Run Services", the files just created were used as input and the service "Taxon match services" -> "Taxon match World Register of Marine Species (WoRMS)" was applied. The results were downloaded and then used to scan the *phyloseq* data sets in *R / RStudio* for taxa found in the WRiMS list. See script below:

Command

Scan 18S and COI for NIS recorded in WRiMS

```
library(phyloseq)
library(dplyr)

### COI ###

setwd("~/COI")

# Load phyloseq object

unfiltered<-readRDS("psCOI_unfiltered_ARMS.rds")

# read the results file from the Taxon match World Register of Marine
Species (WoRMS) service of LifeWatch

worms_results<-
read.table("result_arms_coi_species_check_worms.txt",sep="\t",header=T
)

# In some cases, there were some fuzzy or non-exact matches of our
taxa names vs. the ones found in WoRMS
# In such cases, the entry in the column "accepted_name_aphia_worms"
will be empty
# replace these entries with our original names from the
"scientificname" column

worms_results$accepted_name_aphia_worms<-
ifelse(worms_results$accepted_name_aphia_worms=="",worms_results$scien
tificname,worms_results$accepted_name_aphia_worms)

# Read prepared NIS list

nis<-read.table("WRiMS_ARMS_locations_NIS_List.txt",sep="\t",header=T)

# Make table with NIS listed in WRiMS which are found in our data set

nis_found<-worms_results %>% filter(accepted_name_aphia_worms %in%
nis$ScientificName)

# Subset phyloseq object to these MOTUs and write them to file
```

```
unfiltered_nis<-prune_taxa(nis_found$motu,unfiltered)
saveRDS(unfiltered_nis,"unfiltered_COI_set_NIS.rds")

# Make tables with NIS found in the phyloseq object with relevant
info and write to file

taxa_unfiltered <-worms_results %>% filter(motu %in%
taxa_names(unfiltered_nis))
colnames(nis)[2]<-"accepted_name_aphia_worms"
taxa_unfiltered<-merge(taxa_unfiltered, nis,
by="accepted_name_aphia_worms")
taxa_unfiltered<-taxa_unfiltered[,-c(3:11)]

write.table(taxa_unfiltered,"unfiltered_COI_set_NIS_identified.txt",se
p = "\t",row.names = F,quote=F)

## Get distribution and abundance information for the most unfiltered
NIS data set ##

# Make phyloseq object with samples merged for each ARMS sampling
event

variable1 = as.character(get_variable(unfiltered_nis, "ARMS"))
variable2 = as.character(get_variable(unfiltered_nis, "Deployment"))
variable3 = as.character(get_variable(unfiltered_nis, "Retrieval"))
sample_data(unfiltered_nis)$sample_event <- paste(variable1,
variable2, variable3,sep="_")
saveRDS(unfiltered_nis,"unfiltered_nis.rds")
unfiltered_nis_arms<-merge_samples(unfiltered_nis,"sample_event") #
ATTENTION: this transposes otu_table
otu_table(unfiltered_nis_arms)<-t(otu_table(unfiltered_nis_arms))
saveRDS(unfiltered_nis_arms,"unfiltered_nis_arms.rds")

# Combine count and taxonomy table and replace species name with
accepted Aphia name

nis_taxa_counts<-data.frame(tax_table(unfiltered_nis_arms)
[,7],otu_table(unfiltered_nis_arms),check.names = F)
for (i in 1:nrow(nis_found)) {
  nis_taxa_counts$Species[rownames(nis_taxa_counts) ==
nis_found$motu[i]] <- nis_found$accepted_name_aphia_worms[i]
}

write.table(nis_taxa_counts,"nis_taxa_counts_COI.txt",sep="\t",col.nam
es=NA)
```

```
### 18S ###
setwd("./18S")
# Load phyloseq object
```

The results of this script are count tables of MOTUs with species assignments (one each for COI and 18S) for taxa listed in WRiMS for the ARMS locations. Counts are given per ARMS sampling event (so all fraction samples were merged for each single ARMS deployment).

Tracking ASVs and MOTUs through the pipeline and manually curate identified NIS

14 All occurrences of NIS were carefully curated manually based on this two-step approach:

a) Taxonomic classification of short amplicons from DNA metabarcoding data sets is challenging and species level assignments may lack a certain confidence for closely related taxa. All ASVs ending up in MOTUs identified as NIS were manually classified again using *BOLD* (COI) or *NCBI*'s *GenBank* (18S, and in some rare cases COI) to assess confidence of taxonomic assignments.
b) Based on a literature search (i.e., checking sources mentioned in *WRiMS* for each respective species and performing additional web-based literature search) it was assessed if occurrences of MOTUs identified as NIS could actually be considered as being outside of their native range.

As a first step, the fate of ASVs (i.e., which MOTU they ultimately ended up in) was tracked throughout the entire pipeline using the previously generated files resulting from *swarm* clustering, *LULU* curation, merging of same-species MOTUs and mapping of MOTUs to their representative ASVs. Subsequently, this information was subset to ASVs belonging to NIS MOTUs to obtain their respective sequences and read counts per ARMS sampling event. See below for the script executed in *R / RStudio*:

Command

Track fate of ASVs throughout the pipeline in R

```
library(phyloseq)
library(dplyr)
library(tidyr)
library(data.table)

### COI ###

setwd("~/COI")

## Create final mapping file with all ASVs that have been placed in
one MOTU throughout the pipeline ##

# read LULU and swarm output files and the mapping file of merging
MOTUs with same species assignment

lulu<-read.table("motu_map_lulu_COI.txt",sep="\t",header=T)
swarm<-read.table("output.txt",sep="\t",stringsAsFactors = F)
spec_merge<-
read.table("motu_map_identical_species.txt",sep="\t",header=T)

# Separate swarm table into columns by space
# Remove read count strings from ASV names (lapply and function
necessary to do this for every entry in the table, not just specific
columns)

swarm<-separate_wider_delim(swarm,V1, delim = " ",
names_sep="",too_few = "align_start")
swarm[] <- lapply(swarm, function(y) gsub("_.*","", y))

# Order entries of first column in swarm table based on first column
in lulu table

swarm<-swarm[order(match(swarm$V11,lulu$X)),]

# Merge swarm and lulu tables

swarm_lulu_map<-cbind(lulu[,4],swarm)
colnames(swarm_lulu_map)[1]<-"MOTU"

# Separate MOTU strings in spec merge into separate columns
```

```
spec_merge<-separate_wider_delim(spec_merge,MOTU, delim = ",",
names_sep="",too_few = "align_start")

# Create ID column in spec_merge (= MOTU other MOTUs have been merged
onto) and transform to long format

spec_merge<-cbind(spec_merge[,c(2,2:ncol(spec_merge))])
colnames(spec_merge)[1]<-"ID"

spec_merge_long <- melt(setDT(spec_merge), id.vars = "ID",
variable.name = "string")
spec_merge_long<-spec_merge_long[,-2]
colnames(spec_merge_long)[2]<-"MOTU"
spec_merge_long<-spec_merge_long[!is.na(spec_merge_long$MOTU),]

## Merge swarm_lulu_map and spec_merge_long

swarm_lulu_spec_merge_map<-as.data.frame(merge(swarm_lulu_map,
spec_merge_long, by = "MOTU", all = TRUE))
swarm_lulu_spec_merge_map<-swarm_lulu_spec_merge_map %>% relocate(ID)

# Where ID is NA, fill in with entry of MOTU column. Then, remove
MOTU column.

swarm_lulu_spec_merge_map$ID<-
ifelse(is.na(swarm_lulu_spec_merge_map$ID),swarm_lulu_spec_merge_map$M
OTU,swarm_lulu_spec_merge_map$ID)
swarm_lulu_spec_merge_map<-swarm_lulu_spec_merge_map[,-2]

# Write all columns except ID column into one string, MOTU names
separated by comma

swarm_lulu_spec_merge_map$MOTU_string<-
apply(swarm_lulu_spec_merge_map[,2:ncol(swarm_lulu_spec_merge_map)],
1,paste, collapse=",")

# Remove ,NA strings (occurred during the previous step when empty
columns were pasted together)
# Keep only ID column and the MOTU_string column

swarm_lulu_spec_merge_map$MOTU_string<-gsub(",NA.*","",
swarm_lulu_spec_merge_map$MOTU_string)
swarm_lulu_spec_merge_map<-
swarm_lulu_spec_merge_map[,c(1,ncol(swarm_lulu_spec_merge_map))]
```

```
# Aggregate rows based on ID column

swarm_lulu_spec_merge_map<-aggregate(.~ ID, data =
swarm_lulu_spec_merge_map, paste, collapse = ",")

# Map ASVs to MOTUs

# motu_asv_mapping.txt stems from initial phyloseq processing
performed previously
mapping<-read.table("motu_asv_mapping.txt",sep="\t",header=T)
mapping<-
mapping[order(match(mapping[,1],swarm_lulu_spec_merge_map[,1])),]

swarm_lulu_spec_merge_map<-
cbind(mapping[,2],swarm_lulu_spec_merge_map)
colnames(swarm_lulu_spec_merge_map)[1:2]<-
c("MOTU","ASV_representative")
swarm_lulu_spec_merge_map<-
separate_wider_delim(swarm_lulu_spec_merge_map,MOTU_string, delim =
",", names_sep="",too_few = "align_start")

# Subset to MOTUs previously identified as NIS

unfiltered_nis_arms<-readRDS("unfiltered_nis_arms.rds")# Saved in
previous script when filtering MOTUs for NIS
nis_asv_motu_map<-swarm_lulu_spec_merge_map %>% filter(MOTU %in%
taxa_names(unfiltered_nis_arms))

# Create table mapping ASVs to those MOTUs and get respective ASV
read counts

nis_asv_motu_map <- melt(setDT(nis_asv_motu_map[,-2]), id.vars =
"MOTU", variable.name = "string")
nis_asv_motu_map<-nis_asv_motu_map[,-2]
colnames(nis_asv_motu_map)[2]<-"ASV"
nis_asv_motu_map<-nis_asv_motu_map[!is.na(nis_asv_motu_map$ASV),]
asv_counts<-
read.table("asv_no_contaminants_COI.txt",sep="\t",header=T,row.names
= 1) # read blank-corrected ASV count table
asv_counts<-asv_counts[rownames(asv_counts) %in%
nis_asv_motu_map$ASV,]

# Subset ASV counts to the samples present in NIS data set, set ASV
occurrences below 5 to zero and merge samples per ARMS sampling event

unfiltered_nis<-readRDS("unfiltered_nis.rds") # Saved in previous
script when filtering MOTUs for NIS
```

```
script when filtering MOTUs for NIS
nis_motu_physeq<-subset_samples(unfiltered_nis,sample_event %in%
sample_names(unfiltered_nis_arms)) # get sample names of NIS data set
with MOTUs showing at least 10 reads per ARMS sampling event
asv_counts<-
asv_counts[colnames(asv_counts)%in%sample_names(nis_motu_physeq)] #
Subset respective ASV count table to these samples
asv_counts[asv_counts < 5] <- 0
asv_counts<-asv_counts[rowSums(asv_counts[])>0,]
asv_counts<-asv_counts[, colSums(asv_counts != 0) > 0]
nis_asv_physeq<-phyloseq(otu_table(asv_counts,taxa_are_rows = TRUE),
sample_data(sample_data(nis_motu_physeq))) # make new phyloseq object
with ASV data set to merge samples per ARMS sampling event
nis_asv_physeq<-merge_samples(nis_asv_physeq,"sample_event") #
ATTENTION: this transposes otu_table
asv_nis_counts<-as.data.frame(t(otu_table(nis_asv_physeq)))

# Generate final table of NIS MOTUs, species names, the ASVs they
contain and the ASV read counts per ARMS sampling event

nis_asv_motu_map<-nis_asv_motu_map %>% filter(ASV %in%
rownames(asv_nis_counts))
nis_asv_motu_map<-
nis_asv_motu_map[order(match(nis_asv_motu_map$ASV,rownames(asv_nis_cou
nts))),]
asv_motu_counts<-cbind(nis_asv_motu_map,asv_nis_counts)
asv_motu_counts<-asv_motu_counts[order(asv_motu_counts$MOTU),]
nis_taxa_counts<-
read.table("nis_taxa_counts_COI.txt",sep="\t",header=T,row.names = 1)
# Read the nis_taxa_count table created in the previous NIS script to
get taxonomy information
# add species names for MOTUs
for (i in 1:nrow(nis_taxa_counts)) {
  asv_motu_counts$Species[asv_motu_counts$MOTU ==
rownames(nis_taxa_counts)[i]] <- nis_taxa_counts$Species[i]
}
asv_motu_counts<-asv_motu_counts %>% relocate(Species, .after=MOTU)

write.table(asv_motu_counts,"NIS_MOTU_ASV_counts_COI.txt",sep="\t",row
.names = F)

## Write sequence headers to file to subset the fasta file to the NIS
sequences outside of R

# Write headers with ASV names, as fasta file still has sequences
named by ASV names of representative sequences
nis_headers_asv<-paste0(">",asv_motu_counts$ASV)
```

```
write.table(nis_headers_asv,"nis_headers_asv.txt",sep="\t",row.names
= F,quote=F,col.names = F)

# Write mapping table of ASV to MOTU names to replace the ASV names
in the fasta file with seqkit outside of R

new_headers<-
cbind(asv_motu_counts$ASV,paste(asv_motu_counts$MOTU,asv_motu_counts$A
SV,sep="_"))
write.table(new_headers,"nis_headers_motu.txt",sep="\t",row.names =
F,quote=F,col.names = F)

## Outisde of R, manually classify sequences again to check for
confidence of assignments. In addition, check if occurences at
respective location can actually be considered outside of native
range.

### 18S ###

setwd("~/18S")

## Create final mapping file with all ASVs that have been placed in
one MOTU throughout the pipeline ##

# read LULU and swarm output files and the mapping file of merging
MOTUs with same species assignment

lulu<-read.table("motu_map_lulu_18S.txt",sep="\t",header=T)
swarm<-read.table("output.txt",sep="\t",stringsAsFactors = F)
spec_merge<-
read.table("motu_map_identical_species.txt",sep="\t",header=T)

# Separate swarm table into columns by space
# Remove read count strings from ASV names (lapply and function
necessary to do this for every entry in the table, not just specific
columns)

swarm<-separate_wider_delim(swarm,V1, delim = " ",
names_sep="",too_few = "align_start")
swarm[] <- lapply(swarm, function(y) gsub("_.*","", y))

# Order entries of first column in swarm table based on first column
in lulu table

swarm<-swarm[order(match(swarm$V11,lulu$X)),]
```

```
# Merge swarm and lulu tables

swarm_lulu_map<-cbind(lulu[,4],swarm)
colnames(swarm_lulu_map)[1]<-"MOTU"

# Separate MOTU strings in spec_merge into separate columns

spec_merge<-separate_wider_delim(spec_merge,MOTU, delim = ",",
names_sep="",too_few = "align_start")

# Create ID column in spec_merge (= MOTU other MOTUs have been merged
onto) and transform to long format

spec_merge<-cbind(spec_merge[,c(2,2:ncol(spec_merge))])
colnames(spec_merge)[1]<-"ID"

spec_merge_long <- melt(setDT(spec_merge), id.vars = "ID",
variable.name = "string")
spec_merge_long<-spec_merge_long[,-2]
colnames(spec_merge_long)[2]<-"MOTU"
spec_merge_long<-spec_merge_long[!is.na(spec_merge_long$MOTU),]

## Merge swarm_lulu_map and spec_merge_long

swarm_lulu_spec_merge_map<-as.data.frame(merge(swarm_lulu_map,
spec_merge_long, by = "MOTU", all = TRUE))
swarm_lulu_spec_merge_map<-swarm_lulu_spec_merge_map %>% relocate(ID)

# Where ID is NA, fill in with entry of MOTU column. Then, remove
MOTU column.

swarm_lulu_spec_merge_map$ID<-
ifelse(is.na(swarm_lulu_spec_merge_map$ID),swarm_lulu_spec_merge_map$M
OTU,swarm_lulu_spec_merge_map$ID)
swarm_lulu_spec_merge_map<-swarm_lulu_spec_merge_map[,-2]

# Write all columns except ID column into one string, MOTU names
separated by comma

swarm_lulu_spec_merge_map$MOTU_string<-
apply(swarm_lulu_spec_merge_map[,2:ncol(swarm_lulu_spec_merge_map)],
1,paste, collapse=",")

# Remove ,NA strings (occurred during the previous step when empty
columns were pasted together)
# Keep only ID column and the MOTU_string column
```

```
swarm_lulu_spec_merge_map$MOTU_string<-gsub(",NA.*","",
swarm_lulu_spec_merge_map$MOTU_string)
swarm_lulu_spec_merge_map<-
swarm_lulu_spec_merge_map[,c(1,ncol(swarm_lulu_spec_merge_map))]

# Aggregate rows based on ID column

swarm_lulu_spec_merge_map<-aggregate(.~ ID, data =
swarm_lulu_spec_merge_map, paste, collapse = ",")

# Map ASVs to MOTUs

# motu_asv_mapping.txt stems from initial phyloseq processing
performed previously
mapping<-read.table("motu_asv_mapping.txt",sep="\t",header=T)
mapping<-
mapping[order(match(mapping[,1],swarm_lulu_spec_merge_map[,1])),]

swarm_lulu_spec_merge_map<-
cbind(mapping[,2],swarm_lulu_spec_merge_map)
colnames(swarm_lulu_spec_merge_map)[1:2]<-
c("MOTU","ASV_representative")
swarm_lulu_spec_merge_map<-
separate_wider_delim(swarm_lulu_spec_merge_map,MOTU_string, delim =
",", names_sep="",too_few = "align_start")

# Subset to MOTUs previously identified as NIS

unfiltered_nis_arms<-readRDS("unfiltered_nis_arms.rds")# Saved in
previous script when filtering MOTUs for NIS
nis_asv_motu_map<-swarm_lulu_spec_merge_map %>% filter(MOTU %in%
taxa_names(unfiltered_nis_arms))

# Create table mapping ASVs to those MOTUs and get respective ASV
read counts

nis_asv_motu_map <- melt(setDT(nis_asv_motu_map[,-2]), id.vars =
"MOTU", variable.name = "string")
nis_asv_motu_map<-nis_asv_motu_map[,-2]
colnames(nis_asv_motu_map)[2]<-"ASV"
nis_asv_motu_map<-nis_asv_motu_map[!is.na(nis_asv_motu_map$ASV),]
asv_counts<-
read.table("asv_no_contaminants_18S.txt",sep="\t",header=T,row.names
= 1) # read blank-corrected ASV count table
asv_counts<-asv_counts[rownames(asv_counts) %in%
nis_asv_motu_map$ASV,]
```

```
# Subset ASV counts to the samples present in NIS data set, set ASV
occurrences below 5 to zero and merge samples per ARMS sampling event

unfiltered_nis<-readRDS("unfiltered_nis.rds") # Saved in previous
script when filtering MOTUs for NIS
nis_motu_physeq<-subset_samples(unfiltered_nis,sample_event %in%
sample_names(unfiltered_nis_arms)) # get sample names of NIS data set
with MOTUs showing at least 10 reads per ARMS sampling event
asv_counts<-
asv_counts[colnames(asv_counts)%in%sample_names(nis_motu_physeq)] #
Subset respective ASV count table to these samples
asv_counts[asv_counts < 5] <- 0
asv_counts<-asv_counts[rowSums(asv_counts[])>0,]
asv_counts<-asv_counts[, colSums(asv_counts != 0) > 0]
nis_asv_physeq<-phyloseq(otu_table(asv_counts,taxa_are_rows = TRUE),
sample_data(sample_data(nis_motu_physeq))) # make new phyloseq object
with ASV data set to merge samples per ARMS sampling event
nis_asv_physeq<-merge_samples(nis_asv_physeq,"sample_event") #
ATTENTION: this transposes otu_table
asv_nis_counts<-as.data.frame(t(otu_table(nis_asv_physeq)))

# Generate final table of NIS MOTUs, species names, the ASVs they
contain and the ASV read counts per ARMS sampling event

nis_asv_motu_map<-nis_asv_motu_map %>% filter(ASV %in%
rownames(asv_nis_counts))
nis_asv_motu_map<-
nis_asv_motu_map[order(match(nis_asv_motu_map$ASV,rownames(asv_nis_cou
nts))),]
asv_motu_counts<-cbind(nis_asv_motu_map,asv_nis_counts)
asv_motu_counts<-asv_motu_counts[order(asv_motu_counts$MOTU),]
nis_taxa_counts<-
read.table("nis_taxa_counts_18S.txt",sep="\t",header=T,row.names = 1)
# Read the nis_taxa_count table created in the previous NIS script to
get taxonomy information
# add species names for MOTUs
for (i in 1:nrow(nis_taxa_counts)) {
  asv_motu_counts$Species[asv_motu_counts$MOTU ==
rownames(nis_taxa_counts)[i]] <- nis_taxa_counts$Species[i]
}
asv_motu_counts<-asv_motu_counts %>% relocate(Species, .after=MOTU)

write.table(asv_motu_counts,"NIS_MOTU_ASV_counts_18S.txt",sep="\t",row
.names = F)
```

```
## Write sequence headers to file to subset the fasta file to the NIS
sequences outside of R

# Write headers with ASV names, as fasta file still has sequences
named by ASV names of representative sequences
nis_headers_asv<-paste0(">",asv_motu_counts$ASV)
write.table(nis_headers_asv,"nis_headers_asv.txt",sep="\t",row.names
= F,quote=F,col.names = F)

# Write mapping table of ASV to MOTU names to replace the ASV names
in the fasta file with seqkit outside of R

new_headers<-
cbind(asv_motu_counts$ASV,paste(asv_motu_counts$MOTU,asv_motu_counts$A
SV,sep="_"))
write.table(new_headers,"nis_headers_motu.txt",sep="\t",row.names =
F,quote=F,col.names = F)

## Outisde of R, manually classify sequences again to check for
confidence of assignments. In addition, check if occurences at
respective location can actually be considered outside of native
range.
```

The output of this step are tables (*NIS_MOTU_ASV_counts.txt*; one each for COI and 18S) containing info on MOTUs identified as NIS, their species assignment, the ASVs they contain plus the respective ASV read counts for each ARMS sampling event. We only considered ASV occurrences with at least 5 reads per sample (other occurrences were set to zero and ASVs now left with zero occurrences only were discarded). In addition, the sequence headers of those ASVs were written to file, as well as a mapping file to replace the ASV headers with "MOTUyz_ASVxy". Using the commands below in *Git BASH*, the initial fasta files (*dada2* output) were subset to the ASVs found in NIS MOTUs and the sequence headers were replaced with the aforementioned "MOTUyz_ASVxy" pattern with *SeqKit*:

Command

Make fasta of ASVs found in NIS MOTUs

```
## COI ##

cd ~/COI

# Subset ASV sequences found in NIS MOTUs
grep -w -A 1 -f nis_headers_asv.txt COI_nochim_nosingle_ASVs.fa >
COI_ASV_NIS.fa --no-group-separator

# Replace ASV names with MOTU_ASV names
seqkit replace -p "^(\S+)" --replacement "{kv}" --kv-file
nis_headers_motu.txt COI_ASV_NIS.fa --keep-key > COI_NIS_MOTU.fa

# Sort fasta file to have all ASVs found in the same MOTU as
consecutive sequences

seqkit sort COI_NIS_MOTU.fa -o COI_NIS_MOTU_ASV.fa

# Seqkit creates multiple-line sequences. Change back to single line
awk '/^>/ { print (NR==1 ? "" : RS) $0; next } { printf "%s", $0 }
END { printf RS }' COI_NIS_MOTU_ASV.fa > tmp && mv tmp
COI_NIS_MOTU_ASV_final.fa

## 18S ##

cd ~/18S

# Subset ASV sequences found in NIS MOTUs
grep -w -A 1 -f nis_headers_asv.txt 18S_nochim_nosingle_ASVs.fa >
18S_ASV_NIS.fa --no-group-separator

# Replace ASV names with MOTU_ASV names
seqkit replace -p "^(\S+)" --replacement "{kv}" --kv-file
nis_headers_motu.txt 18S_ASV_NIS.fa --keep-key > 18S_NIS_MOTU.fa

# Sort fasta file to have all ASVs found in the same MOTU as
consecutive sequences

seqkit sort 18S_NIS_MOTU.fa -o 18S_NIS_MOTU_ASV.fa

# Seqkit creates multiple-line sequences. Change back to single line
```

```
awk '/^>/ { print (NR==1 ? "" : RS) $0; next } { printf "%s", $0 }
END { printf RS }' 18S_NIS_MOTU_ASV.fa > tmp && mv tmp
18S_NIS_MOTU_ASV_final.fa
```

We re-did the previous steps briefly to check if more NIS would be detected without the minimum threshold of 5 reads per ASV and sample.

Command

Track fate of ASVs throughout the pipeline without ASV minimum read threshold in R

```
library(phyloseq)
library(dplyr)
library(tidyr)
library(data.table)

### COI ###

setwd("~/COI")

## Create final mapping file with all ASVs that have been placed in
one MOTU throughout the pipeline ##

# read LULU and swarm output files and the mapping file of merging
MOTUs with same species assignment

lulu<-read.table("motu_map_lulu_COI.txt",sep="\t",header=T)
swarm<-read.table("output.txt",sep="\t",stringsAsFactors = F)
spec_merge<-
read.table("motu_map_identical_species.txt",sep="\t",header=T)

# Separate swarm table into columns by space
# Remove read count strings from ASV names (lapply and function
necessary to do this for every entry in the table, not just specific
columns)

swarm<-separate_wider_delim(swarm,V1, delim = " ",
names_sep="",too_few = "align_start")
swarm[] <- lapply(swarm, function(y) gsub("_.*","", y))

# Order entries of first column in swarm table based on first column
in lulu table

swarm<-swarm[order(match(swarm$V11,lulu$X)),]

# Merge swarm and lulu tables

swarm_lulu_map<-cbind(lulu[,4],swarm)
colnames(swarm_lulu_map)[1]<-"MOTU"
```

```
# Separate MOTU strings in spec_merge into separate columns

spec_merge<-separate_wider_delim(spec_merge,MOTU, delim = ",",
names_sep="",too_few = "align_start")

# Create ID column in spec_merge (= MOTU other MOTUs have been merged
onto) and transform to long format

spec_merge<-cbind(spec_merge[,c(2,2:ncol(spec_merge))])
colnames(spec_merge)[1]<-"ID"

spec_merge_long <- melt(setDT(spec_merge), id.vars = "ID",
variable.name = "string")
spec_merge_long<-spec_merge_long[,-2]
colnames(spec_merge_long)[2]<-"MOTU"
spec_merge_long<-spec_merge_long[!is.na(spec_merge_long$MOTU),]

## Merge swarm_lulu_map and spec_merge_long

swarm_lulu_spec_merge_map<-as.data.frame(merge(swarm_lulu_map,
spec_merge_long, by = "MOTU", all = TRUE))
swarm_lulu_spec_merge_map<-swarm_lulu_spec_merge_map %>% relocate(ID)

# Where ID is NA, fill in with entry of MOTU column. Then, remove
MOTU column.

swarm_lulu_spec_merge_map$ID<-
ifelse(is.na(swarm_lulu_spec_merge_map$ID),swarm_lulu_spec_merge_map$M
OTU,swarm_lulu_spec_merge_map$ID)
swarm_lulu_spec_merge_map<-swarm_lulu_spec_merge_map[,-2]

# Write all columns except ID column into one string, MOTU names
separated by comma

swarm_lulu_spec_merge_map$MOTU_string<-
apply(swarm_lulu_spec_merge_map[,2:ncol(swarm_lulu_spec_merge_map)],
1,paste, collapse=",")

# Remove ,NA strings (occurred during the previous step when empty
columns were pasted together)
# Keep only ID column and the MOTU_string column

swarm_lulu_spec_merge_map$MOTU_string<-gsub(",NA.*","",
swarm_lulu_spec_merge_map$MOTU_string)
swarm_lulu_spec_merge_map<-
swarm_lulu_spec_merge_map[,c(1,ncol(swarm_lulu_spec_merge_map))]
```

```
# Aggregate rows based on ID column

swarm_lulu_spec_merge_map<-aggregate(.~ ID, data =
swarm_lulu_spec_merge_map, paste, collapse = ",")

# Map ASVs to MOTUs

# motu_asv_mapping.txt stems from initial phyloseq processing
performed previously
mapping<-read.table("motu_asv_mapping.txt",sep="\t",header=T)
mapping<-
mapping[order(match(mapping[,1],swarm_lulu_spec_merge_map[,1])),]

swarm_lulu_spec_merge_map<-
cbind(mapping[,2],swarm_lulu_spec_merge_map)
colnames(swarm_lulu_spec_merge_map)[1:2]<-
c("MOTU","ASV_representative")
swarm_lulu_spec_merge_map<-
separate_wider_delim(swarm_lulu_spec_merge_map,MOTU_string, delim =
",", names_sep="",too_few = "align_start")

# Subset to MOTUs previously identified as NIS

unfiltered_nis_arms<-readRDS("unfiltered_nis_arms.rds")# Saved in
previous script when filtering MOTUs for NIS
nis_asv_motu_map<-swarm_lulu_spec_merge_map %>% filter(MOTU %in%
taxa_names(unfiltered_nis_arms))

# Create table mapping ASVs to those MOTUs and get respective ASV
read counts

nis_asv_motu_map <- melt(setDT(nis_asv_motu_map[,-2]), id.vars =
"MOTU", variable.name = "string")
nis_asv_motu_map<-nis_asv_motu_map[,-2]
colnames(nis_asv_motu_map)[2]<-"ASV"
nis_asv_motu_map<-nis_asv_motu_map[!is.na(nis_asv_motu_map$ASV),]
asv_counts<-
read.table("asv_no_contaminants_COI.txt",sep="\t",header=T,row.names
= 1) # read blank-corrected ASV count table
asv_counts<-asv_counts[rownames(asv_counts) %in%
nis_asv_motu_map$ASV,]

# Subset ASV counts to the samples present in NIS data set and merge
samples per ARMS sampling event
```

```
unfiltered_nis<-readRDS("unfiltered_nis.rds") # Saved in previous
script when filtering MOTUs for NIS
nis_motu_physeq<-subset_samples(unfiltered_nis,sample_event %in%
sample_names(unfiltered_nis_arms)) # get sample names of NIS data set
with MOTUs showing at least 10 reads per ARMS sampling event
asv_counts<-
asv_counts[colnames(asv_counts)%in%sample_names(nis_motu_physeq)] #
Subset respective ASV count table to these samples
nis_asv_physeq<-phyloseq(otu_table(asv_counts,taxa_are_rows = TRUE),
sample_data(sample_data(nis_motu_physeq))) # make new phyloseq object
with ASV data set to merge samples per ARMS sampling event
nis_asv_physeq<-merge_samples(nis_asv_physeq,"sample_event") #
ATTENTION: this transposes otu_table
asv_nis_counts<-as.data.frame(t(otu_table(nis_asv_physeq)))

# Generate final table of NIS MOTUs, species names, the ASVs they
contain and the ASV read counts per ARMS sampling event

nis_asv_motu_map<-nis_asv_motu_map %>% filter(ASV %in%
rownames(asv_nis_counts))
nis_asv_motu_map<-
nis_asv_motu_map[order(match(nis_asv_motu_map$ASV,rownames(asv_nis_cou
nts))),]
asv_motu_counts<-cbind(nis_asv_motu_map,asv_nis_counts)
asv_motu_counts<-asv_motu_counts[order(asv_motu_counts$MOTU),]
nis_taxa_counts<-
read.table("nis_taxa_counts_COI.txt",sep="\t",header=T,row.names = 1)
# Read the nis_taxa_count table created in the previous NIS script to
get taxonomy information
# add species names for MOTUs
for (i in 1:nrow(nis_taxa_counts)) {
  asv_motu_counts$Species[asv_motu_counts$MOTU ==
rownames(nis_taxa_counts)[i]] <- nis_taxa_counts$Species[i]
}
asv_motu_counts<-asv_motu_counts %>% relocate(Species, .after=MOTU)

write.table(asv_motu_counts,"NIS_MOTU_ASV_counts_COI_nomin5.txt",sep="
\t",row.names = F)

## Write sequence headers to file to subset the fasta file to the NIS
sequences outside of R

# Write headers with ASV names, as fasta file still has sequences
named by ASV names of representative sequences
nis_headers_asv<-paste0(">",asv_motu_counts$ASV)
write.table(nis_headers_asv,"nis_headers_asv_nomin5.txt",sep="\t",row.
names = F,quote=F,col.names = F)
```

```
names = F,quote=F,col.names = F)

# Write mapping table of ASV to MOTU names to replace the ASV names
in the fasta file with seqkit outside of R

new_headers<-
cbind(asv_motu_counts$ASV,paste(asv_motu_counts$MOTU,asv_motu_counts$A
SV,sep="_"))
write.table(new_headers,"nis_headers_motu_nomin5.txt",sep="\t",row.nam
es = F,quote=F,col.names = F)

## Outisde of R, manually classify sequences again to check for
confidence of assignments. In addition, check if occurences at
respective location can actually be considered outside of native
range.

### 18S ###

setwd("~/18S")

## Create final mapping file with all ASVs that have been placed in
one MOTU throughout the pipeline ##

# read LULU and swarm output files and the mapping file of merging
MOTUs with same species assignment

lulu<-read.table("motu_map_lulu_18S.txt",sep="\t",header=T)
swarm<-read.table("output.txt",sep="\t",stringsAsFactors = F)
spec_merge<-
read.table("motu_map_identical_species.txt",sep="\t",header=T)

# Separate swarm table into columns by space
# Remove read count strings from ASV names (lapply and function
necessary to do this for every entry in the table, not just specific
columns)

swarm<-separate_wider_delim(swarm,V1, delim = " ",
names_sep="",too_few = "align_start")
swarm[] <- lapply(swarm, function(y) gsub("_.*","", y))

# Order entries of first column in swarm table based on first column
in lulu table

swarm<-swarm[order(match(swarm$V11,lulu$X)),]

# Merge swarm and lulu tables
```

```
swarm_lulu_map<-cbind(lulu[,4],swarm)
colnames(swarm_lulu_map)[1]<-"MOTU"

# Separate MOTU strings in spec_merge into separate columns

spec_merge<-separate_wider_delim(spec_merge,MOTU, delim = ",",
names_sep="",too_few = "align_start")

# Create ID column in spec_merge (= MOTU other MOTUs have been merged
onto) and transform to long format

spec_merge<-cbind(spec_merge[,c(2,2:ncol(spec_merge))])
colnames(spec_merge)[1]<-"ID"

spec_merge_long <- melt(setDT(spec_merge), id.vars = "ID",
variable.name = "string")
spec_merge_long<-spec_merge_long[,-2]
colnames(spec_merge_long)[2]<-"MOTU"
spec_merge_long<-spec_merge_long[!is.na(spec_merge_long$MOTU),]

## Merge swarm_lulu_map and spec_merge_long

swarm_lulu_spec_merge_map<-as.data.frame(merge(swarm_lulu_map,
spec_merge_long, by = "MOTU", all = TRUE))
swarm_lulu_spec_merge_map<-swarm_lulu_spec_merge_map %>% relocate(ID)

# Where ID is NA, fill in with entry of MOTU column. Then, remove
MOTU column.

swarm_lulu_spec_merge_map$ID<-
ifelse(is.na(swarm_lulu_spec_merge_map$ID),swarm_lulu_spec_merge_map$M
OTU,swarm_lulu_spec_merge_map$ID)
swarm_lulu_spec_merge_map<-swarm_lulu_spec_merge_map[,-2]

# Write all columns except ID column into one string, MOTU names
separated by comma

swarm_lulu_spec_merge_map$MOTU_string<-
apply(swarm_lulu_spec_merge_map[,2:ncol(swarm_lulu_spec_merge_map)],
1,paste, collapse=",")

# Remove ,NA strings (occurred during the previous step when empty
columns were pasted together)
# Keep only ID column and the MOTU_string column

swarm_lulu_spec_merge_map$MOTU_string<-gsub(",NA.*","",
```

```
swarm_lulu_spec_merge_map$MOTU_string)
swarm_lulu_spec_merge_map<-
swarm_lulu_spec_merge_map[,c(1,ncol(swarm_lulu_spec_merge_map))]

# Aggregate rows based on ID column

swarm_lulu_spec_merge_map<-aggregate(.~ ID, data =
swarm_lulu_spec_merge_map, paste, collapse = ",")

# Map ASVs to MOTUs

# motu_asv_mapping.txt stems from initial phyloseq processing
performed previously
mapping<-read.table("motu_asv_mapping.txt",sep="\t",header=T)
mapping<-
mapping[order(match(mapping[,1],swarm_lulu_spec_merge_map[,1])),]

swarm_lulu_spec_merge_map<-
cbind(mapping[,2],swarm_lulu_spec_merge_map)
colnames(swarm_lulu_spec_merge_map)[1:2]<-
c("MOTU","ASV_representative")
swarm_lulu_spec_merge_map<-
separate_wider_delim(swarm_lulu_spec_merge_map,MOTU_string, delim =
",", names_sep="",too_few = "align_start")

# Subset to MOTUs previously identified as NIS

unfiltered_nis_arms<-readRDS("unfiltered_nis_arms.rds")# Saved in
previous script when filtering MOTUs for NIS
nis_asv_motu_map<-swarm_lulu_spec_merge_map %>% filter(MOTU %in%
taxa_names(unfiltered_nis_arms))

# Create table mapping ASVs to those MOTUs and get respective ASV
read counts

nis_asv_motu_map <- melt(setDT(nis_asv_motu_map[,-2]), id.vars =
"MOTU", variable.name = "string")
nis_asv_motu_map<-nis_asv_motu_map[,-2]
colnames(nis_asv_motu_map)[2]<-"ASV"
nis_asv_motu_map<-nis_asv_motu_map[!is.na(nis_asv_motu_map$ASV),]
asv_counts<-
read.table("asv_no_contaminants_18S.txt",sep="\t",header=T,row.names
= 1) # read blank-corrected ASV count table
asv_counts<-asv_counts[rownames(asv_counts) %in%
nis_asv_motu_map$ASV,]

# Subset ASV counts to the samples present in NIS data set and merge
```

```
samples per ARMS sampling event

unfiltered_nis<-readRDS("unfiltered_nis.rds") # Saved in previous
script when filtering MOTUs for NIS
nis_motu_physeq<-subset_samples(unfiltered_nis,sample_event %in%
sample_names(unfiltered_nis_arms)) # get sample names of NIS data set
with MOTUs showing at least 10 reads per ARMS sampling event
asv_counts<-
asv_counts[colnames(asv_counts)%in%sample_names(nis_motu_physeq)] #
Subset respective ASV count table to these samples
nis_asv_physeq<-phyloseq(otu_table(asv_counts,taxa_are_rows = TRUE),
sample_data(sample_data(nis_motu_physeq))) # make new phyloseq object
with ASV data set to merge samples per ARMS sampling event
nis_asv_physeq<-merge_samples(nis_asv_physeq,"sample_event") #
ATTENTION: this transposes otu_table
asv_nis_counts<-as.data.frame(t(otu_table(nis_asv_physeq)))

# Generate final table of NIS MOTUs, species names, the ASVs they
contain and the ASV read counts per ARMS sampling event

nis_asv_motu_map<-nis_asv_motu_map %>% filter(ASV %in%
rownames(asv_nis_counts))
nis_asv_motu_map<-
nis_asv_motu_map[order(match(nis_asv_motu_map$ASV,rownames(asv_nis_cou
nts))),]
asv_motu_counts<-cbind(nis_asv_motu_map,asv_nis_counts)
asv_motu_counts<-asv_motu_counts[order(asv_motu_counts$MOTU),]
nis_taxa_counts<-
read.table("nis_taxa_counts_18S.txt",sep="\t",header=T,row.names = 1)
# Read the nis_taxa_count table created in the previous NIS script to
get taxonomy information
# add species names for MOTUs
for (i in 1:nrow(nis_taxa_counts)) {
  asv_motu_counts$Species[asv_motu_counts$MOTU ==
rownames(nis_taxa_counts)[i]] <- nis_taxa_counts$Species[i]
}
asv_motu_counts<-asv_motu_counts %>% relocate(Species, .after=MOTU)

write.table(asv_motu_counts,"NIS_MOTU_ASV_counts_18S_nomin5.txt",sep="
\t",row.names = F)

## Write sequence headers to file to subset the fasta file to the NIS
sequences outside of R

# Write headers with ASV names, as fasta file still has sequences
named by ASV names of representative sequences
```

```
nis_headers_asv<-paste0(">",asv_motu_counts$ASV)
write.table(nis_headers_asv,"nis_headers_asv_nomin5.txt",sep="\t",row.
names = F,quote=F,col.names = F)

# Write mapping table of ASV to MOTU names to replace the ASV names
in the fasta file with seqkit outside of R

new_headers<-
cbind(asv_motu_counts$ASV,paste(asv_motu_counts$MOTU,asv_motu_counts$A
SV,sep="_"))
write.table(new_headers,"nis_headers_motu_nomin5.txt",sep="\t",row.nam
es = F,quote=F,col.names = F)

## Outisde of R, manually classify sequences again to check for
confidence of assignments. In addition, check if occurences at
respective location can actually be considered outside of native
range.
```

We then also generated fasta files again for these two data sets (COI and 18S) as described above to check taxonomy of these sequences (see below). However, these two data sets without any minimum read threshold for NIS ASVs were not further consiered in the analysis. The steps detailed below were performed for the data sets excluding ASV occurrences of less than 5 reads.

All COI sequences were then classified again manually using *BOLD*'s Identification Engine with the ***Public Record Barcode Database*** (https://v3.boldsystems.org/index.php/IDS_OpenIdEngine). This was done in batches, as classification of a maximum of 50 sequences can be performed at a time. Where there was no match in *BOLD* for COI, sequences were manually classified using the *NCBI blastn* suite (https://blast.ncbi.nlm.nih.gov/Blast.cgi?PROGRAM=blastn&PAGE_TYPE=BlastSearch&LINK_LOC=blasthome) with default settings. The sequences of the 18S NIS data set were also manually classified again using the *NCBI blastn* suite. If there was no hit for 18S sequences on *NCBI* with a query cover of 100% and similarity >98%, ASVs were manually queried against *PR2* using its web service (**https://app.pr2-database.org/pr2-database/**, go to *Query* on the top of the page). *BOLD* gives information on the confidence of a species level assignment for each sequence and notes if an assignment has a high confidence, a sequence may belong to a particular set of taxa or what the nearest match is. *NCBI*'s web service states similarity, query cover and E value for assignments. *PR2* states percentage identity, mismatches and gaps of query alignments.

A literature search based on the sources stated in *WRiMS / WoRMS* for each respective taxon and further scientific and public literature was performed to assess if the occurrences at a

given location in the data set may be considered outside of the respective native range.

In *Excel*, the *NIS_MOTU_ASV_counts_COI.txt* and *NIS_MOTU_ASV_counts_18S.txt* files were then curated manually. This curation was done as follows:

COI ASVs or their occurrences at certain locations were removed if...

- ...there was no match in *BOLD* and where for these ASVs the classification with *NCBI's blastn* had no hits with a query cover of 100% and similarity >98% for the MOTU's species assignment
- ...a species level match could not be made according to *BOLD* and one of the likely species assignments given by *BOLD* was potentially a native species
- ...the presence of the respective taxon was unlikely at the given location (e.g. occurrences in the Baltic Sea of species requiring a higher salinity than present there, etc.)
- ...this ASV and all other ASVs of this MOTU only occurred at locations where the respective species is native. This means the curated data set still contained occurrences of MOTUs where they are native, as long as they occurred at least at one location where they are not native
- ...there was no clear indication based on literature search that a taxon is actually considered as a putative NIS at any of the locations of occurrence

18S ASVs or their occurrences at certain locations were removed if...

- ...the highest valued match (based on E value) in *NCBI's blastn* had no hits with a query cover of 100% and similarity >98% for the MOTU's species assignment and there was no top hit for this assignment in *PR2* with similarity >98%
- ...the top hits in *NCBI's blastn* with a query cover of 100% and similarity >98% with the highest E value represented several taxa and based on its geographic occurrence and the scientific literature it could not be determined if the sequence likely belonged to the respective NIS
- ...the presence of the respective taxon was unlikely at the given location (e.g. occurrences in the Baltic Sea of species requiring a higher salinity than present there, etc.)
- ...this ASV and all other ASVs of this MOTU only occurred at locations where the respective species is native. This means the curated data set still contained occurrences of MOTUs where they are native, as long as they occurred at least at one location where they are not native
- ...there was no clear indication based on literature search that a taxon is actually considered as a putative NIS at any of the locations of occurrence

In some cases, the appropriate taxonomic assignment for certain ASVs and/or MOTUs was manually set as follows:

- Some ASVs had multiple likely assignments, but based on literature and known distribution, the correct assignment could be inferred and was decided on.
- Where all ASVs of the same MOTU had multiple likely species assignments, and all of these where putative NIS and belonged to the same genus, we adjusted the taxonomy of this MOTU to the genus level (i.e., genus X sp.).
- Where some ASVs within the same MOTU had a clear species classification belonging to a NIS, while some ASVs in this MOTU had several likely species assignments and it could not be established based on literature what the likely assignment was, the latter were removed to not have a mix of species and genus level assignments within the same MOTU.

The curated data sets were saved as *NIS_MOTU_ASV_counts_COI_curated.txt* and *NIS_MOTU_ASV_counts_18S_curated.txt*, respectively.

These files are processed in *R / RStudio* to aggregate all remaining MOTU/ASVs and occurrences for each MOTU per ARMS samplig event to create a final NIS presence-absence table for each marker gene. These tables of each marker gene are ultimately merged to obtain a final data set of NIS presence-absence per ARMS sampling event. A second table is created linking ARMS sampling events to the respective coordinates of ARMS. The coordinates for each ARMS unit were obtained from the respective metadata file provided on the ARMS-MBON GitHub page (see above) are can be found in the file below (this file contains coordinates for all ARMS units deployed by ARMS-MBON members to date, and therefore also for ARMS units not part of this present data set):

arms_coordinates.txt 2KB

The final steps were run in *R / RStudio*, see below:

Command

Create final NIS presence-absence and ARMS coordinates tables

```
library(plyr)
library(dplyr)
library(tidyr)

# Get the tables manually curated in Excel

nis_taxa_counts_coi<-
read.table("COI/NIS_MOTU_ASV_counts_COI_curated.txt",sep="\t",header=T
,check.names = F)
nis_taxa_counts_coi<-nis_taxa_counts_coi %>% select(-ASV) # remove
ASV column

nis_taxa_counts_18s<-
read.table("18S/NIS_MOTU_ASV_counts_18S_curated.txt",sep="\t",header=T
,check.names = F)
nis_taxa_counts_18s<-nis_taxa_counts_18s %>% select(-ASV) # remove
ASV column

# Aggregate counts based on MOTU

nis_taxa_counts_coi<-aggregate(.~ MOTU + Species, data =
nis_taxa_counts_coi,FUN=sum)
nis_taxa_counts_18s<-aggregate(.~ MOTU + Species, data =
nis_taxa_counts_18s,FUN=sum)

# Transform to presence-absence and write to file

str(nis_taxa_counts_coi) #check if counts are numeric or integer:
they are integer, so choose "is.integer" in the next line
nis_taxa_counts_coi <- nis_taxa_counts_coi %>% mutate_if(is.integer,
~1 * (. > 0))
str(nis_taxa_counts_18s) #check if counts are numeric or integer:
they are integer, so choose "is.integer" in the next line
nis_taxa_counts_18s <- nis_taxa_counts_18s %>% mutate_if(is.integer,
~1 * (. > 0))

write.table(nis_taxa_counts_coi,"COI/COI_unfiltered_NIS_presence_absen
ce_ARMS.txt",sep="\t",row.names=F)
write.table(nis_taxa_counts_18s,"18S/18S_unfiltered_NIS_presence_absen
ce ARMS.txt",sep="\t",row.names=F)
```

```
# Combine both tables
nis_all<-rbind.fill(nis_taxa_counts_coi,nis_taxa_counts_18s)

# Species will be merged further downstream.
# However, there are MOTU assignments where a NIS could only be
clearly determined on the genus level.
# Where multiple of the same Genus sp. classification remain, we keep
them separate. So these need to be renamed as Genus sp. _1/_2/ etc.

nis_all[grepl("sp\\.", nis_all$Species),"Species" ] # check genus
level assignments to see which ones are duplicated

nis_all[nis_all$Species == "Watersipora sp.",] # Watersipora sp. is
duplicated, get the respective rows

# Manually add number to the respective assignments for these MOTUs
nis_all$Species<-ifelse(nis_all$MOTU=="MOTU209","Watersipora
sp._1",nis_all$Species)
nis_all$Species<-ifelse(nis_all$MOTU=="MOTU53","Watersipora
sp._2",nis_all$Species)
nis_all$Species<-ifelse(nis_all$MOTU=="MOTU431","Watersipora
sp._3",nis_all$Species)

# Remove MOTU column
nis_all<-nis_all %>% select(-MOTU)

# Set NAs in occurrences (got introduced during rbind.fill above) to
zero and aggregate based on species name
nis_all[is.na(nis_all)]<-0
nis_all <- aggregate(. ~ Species, data = nis_all, FUN = sum)

# Set all numeric values above zero to 1
nis_all<-nis_all %>% mutate_if(is.numeric, ~1 * (. > 0))

# Sort columns alphabetically and bring Species column to front
nis_all<-nis_all[,order(colnames(nis_all))]
nis_all<-nis_all %>% relocate(Species)

# Write to file
write.table(nis_all,"ARMS_final_NIS_presence_absence.txt",sep="\t",row
.names = F)

## Write table with coordinates of ARMS locations

arms_events<-as.data.frame(colnames(nis_all[,-1]))
```

```
colnames(arms_events)<-"event"
arms_events$event2 <- arms_events$event
arms_events<-
separate_wider_delim(arms_events,event2,delim="_",names_sep="")
colnames(arms_events)[2:4]<-c("Basin","Deployment","Realm")
head(arms_events)#check whether everything looks correct
write.csv(arms_events,file="ARMS_pairs_coordinates.csv")
arms_events<-arms_events %>% relocate(event)
write.table(arms_events,"ARMS_final_NIS_coordinates.txt",row.names =
F)
```

For further downstream processing, the *NIS_MOTU_ASV_counts_COI_curated.txt* and *NIS_MOTU_ASV_counts_18S_curated.txt* files were additionally manually filtered in *Excel*. These files still contain occurrences of species where they are native, as long as they occurred at least at one location where they are not native. Two new files were generated containing *ONLY* NIS occurrences at locations where they could be considered as NIS. All other occurrences were set to zero. These new files are called *NIS_MOTU_ASV_counts_COI_curated_filtered.txt* and *NIS_MOTU_ASV_counts_18S_curated_filtered.txt*, respectively.

Creating data sets suitable for comparison of NIS prevalence

15 The previously created data sets represent the most unfiltered MOTU data sets for each marker gene. To be able to assess the prevalence of NIS a) among deployment sites, and b) among *types* of deployment sites (i.e., marinas/ports/harbours vs. all other types), we created comparable data sets of equal sampling and sequencing effort for each gene. The following steps are performed for each gene's data set in the *R* scripts found below:

1. The starting point are the previously created unfiltered MOTU *phyloseq* objects, which are read into *R*.
2. In August 2023, some material samples were re-sequenced due to initially poor read yield. In most of those cases, two genetic samples for the respective MaterialSampleID are therefore present in the data set. Based on assessment of rarefaction curves, OTU counts and taxonomy profiles for each of these sample pairs, the sample with lower diversity or taxonomic resolution will be removed.
3. As a trial during the initial phase of the ARMS MBON program, some samples were preserved in DESS/DMSO as well as ethanol. Where both replicates remain in the data set, the sample preserved in ethanol will be removed. Where samples have been preserved in ethanol only or where only the enthanol replicate remained in the data set, those will be kept.
4. There are a couple of samples from Roscoff, France, which have been processed as duplicates. The replicate showing higher read counts and MOTU richness based on rarefaction curves will be kept.
5. Samples with less than 10,000 reads remaining are removed.
6. Samples are rarefied to 10,000 reads to achieve even sequencing depth. **This is now the cleaned and rarefied data set which will be saved!**
7. The previous step created a rarefied MOTU data set. However, we also need to rarefy the initial ASV data set (product of blank correction, see above) because we previously checked ASV occurrences within MOTUs to see if a NIS occurrence can actually be considered as such. So the blank-corrected ASV data sets will be subset to the samples remaining in the rarefied MOTU data sets and then rarefied to an equal sequencing depth of 10,000 reads.

8. The curated and filtered NIS count tables (see section above) are read into *R* and subset to the MOTUs and ASVs remaining after the rarefaction procedures. Occurrences of NIS MOTUs which are now left with less than 10 reads per sampling event (because ASVs got removed) are set to zero.
9. *Phyloseq* objects containing just the NIS MOTUs and their curated and filtered occurrences as NIS in the samples remaining after rarefaction are generated.
10. **Statistical tests are performed to assess differences in NIS richness between samples of marina / harbour/ port sites and all remaining sites.**
11. Info on deployment duration in days is added to the phyloseq objects. In Addition, info on whether samples of all three size or both motile fractions remained for each ARMS unit is added. Five NIS MOTU *phyloseq* objects are generated with ARMS units for which the samples of a) all three fractions; b) both motile fractions; c) the sessile fraction; d) the motile 100um fraction; e) the motile 500um fraction is/are remaining.
12. Some ARMS units have been deployed for multiple, consecutive periods at the same spot (i.e., same ARMS ID, different sampling events). This will of course be the norm as the ARMS-MBON project progresses. However, as the samples processed here stem from the initial years of this monitoring network, some sites contain ARMS units which were only deployed once. Hence, to achieve similar sampling effort among sites for downstream analysis, the sampling events will be assessed visually and the data sets subset to keep only the samples of one sampling event per ARMS. For each of the five *phyloseq* objects (see 11.) only one sampling event is kept for each ARMS ID. The following procedure is applied for selecting sampling events: a) keep the sampling event that temporally overlaps with most of the sampling events of other ARMS, b) within a site = Field_Replicate group with multiple field replicates, keep the sampling events that temporally overlap even if that means some ARMS units will be removed as long as two ARMS units remain within this site = Field_Replicate group. The minimum number of two units is enforced because in the next step two ARMS units are randomly selected for each site = Field_Replicate group to achieve similar sampling effort (this number was chosen after assessing the remaining number of ARMS units for all site = Field_Replicate groups and determining that most sites had two or more units remaining).
13. Randomly select two ARMS units for each for site = Field_Replicate group to achieve similar sampling effort. For site = Field_Replicate groups with only one ARMS remaining, this unit is kept. **These five *phyloseq* objects (ARMS with samples of a) all three fractions; b) both motile fractions; c) the sessile fraction; d) the motile 100um fraction; e) the motile 500um fraction remaining) will be saved for each marker gene.**
14. For each gene, an.xslx file is written. It contains two sheets for each of the five phyloseq objects. One sheet giving the number of NIS found at each site = Field_Replicate group and if this group had one or two ARMS replicates in the curated data set had remaining. The second sheet contains the full taxonomy of each NIS and a table showing presence-absence occurrences for each site = Field_Replicate group.

All steps described above are performed in *R / RStudio* with the script below:

Command

Creating comparable COI NIS data set in R

```
library(phyloseq)
library(dplyr)
library(vegan)
library(ggplot2)
library(tidyr)
library(ggpubr)
library(data.table)
library(xlsx)

setwd("~/COI")

# read the unfiltered phyloseqy object

psCOI_cleaned<-readRDS("psCOI_unfiltered_ARMS.rds")

# Some samples were re-sequenced in August 2023 and in most of those
cases two genetic samples for the respective MaterialSampleID are
therefore present in the data set
# Assess rarefaction curves, OTU counts and taxonomy for each of
those sample pairs and remove sample with lower diversity or
taxonomic resolution

sample_data(psCOI_cleaned)
[duplicated(sample_data(psCOI_cleaned)$MaterialSampleID),"MaterialSamp
leID"] # Check which MaterialSampleIDs appear twice

fornace<-
subset_samples(psCOI_cleaned,MaterialSampleID=="ARMS_GulfOfPiran_Forna
ce_20180815_20181118_SF_ETOH")
fornace<-prune_taxa(rowSums(otu_table(fornace))>0,fornace)
rarecurve(t(otu_table(fornace)), step=50, cex=0.5)
fornace<-cbind(tax_table(fornace),otu_table(fornace))
fornace

katza<-
subset_samples(psCOI_cleaned,MaterialSampleID=="ARMS_Eilat_Katza1_2018
1024_20200706_MF500")
katza<-prune_taxa(rowSums(otu_table(katza))>0,katza)
rarecurve(t(otu_table(katza)), step=50, cex=0.5)
katza<-cbind(tax_table(katza),otu_table(katza))
```

```
katza

koster<-
subset_samples(psCOI_cleaned,MaterialSampleID=="ARMS_Koster_VH1_201905
27_20200716_MF100")
koster<-prune_taxa(rowSums(otu_table(koster))>0,koster)
rarecurve(t(otu_table(koster)), step=50, cex=0.5)
koster<-cbind(tax_table(koster),otu_table(koster))
koster

torallaB<-
subset_samples(psCOI_cleaned,MaterialSampleID=="ARMS_Vigo_TorallaB_201
90625_20191014_MF500")
torallaB<-prune_taxa(rowSums(otu_table(torallaB))>0,torallaB)
rarecurve(t(otu_table(torallaB)), step=50, cex=0.5)
torallaB<-cbind(tax_table(torallaB),otu_table(torallaB))
torallaB

torallaC<-
subset_samples(psCOI_cleaned,MaterialSampleID=="ARMS_Vigo_TorallaC_201
90625_20191014_MF500")
torallaC<-prune_taxa(rowSums(otu_table(torallaC))>0,torallaC)
rarecurve(t(otu_table(torallaC)), step=50, cex=0.5)
torallaC<-cbind(tax_table(torallaC),otu_table(torallaC))
torallaC

to_remove_samples<-
c("ERR7127624","ERR7127581","ERR12541475","ERR4018719","ERR4018721")

psCOI_cleaned <- prune_samples(!(sample_names(psCOI_cleaned) %in%
to_remove_samples), psCOI_cleaned)

# Some samples have been preserved in DMSO as well as EtOH initially
as a trial. Where samples have been preserved in both, keep only DMSO
samples.

sample_data(psCOI_cleaned)
[order(sample_data(psCOI_cleaned)$MaterialSampleID),1] # Check were
MaterialSampleID is present twice with two preservatives

to_remove_pres <-
c("ERR9632064","ERR3460469","ERR3460467","ERR7127579","ERR12541472","E
RR4018450","ERR4018615","ERR7125592","ERR7125594","ERR7125596","ERR712
5598","ERR7125634","ERR7125639","ERR7125641","ERR7125643")

psCOI_cleaned <- prune_samples(!(sample_names(psCOI_cleaned) %in%
```

```
to_remove_pres), psCOI_cleaned)

# Two samples from Roscoff, France have been processed as replicates.

# Assess rarefaction curves to see which sample to keep
rarecurve(t(otu_table(subset_samples(psCOI_cleaned,Lab_Replicate!=""))
), step=50, cex=0.5)

# Remove the sample with lower sample size/MOTU abundance
to_remove_rep <-"ERR7125633"
psCOI_cleaned <- prune_samples(!(sample_names(psCOI_cleaned) %in%
to_remove_rep), psCOI_cleaned)

# Remove samples with a read number of < 10,000

psCOI_cleaned <- prune_samples(sample_sums(psCOI_cleaned) > 10000,
psCOI_cleaned)

### Rarefaction procedure ###

# To get equal sequencing depth, rarefy samples of the MOTU data set
to 10,000 reads with default set.seed(1)

psCOI_cleaned_rarefied <- rarefy_even_depth(psCOI_cleaned, rngseed=1,
sample.size=10000, replace=F)
saveRDS(psCOI_cleaned_rarefied,"psCOI_cleaned_rarefied.rds")

## Rarefaction of ASV data set to get equal sequencing depth also for
the sequences which are WITHIN the MOTUs

# Read the ASV count table (result of blank correction)

asv_counts<-
read.table("asv_no_contaminants_COI.txt",sep="\t",header=T)

# Set ASV names as rownames
rownames(asv_counts) <-asv_counts[,1]
asv_counts[,1] <- NULL

# Subset to samples which are present in psCOI_cleaned_rarefied and
remove ASVs that have a read count of zero as a result

asv_counts_select<-asv_counts[,colnames(asv_counts) %in%
sample_names(psCOI_cleaned_rarefied)]
asv_counts_select<-asv_counts_select[rowSums(asv_counts_select[])>0,]

# Rarefy to 10,000 reads per sample and remove ASVs that have a read
```

```
# Rarefy to 10,000 reads per sample and remove ASVs that have a read
count of zero as a result

set.seed(1)
asv_counts_rarefied<-t(rrarefy(t(asv_counts_select),sample=10000))
asv_counts_rarefied<-
asv_counts_rarefied[rowSums(asv_counts_rarefied[])>0,]

## Create NIS data set based on rarefied MOTU and ASV data sets

# read the previously created and manually curated table with MOTU-
ASV NIS counts per sampling event
# Subset to MOTUs, ASVs and sample events still present after
rarefaction

asv_motu_counts<-
read.table("NIS_MOTU_ASV_counts_COI_curated_filtered.txt",sep="\t",hea
der=T,check.names = F)
asv_motu_counts<-asv_motu_counts %>% filter(MOTU %in%
taxa_names(psCOI_cleaned_rarefied))
asv_motu_counts<-asv_motu_counts %>% filter(ASV %in%
rownames(asv_counts_rarefied))
# make sample_event data
variable1 = as.character(get_variable(psCOI_cleaned_rarefied, "ARMS"))
variable2 = as.character(get_variable(psCOI_cleaned_rarefied,
"Deployment"))
variable3 = as.character(get_variable(psCOI_cleaned_rarefied,
"Retrieval"))
sample_data(psCOI_cleaned_rarefied)$sample_event <- paste(variable1,
variable2, variable3,sep="_")
common_cols <- intersect(colnames(asv_motu_counts[,-
(1:3)]),sample_data(psCOI_cleaned_rarefied)$sample_event)
asv_motu_counts <- asv_motu_counts %>%
select(c(MOTU,Species,ASV,all_of(common_cols)))

# Remove MOTUs in case they are only made up of occurrences with less
than 10 reads per sampling event

abundance_check<-asv_motu_counts %>% select(-c(Species,ASV))
abundance_check<-aggregate(.~ MOTU,abundance_check,FUN=sum)
rownames(abundance_check) <-abundance_check[,1]
abundance_check[,1] <- NULL
abundance_check[abundance_check < 10] <- 0
abundance_check<-abundance_check[rowSums(abundance_check[])>0,]
abundance_check<-abundance_check[, colSums(abundance_check != 0) > 0]

# The previously created and curated NIS data set is based on sample
```

```
events
# we will create a NIS phyloseq object based on individual samples,
however occurrences belonging to sample events where NIS MOTUs were
not considered as NIS need to be set to zero

ps_nis <- prune_taxa(taxa_names(psCOI_cleaned_rarefied) %in%
rownames(abundance_check),psCOI_cleaned_rarefied)

# Make a dataframes of NIS MOTU counts in long format to subsequently
add the actual presence-absence info of NIS MOTUs
nis_counts<-as.data.frame(otu_table(ps_nis))
nis_counts$MOTU<-rownames(nis_counts)
nis_counts<-nis_counts %>% pivot_longer(cols=-MOTU)
abundance_check$MOTU<-rownames(abundance_check)
abundance_check<-abundance_check %>% pivot_longer(cols=-MOTU)

for(i in 1:nrow(sample_data(ps_nis))) { # add sample_event info to
nisc_counts
     nis_counts$event[nis_counts$name == rownames(sample_data(ps_nis))
[i]] <- sample_data(ps_nis)$sample_event[i]
} # ignore warning

for(j in 1:nrow(abundance_check)) { # for each MOTU-sample_event
combination, add the curated NIS MOTU count
    nis_counts$count_new[nis_counts$event == abundance_check$name[j]
& nis_counts$MOTU == abundance_check$MOTU[j]] <-
abundance_check$value[j]
} # ignore warning

nis_counts$count_new[is.na(nis_counts$count_new)]<-0 # where a sample
event was not present in the curated NIS data set because it
contained no NIS, the count_new entry will be NA. we set it to zero.
nis_counts$value<-ifelse(nis_counts$count_new>0,nis_counts$value,0) #
Where curated NIS MOTU count is zero, set count of MOTU in the
respective samples to zero

# There will be cases where the curated NIS MOTU count for a sample
event is not zero (i.e., in the abundance_check object), but is zero
in the rarefied phyloseq object (i.e., in ps_nis/nis_counts).
# This is the case when all counts of a NIS MOTU stem from a sample
which was removed previously because it contained less than 10,00
reads
# The zero count will therefore be kept.

nis_counts<-nis_counts[,-(4:5)] %>% pivot_wider(names_from =
name,values_from = value)
```

```
nis_counts<-as.data.frame(nis_counts)
rownames(nis_counts)<-nis_counts$MOTU
nis_counts$MOTU <- NULL

# Replace the otu_table of the NIS phyloseq object with the table we
just created
otu_table(ps_nis)<-otu_table(nis_counts,taxa_are_rows = TRUE)

## Do statistical tests to check if NIS prevalence is higher at
locations that are marinas / harbours / ports ##

# Check levels of Monitoring_Area
unique(sample_data(ps_nis)$Monitoring_Area)

# Subset samples of marinas, ports, harbours and estimate NIS richness
ports<-subset_samples(ps_nis,Monitoring_Area %in%
c("Marina/Harbour","Marina","Industrial port","Harbour"))
ports_richness<-estimate_richness(ports,measures = "Observed")
ports_richness$type<-"port"

# Subset samples which are NOT from marinas, ports, harbours and
estimate NIS richness
no_ports<-subset_samples(ps_nis,!Monitoring_Area %in%
c("Marina/Harbour","Marina","Industrial port","Harbour"))
no_ports_richness<-estimate_richness(no_ports,measures = "Observed")
no_ports_richness$type<-"no_port"

# Combine tables and do statistical tests (manually copy result from
output field to Excel file)

richness<-rbind(ports_richness,no_ports_richness)
richness$gene<-"COI" # add gene info to make plot with 18S data later
on
write.table(richness,"NIS_richness_COI_port_no_port.txt",sep="\t",row.
names = F)

# For The next command, make sure the plyr package is detached from
your R session in case you used it for something else.
# It will not run properly if the plyr package has been loaded after
dplyr (plyr is not part of this script, but could be in your
environment from another script. Run detach(package:plyr) )
# Could also happen with ggplot2 packagae loaded.
richness %>%  group_by(type) %>% summarize(mean =
mean(Observed),sd=sd(Observed)) # Calculate Means and SD

# Test normality
shapiro.test(richness$Observed) # significant
```

```
shapiro.test(richness$Observed) # significant
shapiro.test(sqrt(richness$Observed)) # significant
shapiro.test(log1p(richness$Observed)) # significant

# Do non-parametric test
kruskal.test(Observed~type,richness)

####

# Add info on deployment duration

# Format respective columns in sample_data as dates
sample_data(ps_nis)<-transform(sample_data(ps_nis),Deployment =
as.Date(as.character(Deployment), "%Y%m%d"))
sample_data(ps_nis)<-transform(sample_data(ps_nis),Retrieval =
as.Date(as.character(Retrieval), "%Y%m%d"))

# Calculate number of days between retrieval and deployment and add a
column
sample_data(ps_nis)$Deployment_Duration<-
sample_data(ps_nis)$Retrieval-sample_data(ps_nis)$Deployment

## Identify ARMS deployments for which all three fractions are
present ##

# Count samples present for each unique ARMS - Deployment - Retrieval
combo

fractions<-sample_data(ps_nis)  %>% count(ARMS, Deployment,Retrieval)

# Make data.frame with column "n" representing number of samples for
ARMS - Deployment - Retrieval combos

fractions<- data.frame(lapply(fractions, function(x) Reduce(c, x)))

# Merge sample_data and fractions table

samples_fractions<-merge(sample_data(ps_nis), fractions,
by=c("ARMS","Deployment","Retrieval"))

# Sort this table based on MaterialSampleID in sample_data of ps_nis

samples_fractions<-
samples_fractions[order(match(samples_fractions$MaterialSampleID,sampl
e_data(ps_nis)$MaterialSampleID)),]

# Add "n" column to sample_data
```

```
sample_data(ps_nis)$Fractions_Present<-samples_fractions$n

## Identify ARMS deployments for which all motile (MF) fractions are
present ##

# Count samples present for each unique ARMS - Deployment - Retrieval
- MF combo

fractions2<-sample_data(ps_nis)  %>% count(ARMS,
Deployment,Retrieval,Fraction)

# Make data.frame with column "mf" representing number of samples for
ARMS - Deployment - Retrieval - MF combos

fractions2<- data.frame(lapply(fractions2, function(x) Reduce(c, x)))
colnames(fractions2)[5]<-"mf"

# Merge sample_data and fractions table

samples_fractions2<-merge(sample_data(ps_nis), fractions2,
by=c("ARMS","Deployment","Retrieval","Fraction"))

# Sort this table based on MaterialSampleID in sample_data of ps_nis

samples_fractions2<-
samples_fractions2[order(match(samples_fractions2$MaterialSampleID,sam
ple_data(ps_nis)$MaterialSampleID)),]

# Add "mf" column to sample_data

sample_data(ps_nis)$Fractions_Present_MF<-samples_fractions2$mf

# Create separate phyloseq objects for ARMS where samples are present
in all size fractions for each sampling event

psCOI_all_frac<-subset_samples(ps_nis,Fractions_Present==3)

# Create separate phyloseq objects for ARMS where samples are present
in all motile (MF) fractions for each sampling event

psCOI_all_mf_frac<-subset_samples(ps_nis,Fractions_Present_MF==2)

# Create separate phyloseq objects for samples separated by each
fraction and for t
```

```
psCOI_mf100<-subset_samples(ps_nis,Fraction_Group=="MF100")

psCOI_mf500<-subset_samples(ps_nis,Fraction_Group=="MF500")

psCOI_sf40<-subset_samples(ps_nis,Fraction_Group=="SF40")

# Checkout deployment duration of ARMS in these five groups

duration_all_frac<-ggplot(sample_data(psCOI_all_frac), aes(x =
Deployment, y = ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("All fractions")
duration_all_frac

duration_all_mf_frac<-ggplot(sample_data(psCOI_all_mf_frac), aes(x =
Deployment, y = ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("All MF fractions")
duration_all_mf_frac

duration_sf40<-ggplot(sample_data(psCOI_sf40), aes(x = Deployment, y
= ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("SF40")
duration_sf40

duration_mf100<-ggplot(sample_data(psCOI_mf100), aes(x = Deployment,
y = ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("MF 100")
duration_mf100

duration_mf500<-ggplot(sample_data(psCOI_mf500), aes(x = Deployment,
y = ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
```

```
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("MF 500")
duration_mf500

psCOI_all_periods<-
ggarrange(duration_all_frac,duration_all_mf_frac,duration_sf40,duratio
n_mf100,duration_mf500,ncol=3,nrow=2)

ggsave(psCOI_all_periods,file="psCOI_all_periods.png",height=10,width=
12)

# Some sites have multiple deployments at different time points. Keep
samples of one period only (the one that overlaps with most other
deployments).

psCOI_all_frac<-subset_samples(psCOI_all_frac,Observatory!="Koster" |
c(Deployment!="2018-04-18" & Deployment!="2020-07-16"))
psCOI_all_frac<-subset_samples(psCOI_all_frac,Observatory!="Limfjord"
| Deployment!="2019-06-18")

psCOI_all_mf_frac<-
subset_samples(psCOI_all_mf_frac,Observatory!="Koster" |
c(Deployment!="2018-04-18" & Deployment!="2020-07-16"))
psCOI_all_mf_frac<-
subset_samples(psCOI_all_mf_frac,Observatory!="Limfjord" |
c(Deployment!="2019-10-29" & Deployment!="2020-11-10"))
psCOI_all_mf_frac<-
subset_samples(psCOI_all_mf_frac,Observatory!="Plymouth" |
Deployment!="2020-07-17")

psCOI_sf40<-subset_samples(psCOI_sf40,Observatory!="Koster" |
c(Deployment!="2018-04-18" & Deployment!="2020-07-16"))
psCOI_sf40<-subset_samples(psCOI_sf40,Observatory!="Limfjord" |
Deployment!="2019-06-18")
psCOI_sf40<-subset_samples(psCOI_sf40,Observatory!="GulfOfPiran" |
Deployment!="2021-02-23")
psCOI_sf40<-subset_samples(psCOI_sf40,Observatory!="Vigo" |
c(Deployment!="2018-06-07" & Deployment!="2019-06-25"))

psCOI_mf100<-subset_samples(psCOI_mf100,Observatory!="Koster" |
c(Deployment!="2018-04-18" & Deployment!="2020-07-16"))
psCOI_mf100<-subset_samples(psCOI_mf100,Observatory!="Limfjord" |
c(Deployment!="2019-10-29" & Deployment!="2020-11-10"))
psCOI_mf100<-subset_samples(psCOI_mf100,Observatory!="Svalbard" |
```

```
Deployment!="2018-07-08")
psCOI_mf100<-subset_samples(psCOI_mf100,Observatory!="Plymouth" |
Deployment!="2019-07-16")

psCOI_mf500<-subset_samples(psCOI_mf500,Observatory!="Koster" |
c(Deployment!="2018-04-18" & Deployment!="2020-07-16"))
psCOI_mf500<-subset_samples(psCOI_mf500,Observatory!="Limfjord" |
c(Deployment!="2019-10-29" & Deployment!="2020-11-10"))
psCOI_mf500<-subset_samples(psCOI_mf500,Observatory!="Roscoff" |
Deployment!="2018-07-09")
psCOI_mf500<-subset_samples(psCOI_mf500,Observatory!="Vigo" |
Deployment!="2019-06-25")
psCOI_mf500<-subset_samples(psCOI_mf500,Observatory!="TZS" |
Deployment!="2020-06-08")
psCOI_mf500<-subset_samples(psCOI_mf500,Observatory!="Plymouth" |
c(Deployment!="2018-07-01" &Deployment!="2020-07-17"))

# Remove MOTUs which have a total abundance of zero after removing
samples during the previous steps

psCOI_all_frac<-
prune_taxa(rowSums(otu_table(psCOI_all_frac))>0,psCOI_all_frac)
psCOI_all_mf_frac<-
prune_taxa(rowSums(otu_table(psCOI_all_mf_frac))>0,psCOI_all_mf_frac)
psCOI_sf40<-prune_taxa(rowSums(otu_table(psCOI_sf40))>0,psCOI_sf40)
psCOI_mf100<-prune_taxa(rowSums(otu_table(psCOI_mf100))>0,psCOI_mf100)
psCOI_mf500<-prune_taxa(rowSums(otu_table(psCOI_mf500))>0,psCOI_mf500)

# Save phyloseq objects to file

saveRDS(psCOI_all_frac,"psCOI_all_frac.rds")
saveRDS(psCOI_all_mf_frac,"psCOI_all_mf_frac.rds")
saveRDS(psCOI_sf40,"psCOI_sf40.rds")
saveRDS(psCOI_mf100,"psCOI_mf100.rds")
saveRDS(psCOI_mf500,"psCOI_mf500.rds")

# Checkout deployment duration of ARMS in these five groups after the
filtering

duration_all_frac<-ggplot(sample_data(psCOI_all_frac), aes(x =
Deployment, y = ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("All fractions")
```

```
duration_sf40

duration_all_mf_frac<-ggplot(sample_data(psCOI_all_mf_frac), aes(x =
Deployment, y = ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("All MF fractions")
duration_all_mf_frac

duration_sf40<-ggplot(sample_data(psCOI_sf40), aes(x = Deployment, y
= ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("SF40")
duration_sf40

duration_mf100<-ggplot(sample_data(psCOI_mf100), aes(x = Deployment,
y = ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("MF 100")
duration_mf100

duration_mf500<-ggplot(sample_data(psCOI_mf500), aes(x = Deployment,
y = ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("MF 500")
duration_mf500

psCOI_all_periods<-
ggarrange(duration_all_frac,duration_all_mf_frac,duration_sf40,duratio
n_mf100,duration_mf500,ncol=3,nrow=2)

ggsave(psCOI_all_periods,file="psCOI_filtered_periods.png",height=10,w
idth=12)

## For each of the five phyloseq objects, randomly sample two ARMS
for each Field Replicate group. Where only one ARMS exists, keep this
```

```
one. ##

set.seed(1) # set.seed for random subsampling

arms_all_frac<- data.frame(sample_data(psCOI_all_frac)) %>%
select(ARMS,Field_Replicate) %>% distinct() %>%
group_by(Field_Replicate) %>% slice_sample(n=2)
psCOI_all_frac<-subset_samples(psCOI_all_frac,ARMS %in%
arms_all_frac$ARMS)

arms_all_mf_frac<- data.frame(sample_data(psCOI_all_mf_frac)) %>%
select(ARMS,Field_Replicate) %>% distinct() %>%
group_by(Field_Replicate) %>% slice_sample(n=2)
psCOI_all_mf_frac<-subset_samples(psCOI_all_mf_frac,ARMS %in%
arms_all_mf_frac$ARMS)

arms_sf40<- data.frame(sample_data(psCOI_sf40)) %>%
select(ARMS,Field_Replicate) %>% distinct() %>%
group_by(Field_Replicate) %>% slice_sample(n=2)
psCOI_sf40<-subset_samples(psCOI_sf40,ARMS %in% arms_sf40$ARMS)

arms_mf100<- data.frame(sample_data(psCOI_mf100)) %>%
select(ARMS,Field_Replicate) %>% distinct() %>%
group_by(Field_Replicate) %>% slice_sample(n=2)
psCOI_mf100<-subset_samples(psCOI_mf100,ARMS %in% arms_mf100$ARMS)

arms_mf500<- data.frame(sample_data(psCOI_mf500)) %>%
select(ARMS,Field_Replicate) %>% distinct() %>%
group_by(Field_Replicate) %>% slice_sample(n=2)
psCOI_mf500<-subset_samples(psCOI_mf500,ARMS %in% arms_mf500$ARMS)

# Merge samples based on Field_Replicate group

psCOI_all_frac<-merge_samples(psCOI_all_frac,"Field_Replicate")
psCOI_all_mf_frac<-merge_samples(psCOI_all_mf_frac,"Field_Replicate")
psCOI_sf40<-merge_samples(psCOI_sf40,"Field_Replicate")
psCOI_mf100<-merge_samples(psCOI_mf100,"Field_Replicate")
psCOI_mf500<-merge_samples(psCOI_mf500,"Field_Replicate")

# Determine NIS richness for each phyloseq object, get coordinates,
taxonomy of NIS and presence-absence at each site and write to file

rich_all_frac<-estimate_richness(psCOI_all_frac,measures="Observed")
rich_all_frac$n_arms<-2 # add number of ARMS present for this
Field_Replicate group
# Adjust number of ARMS per Field_Replicate for some cases
```

```
rich_all_frac$n_arms<-ifelse(rownames(rich_all_frac)=="GDY" |
rownames(rich_all_frac)=="Preemraff" | rownames(rich_all_frac)==
"Gurr" | rownames(rich_all_frac)=="MBA",1,rich_all_frac$n_arms)
for(i in 1:nrow(sample_data(psCOI_all_frac))) { # add coordinates
(have already been averaged for all ARMS of each site during the
merge_samples step)
  rownames(rich_all_frac)<-gsub("\\.","-",rownames(rich_all_frac))
  rich_all_frac$Latitude[rownames(rich_all_frac) ==
rownames(sample_data(psCOI_all_frac))[i]] <-
sample_data(psCOI_all_frac)$Latitude[i]
  rich_all_frac$Longitude[rownames(rich_all_frac) ==
rownames(sample_data(psCOI_all_frac))[i]] <-
sample_data(psCOI_all_frac)$Longitude[i]
}
otu_table_all_frac<-data.frame(t(otu_table(psCOI_all_frac)))
otu_table_all_frac[otu_table_all_frac>0]<-1
tax_table_all_frac<-data.frame(tax_table(psCOI_all_frac))
tax_table_all_frac<-
tax_table_all_frac[order(match(rownames(tax_table_all_frac),rownames(o
tu_table_all_frac))),]
all_frac_nis<-cbind(tax_table_all_frac,otu_table_all_frac)

rich_all_mf_frac<-
estimate_richness(psCOI_all_mf_frac,measures="Observed")
rich_all_mf_frac$n_arms<-2 # add number of ARMS present for this
Field_Replicate group
# Adjust number of ARMS per Field_Replicate for some cases
rich_all_mf_frac$n_arms<-ifelse(rownames(rich_all_mf_frac)=="GDY" |
rownames(rich_all_mf_frac)=="Preemraff" |
rownames(rich_all_mf_frac)== "Gurr" |
rownames(rich_all_mf_frac)=="MBA" |
rownames(rich_all_mf_frac)=="AJJ.AZFP",1,rich_all_mf_frac$n_arms)
for(i in 1:nrow(sample_data(psCOI_all_mf_frac))) { # add coordinates
(have already been averaged for all ARMS of each site during the
merge_samples step)
  rownames(rich_all_mf_frac)<-gsub("\\.","-
",rownames(rich_all_mf_frac))
  rich_all_mf_frac$Latitude[rownames(rich_all_mf_frac) ==
rownames(sample_data(psCOI_all_mf_frac))[i]] <-
sample_data(psCOI_all_mf_frac)$Latitude[i]
  rich_all_mf_frac$Longitude[rownames(rich_all_mf_frac) ==
rownames(sample_data(psCOI_all_mf_frac))[i]] <-
sample_data(psCOI_all_mf_frac)$Longitude[i]
}
otu_table_all_mf_frac<-data.frame(t(otu_table(psCOI_all_mf_frac)))
otu_table_all_mf_frac[otu_table_all_mf_frac>0]<-1
```

```
tax_table_all_mf_frac<-data.frame(tax_table(psCOI_all_mf_frac))
tax_table_all_mf_frac<-
tax_table_all_mf_frac[order(match(rownames(tax_table_all_mf_frac),rown
ames(otu_table_all_mf_frac))),]
all_mf_frac_nis<-cbind(tax_table_all_mf_frac,otu_table_all_mf_frac)

rich_sf40<-estimate_richness(psCOI_sf40,measures="Observed")
rich_sf40$n_arms<-2 # add number of ARMS present for this
Field_Replicate group
# Adjust number of ARMS per Field_Replicate for some cases
rich_sf40$n_arms<-ifelse(rownames(rich_sf40)=="GDY" |
rownames(rich_sf40)=="AZBE" | rownames(rich_sf40)== "Coastbusters" |
rownames(rich_sf40)=="Laeso" |
rownames(rich_sf40)=="Crete",1,rich_sf40$n_arms)
for(i in 1:nrow(sample_data(psCOI_sf40))) { # add coordinates (have
already been averaged for all ARMS of each site during the
merge_samples step)
  rownames(rich_sf40)<-gsub("\\.","-",rownames(rich_sf40))
  rich_sf40$Latitude[rownames(rich_sf40) ==
rownames(sample_data(psCOI_sf40))[i]] <-
sample_data(psCOI_sf40)$Latitude[i]
  rich_sf40$Longitude[rownames(rich_sf40) ==
rownames(sample_data(psCOI_sf40))[i]] <-
sample_data(psCOI_sf40)$Longitude[i]
}
otu_table_sf40<-data.frame(t(otu_table(psCOI_sf40)))
otu_table_sf40[otu_table_sf40>0]<-1
tax_table_sf40<-data.frame(tax_table(psCOI_sf40))
tax_table_sf40<-
tax_table_sf40[order(match(rownames(tax_table_sf40),rownames(otu_table
_sf40))),]
sf40_nis<-cbind(tax_table_sf40,otu_table_sf40)

rich_mf100<-estimate_richness(psCOI_mf100,measures="Observed")
rich_mf100$n_arms<-2 # add number of ARMS present for this
Field_Replicate group
# Adjust number of ARMS per Field_Replicate for some cases
rich_mf100$n_arms<-ifelse(rownames(rich_mf100)=="GDY" |
rownames(rich_mf100)=="Crete" | rownames(rich_mf100)== "Coastbusters"
| rownames(rich_mf100)=="S" |
rownames(rich_mf100)=="Preemraff",1,rich_mf100$n_arms)
for(i in 1:nrow(sample_data(psCOI_mf100))) { # add coordinates (have
already been averaged for all ARMS of each site during the
merge_samples step)
  rownames(rich_mf100)<-gsub("\\.","-",rownames(rich_mf100))
  rich_mf100$Latitude[rownames(rich_mf100) ==
rownames(sample_data(psCOI_mf100))[i]] <-
```

```
sample_data(psCOI_mf100)$Latitude[i]
  rich_mf100$Longitude[rownames(rich_mf100) ==
rownames(sample_data(psCOI_mf100))[i]] <-
sample_data(psCOI_mf100)$Longitude[i]
}
otu_table_mf100<-data.frame(t(otu_table(psCOI_mf100)))
otu_table_mf100[otu_table_mf100>0]<-1
tax_table_mf100<-data.frame(tax_table(psCOI_mf100))
tax_table_mf100<-
tax_table_mf100[order(match(rownames(tax_table_mf100),rownames(otu_tab
le_mf100))),]
mf100_nis<-cbind(tax_table_mf100,otu_table_mf100)

rich_mf500<-estimate_richness(psCOI_mf500,measures="Observed")
rich_mf500$n_arms<-2 # add number of ARMS present for this
Field_Replicate group
# Adjust number of ARMS per Field_Replicate for some cases
rich_mf500$n_arms<-ifelse(rownames(rich_mf500)=="GDY" |
rownames(rich_mf500)=="Crete" | rownames(rich_mf500)== "AZBE" |
rownames(rich_mf500)=="S" | rownames(rich_mf500)=="Preemraff" |
rownames(rich_mf500)=="Gurr"|
rownames(rich_mf500)=="GulfOfPiran",1,rich_mf500$n_arms)
for(i in 1:nrow(sample_data(psCOI_mf500))) { # add coordinates (have
already been averaged for all ARMS of each site during the
merge_samples step)
  rownames(rich_mf500)<-gsub("\\.","-",rownames(rich_mf500))
  rich_mf500$Latitude[rownames(rich_mf500) ==
rownames(sample_data(psCOI_mf500))[i]] <-
sample_data(psCOI_mf500)$Latitude[i]
  rich_mf500$Longitude[rownames(rich_mf500) ==
rownames(sample_data(psCOI_mf500))[i]] <-
sample_data(psCOI_mf500)$Longitude[i]
}
otu_table_mf500<-data.frame(t(otu_table(psCOI_mf500)))
otu_table_mf500[otu_table_mf500>0]<-1
tax_table_mf500<-data.frame(tax_table(psCOI_mf500))
tax_table_mf500<-
tax_table_mf500[order(match(rownames(tax_table_mf500),rownames(otu_tab
le_mf500))),]
mf500_nis<-cbind(tax_table_mf500,otu_table_mf500)

# Write number of NIS abundance per site = Field_Replicate group and
otu_tables of the 5 phyloseq objects as several sheets to one xlsx
file

xlsx::write.xlsx(rich_all_frac, file =
```

```
"NIS_presence_absence_comparison_COI.xlsx",sheetName =
"all_frac_richness", append = FALSE)
xlsx::write.xlsx(all_frac_nis, file =
"NIS_presence_absence_comparison_COI.xlsx",sheetName =
"all_frac_counts_taxa", append = TRUE)
xlsx::write.xlsx(rich_all_mf_frac, file =
"NIS_presence_absence_comparison_COI.xlsx",sheetName =
"all_mf_frac_richness", append = TRUE)
xlsx::write.xlsx(all_mf_frac_nis, file =
"NIS_presence_absence_comparison_COI.xlsx",sheetName =
"all_mf_frac_counts_taxa", append = TRUE)
xlsx::write.xlsx(rich_sf40, file =
"NIS_presence_absence_comparison_COI.xlsx",sheetName =
"sf40_richness", append = TRUE)
xlsx::write.xlsx(sf40_nis, file =
"NIS_presence_absence_comparison_COI.xlsx",sheetName =
"sf40_counts_taxa", append = TRUE)
xlsx::write.xlsx(rich_mf100, file =
"NIS_presence_absence_comparison_COI.xlsx",sheetName =
"mf100_richness", append = TRUE)
xlsx::write.xlsx(mf100_nis, file =
"NIS_presence_absence_comparison_COI.xlsx",sheetName =
"mf100_counts_taxa", append = TRUE)
xlsx::write.xlsx(rich_mf500, file =
"NIS_presence_absence_comparison_COI.xlsx",sheetName =
"mf500_richness", append = TRUE)
xlsx::write.xlsx(mf500_nis, file =
"NIS_presence_absence_comparison_COI.xlsx",sheetName =
"mf500_counts_taxa", append = TRUE)
```

Command

Creating comparable 18S NIS data set in R

```
library(phyloseq)
library(dplyr)
library(vegan)
library(ggplot2)
library(tidyr)
library(ggpubr)
library(data.table)
library(xlsx)

setwd("~/18S")

# read the unfiltered phyloseqy object

ps18S_cleaned<-readRDS("ps18S_unfiltered_ARMS.rds")

# Some samples were re-sequenced in August 2023 and in most of those
cases two genetic samples for the respective MaterialSampleID are
therefore present in the data set
# Assess rarefaction curves, OTU counts and taxonomy for each of
those sample pairs and remove sample with lower diversity or
taxonomic resolution

sample_data(ps18S_cleaned)
[duplicated(sample_data(ps18S_cleaned)$MaterialSampleID),"MaterialSamp
leID"] # Check which MaterialSampleIDs appear twice

koster1<-
subset_samples(ps18S_cleaned,MaterialSampleID=="ARMS_Koster_VH1_201905
27_20200716_MF100")
koster1<-prune_taxa(rowSums(otu_table(koster1))>0,koster1)
rarecurve(t(otu_table(koster1)), step=50, cex=0.5)
koster1<-cbind(tax_table(koster1),otu_table(koster1))
koster1

koster2<-
subset_samples(ps18S_cleaned,MaterialSampleID=="ARMS_Koster_VH1_201905
27_20200716_MF500")
koster2<-prune_taxa(rowSums(otu_table(koster2))>0,koster2)
rarecurve(t(otu_table(koster2)), step=50, cex=0.5)
koster2<-cbind(tax_table(koster2),otu_table(koster2))
```

```
koster2

fornace1<-
subset_samples(ps18S_cleaned,MaterialSampleID=="ARMS_GulfOfPiran_Forna
ce_20180815_20181118_SF_ETOH")
fornace1<-prune_taxa(rowSums(otu_table(fornace1))>0,fornace1)
rarecurve(t(otu_table(fornace1)), step=50, cex=0.5)
fornace1<-cbind(tax_table(fornace1),otu_table(fornace1))
fornace1

fornace2<-
subset_samples(ps18S_cleaned,MaterialSampleID=="ARMS_GulfOfPiran_Forna
ce_20180815_20181118_MF500_ETOH")
fornace2<-prune_taxa(rowSums(otu_table(fornace2))>0,fornace2)
rarecurve(t(otu_table(fornace2)), step=50, cex=0.5)
fornace2<-cbind(tax_table(fornace2),otu_table(fornace2))
fornace2

katza<-
subset_samples(ps18S_cleaned,MaterialSampleID=="ARMS_Eilat_Katza1_2018
1024_20200706_MF500")
katza<-prune_taxa(rowSums(otu_table(katza))>0,katza)
rarecurve(t(otu_table(katza)), step=50, cex=0.5)
katza<-cbind(tax_table(katza),otu_table(katza))
katza

torallaA<-
subset_samples(ps18S_cleaned,MaterialSampleID=="ARMS_Vigo_TorallaA_201
90625_20191014_MF500")
torallaA<-prune_taxa(rowSums(otu_table(torallaA))>0,torallaA)
rarecurve(t(otu_table(torallaA)), step=50, cex=0.5)
torallaA<-cbind(tax_table(torallaA),otu_table(torallaA))
torallaA

torallaB<-
subset_samples(ps18S_cleaned,MaterialSampleID=="ARMS_Vigo_TorallaB_201
90625_20191014_MF500")
torallaB<-prune_taxa(rowSums(otu_table(torallaB))>0,torallaB)
rarecurve(t(otu_table(torallaB)), step=50, cex=0.5)
torallaB<-cbind(tax_table(torallaB),otu_table(torallaB))
torallaB

to_remove_samples<-
c("ERR4914045","ERR4914046","ERR7127577","ERR12541375","ERR7127612","E
RR4018705","ERR4018707")
```

```
ps18S_cleaned <- prune_samples(!(sample_names(ps18S_cleaned) %in%
to_remove_samples), ps18S_cleaned)

# Some samples have been preserved in DMSO as well as EtOH initially
as a trial. Where samples have been preserved in both, keep only DMSO
samples.

sample_data(ps18S_cleaned)
[order(sample_data(ps18S_cleaned)$MaterialSampleID),1] # Check were
MaterialSampleID is present twice with two preservatives

to_remove_pres <-c("ERR9632061","ERR4605087","ERR4605085",
                   "ERR7127575","ERR4605230","ERR12541374",
                   "ERR7125584","ERR7125586","ERR7125588",
                   "ERR4605221","ERR7125590","ERR4605226",
                   "ERR7125621","ERR7125626","ERR7125628",
                   "ERR7125630")

ps18S_cleaned <- prune_samples(!(sample_names(ps18S_cleaned) %in%
to_remove_pres), ps18S_cleaned)

# Two samples from Roscoff, France have been processed as replicates.

# Assess rarefaction curves to see which sample to keep
rarecurve(t(otu_table(subset_samples(ps18S_cleaned,Lab_Replicate!=""))
), step=50, cex=0.5)

# Remove sample with lower sample size/MOTU abundance
to_remove_rep <-"ERR7125620"
ps18S_cleaned <- prune_samples(!(sample_names(ps18S_cleaned) %in%
to_remove_rep), ps18S_cleaned)

# Remove samples with a read number of < 10,000

ps18S_cleaned <- prune_samples(sample_sums(ps18S_cleaned) > 10000,
ps18S_cleaned)

### Rarefaction procedure ###

# To get equal sequencing depth, rarefy samples of the MOTU data set
to 10,000 reads with default set.seed(1)

ps18S_cleaned_rarefied <- rarefy_even_depth(ps18S_cleaned, rngseed=1,
sample.size=10000, replace=F)
saveRDS(ps18S_cleaned_rarefied,"ps18S_cleaned_rarefied.rds")

# Read the ASV count table (result of blank correction)
```

```
# Read the ASV count table (result of blank correction)

asv_counts<-
read.table("asv_no_contaminants_18S.txt",sep="\t",header=T)

# Set ASV names as rownames
rownames(asv_counts) <-asv_counts[,1]
asv_counts[,1] <- NULL

# Subset to samples which are present in ps18S_cleaned_rarefied and
remove ASVs that have a read count of zero as a result

asv_counts_select<-asv_counts[,colnames(asv_counts) %in%
sample_names(ps18S_cleaned_rarefied)]
asv_counts_select<-asv_counts_select[rowSums(asv_counts_select[])>0,]

# Rarefy to 10,000 reads per sample and remove ASVs that have a read
count of zero as a result

set.seed(1)
asv_counts_rarefied<-t(rrarefy(t(asv_counts_select),sample=10000))
asv_counts_rarefied<-
asv_counts_rarefied[rowSums(asv_counts_rarefied[])>0,]

## Create NIS data set based on rarefied MOTU and ASV data sets

# read the previously created and manually curated table with MOTU-
ASV NIS counts per sampling event
# Subset to MOTUs, ASVs and sample events still present after
rarefaction

asv_motu_counts<-
read.table("NIS_MOTU_ASV_counts_18S_curated_filtered.txt",sep="\t",hea
der=T,check.names = F)
asv_motu_counts<-asv_motu_counts %>% filter(MOTU %in%
taxa_names(ps18S_cleaned_rarefied))
asv_motu_counts<-asv_motu_counts %>% filter(ASV %in%
rownames(asv_counts_rarefied))
# make sample_event data
variable1 = as.character(get_variable(ps18S_cleaned_rarefied, "ARMS"))
variable2 = as.character(get_variable(ps18S_cleaned_rarefied,
"Deployment"))
variable3 = as.character(get_variable(ps18S_cleaned_rarefied,
"Retrieval"))
sample_data(ps18S_cleaned_rarefied)$sample_event <- paste(variable1,
variable2, variable3,sep="_")
common cols <- intersect(colnames(asv motu counts[,-
```

```
(1:3)]),sample_data(ps18S_cleaned_rarefied)$sample_event)
asv_motu_counts <- asv_motu_counts %>%
select(c(MOTU,Species,ASV,all_of(common_cols)))

# Remove MOTUs in case they are only made up of occurrences with less
than 10 reads per sampling event

abundance_check<-asv_motu_counts %>% select(-c(Species,ASV))
abundance_check<-aggregate(.~ MOTU,abundance_check,FUN=sum)
rownames(abundance_check) <-abundance_check[,1]
abundance_check[,1] <- NULL
abundance_check[abundance_check < 10] <- 0
abundance_check<-abundance_check[rowSums(abundance_check[])>0,]
abundance_check<-abundance_check[, colSums(abundance_check != 0) > 0]

# The previously created and curated NIS data set is based on sample
events
# we will create a NIS phyloseq object based on individual samples,
however occurrences belonging to sample events where NIS MOTUs were
not considered as NIS need to be set to zero

ps_nis <- prune_taxa(taxa_names(ps18S_cleaned_rarefied) %in%
rownames(abundance_check),ps18S_cleaned_rarefied)

# Make a dataframes of NIS MOTU counts in long format to subsequently
add the actual presence-absence info of NIS MOTUs
nis_counts<-as.data.frame(otu_table(ps_nis))
nis_counts$MOTU<-rownames(nis_counts)
nis_counts<-nis_counts %>% pivot_longer(cols=-MOTU)
abundance_check$MOTU<-rownames(abundance_check)
abundance_check<-abundance_check %>% pivot_longer(cols=-MOTU)

for(i in 1:nrow(sample_data(ps_nis))) { # add sample_event info to
nisc_counts
  nis_counts$event[nis_counts$name == rownames(sample_data(ps_nis))
[i]] <- sample_data(ps_nis)$sample_event[i]
} # ignore warning

for(j in 1:nrow(abundance_check)) { # for each MOTU-sample_event
combination, add the curated NIS MOTU count
  nis_counts$count_new[nis_counts$event == abundance_check$name[j] &
nis_counts$MOTU == abundance_check$MOTU[j]] <-
abundance_check$value[j]
} # ignore warning

nis_counts$count_new[is.na(nis_counts$count_new)]<-0 # where a sample
```

```
event was not present in the curated NIS data set because it
contained no NIS, the count_new entry will be NA. we set it to zero.
nis_counts$value<-ifelse(nis_counts$count_new>0,nis_counts$value,0) #
Where curated NIS MOTU count is zero, set count of MOTU in the
respective samples to zero

# There will be cases where the curated NIS MOTU count for a sample
event is not zero (i.e., in the abundance_check object), but is zero
in the rarefied phyloseq object (i.e., in ps_nis/nis_counts).
# This is the case when all counts of a NIS MOTU stem from a sample
which was removed previously because it contained less than 10,00
reads
# The zero count will therefore be kept.

nis_counts<-nis_counts[,-(4:5)] %>% pivot_wider(names_from =
name,values_from = value)
nis_counts<-as.data.frame(nis_counts)
rownames(nis_counts)<-nis_counts$MOTU
nis_counts$MOTU <- NULL

# Replace the otu_table of the NIS phyloseq object with the table we
just created
otu_table(ps_nis)<-otu_table(nis_counts,taxa_are_rows = TRUE)

## Do statistical tests to check if NIS prevalence is higher at
locations that are marinas / harbours / ports ##

# Check levels of Monitoring_Area
unique(sample_data(ps_nis)$Monitoring_Area)

# Subset samples of marinas, ports, harbours and estimate NIS richness
ports<-subset_samples(ps_nis,Monitoring_Area %in%
c("Marina/Harbour","Marina","Industrial port","Harbour"))
ports_richness<-estimate_richness(ports,measures = "Observed")
ports_richness$type<-"port"

# Subset samples which are NOT from marinas, ports, harbours and
estimate NIS richness
no_ports<-subset_samples(ps_nis,!Monitoring_Area %in%
c("Marina/Harbour","Marina","Industrial port","Harbour"))
no_ports_richness<-estimate_richness(no_ports,measures = "Observed")
no_ports_richness$type<-"no_port"

# Combine tables and do statistical tests (manually copy result from
output field to Excel file)

richness<-rbind(ports_richness,no_ports_richness)
```

```
richness<- rbind(ports_richness,no_ports_richness)
richness$gene<-"18S" # add gene info to make plot with COI data later
on
write.table(richness,"NIS_richness_18S_port_no_port.txt",sep="\t",row.
names = F)

# For The next command, make sure the plyr package is detached from
your R session in case you used it for something else.
# It will not run properly if the plyr package has been loaded after
dplyr (plyr is not part of this script, but could be in your
environment from another script. Run detach(package:plyr) )
# Could also happen with ggplot2 packagae loaded.
richness %>%  group_by(type) %>% summarize(mean =
mean(Observed),sd=sd(Observed)) # Calculate Means and SD

# Test normality
shapiro.test(richness$Observed) # significant
shapiro.test(sqrt(richness$Observed)) # significant
shapiro.test(log1p(richness$Observed)) # significant

# Do non-parametric test
kruskal.test(Observed~type,richness)

####

# Add info on deployment duration

# Format respective columns in sample_data as dates
sample_data(ps_nis)<-transform(sample_data(ps_nis),Deployment =
as.Date(as.character(Deployment), "%Y%m%d"))
sample_data(ps_nis)<-transform(sample_data(ps_nis),Retrieval =
as.Date(as.character(Retrieval), "%Y%m%d"))

# Calculate number of days between retrieval and deployment and add a
column
sample_data(ps_nis)$Deployment_Duration<-
sample_data(ps_nis)$Retrieval-sample_data(ps_nis)$Deployment

## Identify ARMS deployments for which all three fractions are
present ##

# Count samples present for each unique ARMS - Deployment - Retrieval
combo

fractions<-sample_data(ps_nis)  %>% count(ARMS, Deployment,Retrieval)

# Make data.frame with column "n" representing number of samples for
```

```
ARMS - Deployment - Retrieval combos

fractions<- data.frame(lapply(fractions, function(x) Reduce(c, x)))

# Merge sample_data and fractions table

samples_fractions<-merge(sample_data(ps_nis), fractions,
by=c("ARMS","Deployment","Retrieval"))

# Sort this table based on MaterialSampleID in sample_data of ps_nis

samples_fractions<-
samples_fractions[order(match(samples_fractions$MaterialSampleID,sampl
e_data(ps_nis)$MaterialSampleID)),]

# Add "n" column to sample_data

sample_data(ps_nis)$Fractions_Present<-samples_fractions$n

## Identify ARMS deployments for which all motile (MF) fractions are
present ##

# Count samples present for each unique ARMS - Deployment - Retrieval
- MF combo

fractions2<-sample_data(ps_nis)  %>% count(ARMS,
Deployment,Retrieval,Fraction)

# Make data.frame with column "mf" representing number of samples for
ARMS - Deployment - Retrieval - MF combos

fractions2<- data.frame(lapply(fractions2, function(x) Reduce(c, x)))
colnames(fractions2)[5]<-"mf"

# Merge sample_data and fractions table

samples_fractions2<-merge(sample_data(ps_nis), fractions2,
by=c("ARMS","Deployment","Retrieval","Fraction"))

# Sort this table based on MaterialSampleID in sample_data of ps_nis

samples_fractions2<-
samples_fractions2[order(match(samples_fractions2$MaterialSampleID,sam
ple_data(ps_nis)$MaterialSampleID)),]

# Add "mf" column to sample_data
```

```
sample_data(ps_nis)$Fractions_Present_MF<-samples_fractions2$mf

# Create separate phyloseq objects for ARMS where samples are present
in all size fractions for each sampling event

ps18S_all_frac<-subset_samples(ps_nis,Fractions_Present==3)

# Create separate phyloseq objects for ARMS where samples are present
in all motile (MF) fractions for each sampling event

ps18S_all_mf_frac<-subset_samples(ps_nis,Fractions_Present_MF==2)

# Create separate phyloseq objects for samples separated by each
fraction and for t

ps18S_mf100<-subset_samples(ps_nis,Fraction_Group=="MF100")

ps18S_mf500<-subset_samples(ps_nis,Fraction_Group=="MF500")

ps18S_sf40<-subset_samples(ps_nis,Fraction_Group=="SF40")

# Checkout deployment duration of ARMS in these five groups

duration_all_frac<-ggplot(sample_data(ps18S_all_frac), aes(x =
Deployment, y = ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("All fractions")
duration_all_frac

duration_all_mf_frac<-ggplot(sample_data(ps18S_all_mf_frac), aes(x =
Deployment, y = ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("All MF fractions")
duration_all_mf_frac

duration_sf40<-ggplot(sample_data(ps18S_sf40), aes(x = Deployment, y
= ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
```

```
  theme(legend.position = "none")+
  ggtitle("SF40")
duration_sf40

duration_mf100<-ggplot(sample_data(ps18S_mf100), aes(x = Deployment,
y = ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("MF 100")
duration_mf100

duration_mf500<-ggplot(sample_data(ps18S_mf500), aes(x = Deployment,
y = ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("MF 500")
duration_mf500

ps18S_all_periods<-
ggarrange(duration_all_frac,duration_all_mf_frac,duration_sf40,duratio
n_mf100,duration_mf500,ncol=3,nrow=2)

ggsave(ps18S_all_periods,file="ps18S_all_periods.png",height=15,width=
12)

# Some sites have multiple deployments at different time points. Keep
samples of one period only (the one that overlaps with most other
deployments).

ps18S_all_frac<-subset_samples(ps18S_all_frac,Observatory!="Koster" |
Deployment!="2018-04-18" )
ps18S_all_frac<-subset_samples(ps18S_all_frac,Observatory!="Limfjord"
|c(Deployment!="2019-10-29" & Deployment!="2019-06-18"))
ps18S_all_frac<-subset_samples(ps18S_all_frac,Observatory!="Plymouth"
| Deployment!="2019-07-16")

ps18S_all_mf_frac<-
subset_samples(ps18S_all_mf_frac,Observatory!="Koster" |
c(Deployment!="2018-04-18" & Deployment!="2019-05-27"))
ps18S_all_mf_frac<-
subset_samples(ps18S_all_mf_frac,Observatory!="Limfjord" |
c(Deployment!="2019-10-29" & Deployment!="2019-06-18"))
```

```
ps18S_all_mf_frac<-
subset_samples(ps18S_all_mf_frac,Observatory!="Plymouth" |
c(Deployment!="2018-07-01" & Deployment!="2019-07-16"))
ps18S_all_mf_frac<-subset_samples(ps18S_all_mf_frac,ARMS!="BasBloS1"
|  Deployment!="2018-07-11")

ps18S_sf40<-subset_samples(ps18S_sf40,Observatory!="Koster" |
c(Deployment!="2018-04-18" & Deployment!="2019-05-27"))
ps18S_sf40<-subset_samples(ps18S_sf40,Observatory!="Limfjord" |
c(Deployment!="2019-10-29" & Deployment!="2019-06-18"))
ps18S_sf40<-subset_samples(ps18S_sf40,Observatory!="GulfOfPiran" |
Deployment!="2021-02-23")
ps18S_sf40<-subset_samples(ps18S_sf40,Observatory!="Vigo" |
c(Deployment!="2018-06-07" & Deployment!="2019-06-25"))
ps18S_sf40<-subset_samples(ps18S_sf40,Observatory!="TZS" |
Deployment!="2018-07-11")
ps18S_sf40<-subset_samples(ps18S_sf40,Observatory!="Svalbard" |
Deployment!="2018-07-08")
ps18S_sf40<-subset_samples(ps18S_sf40,Observatory!="Plymouth" |
c(Deployment!="2018-07-01" & Deployment!="2019-07-16"))
ps18S_sf40<-subset_samples(ps18S_sf40,Observatory!="Getxo" |
Deployment!="2019-06-24")
ps18S_sf40<-subset_samples(ps18S_sf40,Observatory!="Crete" |
Deployment!="2018-09-28")

ps18S_mf100<-subset_samples(ps18S_mf100,Observatory!="Koster" |
c(Deployment!="2018-04-18" & Deployment!="2019-05-27"))
ps18S_mf100<-subset_samples(ps18S_mf100,Observatory!="Limfjord" |
c(Deployment!="2019-10-29" & Deployment!="2019-06-18"))
ps18S_mf100<-subset_samples(ps18S_mf100,Observatory!="GulfOfPiran" |
Deployment!="2021-02-23")
ps18S_mf100<-subset_samples(ps18S_mf100,Observatory!="Svalbard" |
Deployment!="2018-07-08")
ps18S_mf100<-subset_samples(ps18S_mf100,Observatory!="Plymouth" |
c(Deployment!="2018-07-01" & Deployment!="2019-07-16"))
ps18S_mf100<-subset_samples(ps18S_mf100,ARMS!="BasBloS1" |
Deployment!="2018-07-11")

ps18S_mf500<-subset_samples(ps18S_mf500,Observatory!="Koster" |
c(Deployment!="2018-04-18" & Deployment!="2020-07-16"))
ps18S_mf500<-subset_samples(ps18S_mf500,Observatory!="Limfjord" |
c(Deployment!="2019-10-29" & Deployment!="2019-06-18"))
ps18S_mf500<-subset_samples(ps18S_mf500,Observatory!="Roscoff" |
c(Deployment!="2018-07-09" & Deployment!="2018-07-11"))
ps18S_mf500<-subset_samples(ps18S_mf500,Observatory!="Vigo" |
c(Deployment!="2018-06-07" & Deployment!="2019-06-25"))
```

```
ps18S_mf500<-subset_samples(ps18S_mf500,Observatory!="TZS" |
Deployment!="2020-06-08")
ps18S_mf500<-subset_samples(ps18S_mf500,Observatory!="Plymouth" |
c(Deployment!="2018-07-01" & Deployment!="2019-07-16"))
ps18S_mf500<-subset_samples(ps18S_mf500,Observatory!="Crete" |
Deployment!="2018-09-28")
ps18S_mf500<-subset_samples(ps18S_mf500,Observatory!="Svalbard" |
Deployment!="2018-07-06")

# Remove MOTUs which have a total abundance of zero after removing
samples during the previous steps

ps18S_all_frac<-
prune_taxa(rowSums(otu_table(ps18S_all_frac))>0,ps18S_all_frac)
ps18S_all_mf_frac<-
prune_taxa(rowSums(otu_table(ps18S_all_mf_frac))>0,ps18S_all_mf_frac)
ps18S_sf40<-prune_taxa(rowSums(otu_table(ps18S_sf40))>0,ps18S_sf40)
ps18S_mf100<-prune_taxa(rowSums(otu_table(ps18S_mf100))>0,ps18S_mf100)
ps18S_mf500<-prune_taxa(rowSums(otu_table(ps18S_mf500))>0,ps18S_mf500)

# Save phyloseq objects to file

saveRDS(ps18S_all_frac,"ps18S_all_frac.rds")
saveRDS(ps18S_all_mf_frac,"ps18S_all_mf_frac.rds")
saveRDS(ps18S_sf40,"ps18S_sf40.rds")
saveRDS(ps18S_mf100,"ps18S_mf100.rds")
saveRDS(ps18S_mf500,"ps18S_mf500.rds")

# Checkout deployment duration of ARMS in these five groups after the
filtering

duration_all_frac<-ggplot(sample_data(ps18S_all_frac), aes(x =
Deployment, y = ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("All fractions")
duration_sf40

duration_all_mf_frac<-ggplot(sample_data(ps18S_all_mf_frac), aes(x =
Deployment, y = ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("All MF fractions")
```

```
duration_all_mf_frac

duration_sf40<-ggplot(sample_data(ps18S_sf40), aes(x = Deployment, y
= ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("SF40")
duration_sf40

duration_mf100<-ggplot(sample_data(ps18S_mf100), aes(x = Deployment,
y = ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("MF 100")
duration_mf100

duration_mf500<-ggplot(sample_data(ps18S_mf500), aes(x = Deployment,
y = ARMS, colour = Deployment_Duration)) +
  geom_segment(aes(xend = Retrieval, yend = ARMS), colour = "black") +
  geom_point(size = 3) +
  geom_point(aes(x = Retrieval), size = 3) +
  theme(legend.position = "none")+
  ggtitle("MF 500")
duration_mf500

ps18S_all_periods<-
ggarrange(duration_all_frac,duration_all_mf_frac,duration_sf40,duratio
n_mf100,duration_mf500,ncol=3,nrow=2)

ggsave(ps18S_all_periods,file="ps18S_filtered_periods.png",height=15,w
idth=12)

## For each of the five phyloseq objects, randomly sample two ARMS
for each Field_Replicate group. Where only one ARMS exists, keep this
one. ##

set.seed(1) # set.seed for random subsampling

arms_all_frac<- data.frame(sample_data(ps18S_all_frac)) %>%
select(ARMS,Field_Replicate) %>% distinct() %>%
group_by(Field_Replicate) %>% slice_sample(n=2)
ps18S_all_frac<-subset_samples(ps18S_all_frac,ARMS %in%
```

```
arms_all_frac$ARMS)

arms_all_mf_frac<- data.frame(sample_data(ps18S_all_mf_frac)) %>%
select(ARMS,Field_Replicate) %>% distinct() %>%
group_by(Field_Replicate) %>% slice_sample(n=2)
ps18S_all_mf_frac<-subset_samples(ps18S_all_mf_frac,ARMS %in%
arms_all_mf_frac$ARMS)

arms_sf40<- data.frame(sample_data(ps18S_sf40)) %>%
select(ARMS,Field_Replicate) %>% distinct() %>%
group_by(Field_Replicate) %>% slice_sample(n=2)
ps18S_sf40<-subset_samples(ps18S_sf40,ARMS %in% arms_sf40$ARMS)

arms_mf100<- data.frame(sample_data(ps18S_mf100)) %>%
select(ARMS,Field_Replicate) %>% distinct() %>%
group_by(Field_Replicate) %>% slice_sample(n=2)
ps18S_mf100<-subset_samples(ps18S_mf100,ARMS %in% arms_mf100$ARMS)

arms_mf500<- data.frame(sample_data(ps18S_mf500)) %>%
select(ARMS,Field_Replicate) %>% distinct() %>%
group_by(Field_Replicate) %>% slice_sample(n=2)
ps18S_mf500<-subset_samples(ps18S_mf500,ARMS %in% arms_mf500$ARMS)

# Merge samples based on Field_Replicate group

ps18S_all_frac<-merge_samples(ps18S_all_frac,"Field_Replicate")
ps18S_all_mf_frac<-merge_samples(ps18S_all_mf_frac,"Field_Replicate")
ps18S_sf40<-merge_samples(ps18S_sf40,"Field_Replicate")
ps18S_mf100<-merge_samples(ps18S_mf100,"Field_Replicate")
ps18S_mf500<-merge_samples(ps18S_mf500,"Field_Replicate")

# Determine NIS richness for each phyloseq object, get coordinates,
taxonomy of NIS and presence-absence at each site and write to file

rich_all_frac<-estimate_richness(ps18S_all_frac,measures="Observed")
rich_all_frac$n_arms<-2 # add number of ARMS present for this
Field_Replicate group
# Adjust number of ARMS per Field_Replicate for some cases
rich_all_frac$n_arms<-ifelse(rownames(rich_all_frac)=="Varberg" |
rownames(rich_all_frac)== "TZS" |
rownames(rich_all_frac)=="Marstrand" |
rownames(rich_all_frac)=="Laeso"| rownames(rich_all_frac)=="Eilat"|
rownames(rich_all_frac)=="Gbg"|
rownames(rich_all_frac)=="Helsingborg"|
rownames(rich_all_frac)=="GulfOfPiran"|
rownames(rich_all_frac)=="AZBE"|
```

```
rownames(rich_all_frac)=="A05.AZRP",1,rich_all_frac$n_arms)
for(i in 1:nrow(sample_data(ps18S_all_frac))) { # add coordinates
(have already been averaged for all ARMS of each site during the
merge_samples step)
  rownames(rich_all_frac)<-gsub("\\.","-",rownames(rich_all_frac))
  rich_all_frac$Latitude[rownames(rich_all_frac) ==
rownames(sample_data(ps18S_all_frac))[i]] <-
sample_data(ps18S_all_frac)$Latitude[i]
  rich_all_frac$Longitude[rownames(rich_all_frac) ==
rownames(sample_data(ps18S_all_frac))[i]] <-
sample_data(ps18S_all_frac)$Longitude[i]
}
otu_table_all_frac<-data.frame(t(otu_table(ps18S_all_frac)))
otu_table_all_frac[otu_table_all_frac>0]<-1
tax_table_all_frac<-data.frame(tax_table(ps18S_all_frac))
tax_table_all_frac<-
tax_table_all_frac[order(match(rownames(tax_table_all_frac),rownames(o
tu_table_all_frac))),]
all_frac_nis<-cbind(tax_table_all_frac,otu_table_all_frac)

rich_all_mf_frac<-
estimate_richness(ps18S_all_mf_frac,measures="Observed")
rich_all_mf_frac$n_arms<-2 # add number of ARMS present for this
Field_Replicate group
# Adjust number of ARMS per Field_Replicate for some cases
rich_all_mf_frac$n_arms<-ifelse(rownames(rich_all_mf_frac)=="Varberg"
| rownames(rich_all_mf_frac)=="GulfOfPiran"|
rownames(rich_all_mf_frac)=="AZBE"|
rownames(rich_all_mf_frac)=="G",1,rich_all_mf_frac$n_arms)
for(i in 1:nrow(sample_data(ps18S_all_mf_frac))) { # add coordinates
(have already been averaged for all ARMS of each site during the
merge_samples step)
  rownames(rich_all_mf_frac)<-gsub("\\.","-
",rownames(rich_all_mf_frac))
  rich_all_mf_frac$Latitude[rownames(rich_all_mf_frac) ==
rownames(sample_data(ps18S_all_mf_frac))[i]] <-
sample_data(ps18S_all_mf_frac)$Latitude[i]
  rich_all_mf_frac$Longitude[rownames(rich_all_mf_frac) ==
rownames(sample_data(ps18S_all_mf_frac))[i]] <-
sample_data(ps18S_all_mf_frac)$Longitude[i]
}
otu_table_all_mf_frac<-data.frame(t(otu_table(ps18S_all_mf_frac)))
otu_table_all_mf_frac[otu_table_all_mf_frac>0]<-1
tax_table_all_mf_frac<-data.frame(tax_table(ps18S_all_mf_frac))
tax_table_all_mf_frac<-
tax_table_all_mf_frac[order(match(rownames(tax_table_all_mf_frac),rown
ames(otu_table_all_mf_frac))),]
```

```
all_mf_frac_nis<-cbind(tax_table_all_mf_frac,otu_table_all_mf_frac)

rich_sf40<-estimate_richness(ps18S_sf40,measures="Observed")
rich_sf40$n_arms<-2 # add number of ARMS present for this
Field_Replicate group
# Adjust number of ARMS per Field_Replicate for some cases
rich_sf40$n_arms<-ifelse(rownames(rich_sf40)=="Laeso"|
rownames(rich_sf40)=="Eilat"| rownames(rich_sf40)=="Helsingborg" |
rownames(rich_sf40)=="AZBE"| rownames(rich_sf40)=="Coastbusters"|
rownames(rich_sf40)=="Crete",1,rich_sf40$n_arms)
for(i in 1:nrow(sample_data(ps18S_sf40))) { # add coordinates (have
already been averaged for all ARMS of each site during the
merge_samples step)
  rownames(rich_sf40)<-gsub("\\.","-",rownames(rich_sf40))
  rich_sf40$Latitude[rownames(rich_sf40) ==
rownames(sample_data(ps18S_sf40))[i]] <-
sample_data(ps18S_sf40)$Latitude[i]
  rich_sf40$Longitude[rownames(rich_sf40) ==
rownames(sample_data(ps18S_sf40))[i]] <-
sample_data(ps18S_sf40)$Longitude[i]
}
otu_table_sf40<-data.frame(t(otu_table(ps18S_sf40)))
otu_table_sf40[otu_table_sf40>0]<-1
tax_table_sf40<-data.frame(tax_table(ps18S_sf40))
tax_table_sf40<-
tax_table_sf40[order(match(rownames(tax_table_sf40),rownames(otu_table
_sf40))),]
sf40_nis<-cbind(tax_table_sf40,otu_table_sf40)

rich_mf100<-estimate_richness(ps18S_mf100,measures="Observed")
rich_mf100$n_arms<-2 # add number of ARMS present for this
Field_Replicate group
# Adjust number of ARMS per Field_Replicate for some cases
rich_mf100$n_arms<-ifelse(rownames(rich_mf100)=="Getxo" |
rownames(rich_mf100)=="AZBE"| rownames(rich_mf100)=="Coastbusters"|
rownames(rich_mf100)=="GulfOfPiran",1,rich_mf100$n_arms)
for(i in 1:nrow(sample_data(ps18S_mf100))) { # add coordinates (have
already been averaged for all ARMS of each site during the
merge_samples step)
  rownames(rich_mf100)<-gsub("\\.","-",rownames(rich_mf100))
  rich_mf100$Latitude[rownames(rich_mf100) ==
rownames(sample_data(ps18S_mf100))[i]] <-
sample_data(ps18S_mf100)$Latitude[i]
  rich_mf100$Longitude[rownames(rich_mf100) ==
rownames(sample_data(ps18S_mf100))[i]] <-
sample_data(ps18S_mf100)$Longitude[i]
```

```
}
otu_table_mf100<-data.frame(t(otu_table(ps18S_mf100)))
otu_table_mf100[otu_table_mf100>0]<-1
tax_table_mf100<-data.frame(tax_table(ps18S_mf100))
tax_table_mf100<-
tax_table_mf100[order(match(rownames(tax_table_mf100),rownames(otu_tab
le_mf100))),]
mf100_nis<-cbind(tax_table_mf100,otu_table_mf100)

rich_mf500<-estimate_richness(ps18S_mf500,measures="Observed")
rich_mf500$n_arms<-2 # add number of ARMS present for this
Field_Replicate group
# Adjust number of ARMS per Field_Replicate for some cases
rich_mf500$n_arms<-ifelse(rownames(rich_mf500)=="Crete" |
rownames(rich_mf500)== "AZBE" | rownames(rich_mf500)=="varberg" |
rownames(rich_mf500)=="GulfOfPiran",1,rich_mf500$n_arms)
for(i in 1:nrow(sample_data(ps18S_mf500))) { # add coordinates (have
already been averaged for all ARMS of each site during the
merge_samples step)
  rownames(rich_mf500)<-gsub("\\.","-",rownames(rich_mf500))
  rich_mf500$Latitude[rownames(rich_mf500) ==
rownames(sample_data(ps18S_mf500))[i]] <-
sample_data(ps18S_mf500)$Latitude[i]
  rich_mf500$Longitude[rownames(rich_mf500) ==
rownames(sample_data(ps18S_mf500))[i]] <-
sample_data(ps18S_mf500)$Longitude[i]
}
otu_table_mf500<-data.frame(t(otu_table(ps18S_mf500)))
otu_table_mf500[otu_table_mf500>0]<-1
tax_table_mf500<-data.frame(tax_table(ps18S_mf500))
tax_table_mf500<-
tax_table_mf500[order(match(rownames(tax_table_mf500),rownames(otu_tab
le_mf500))),]
mf500_nis<-cbind(tax_table_mf500,otu_table_mf500)

# Write number of NIS abundance per site = Field_Replicate group and
otu_tables of the 5 phyloseq objects as several sheets to one xlsx
file
xlsx::write.xlsx(rich_all_frac, file =
"NIS_presence_absence_comparison_18S.xlsx",sheetName =
"all_frac_richness", append = FALSE)
xlsx::write.xlsx(all_frac_nis, file =
"NIS_presence_absence_comparison_18S.xlsx",sheetName =
"all_frac_counts_taxa", append = TRUE)
xlsx::write.xlsx(rich_all_mf_frac, file =
"NIS_presence_absence_comparison_18S.xlsx",sheetName =
"all_mf_frac_richness", append = TRUE)
```

```
"all_mf_frac_richness", append = TRUE)
xlsx::write.xlsx(all_mf_frac_nis, file =
"NIS_presence_absence_comparison_18S.xlsx",sheetName =
"all_mf_frac_counts_taxa", append = TRUE)
xlsx::write.xlsx(rich_sf40, file =
"NIS_presence_absence_comparison_18S.xlsx",sheetName =
"sf40_richness", append = TRUE)
xlsx::write.xlsx(sf40_nis, file =
"NIS_presence_absence_comparison_18S.xlsx",sheetName =
"sf40_counts_taxa", append = TRUE)
xlsx::write.xlsx(rich_mf100, file =
"NIS_presence_absence_comparison_18S.xlsx",sheetName =
"mf100_richness", append = TRUE)
xlsx::write.xlsx(mf100_nis, file =
"NIS_presence_absence_comparison_18S.xlsx",sheetName =
"mf100_counts_taxa", append = TRUE)
xlsx::write.xlsx(rich_mf500, file =
"NIS_presence_absence_comparison_18S.xlsx",sheetName =
"mf500_richness", append = TRUE)
xlsx::write.xlsx(mf500_nis, file =
"NIS_presence_absence_comparison_18S.xlsx",sheetName =
"mf500_counts_taxa", append = TRUE)
```