# A QUANTILE REGRESSION APPROACH TO DEFINE OPTIMAL ECOLOGICAL NICHE (HABITAT SUITABILITY) OF COCKLE POPULATIONS (CERASTODERMA EDULE)

Amélie Lehuen[a*], Chloé Dancie[b], Florent Grasso[c], Francis Orvain[a]

[a] Biologie des Organismes et Ecosystèmes Aquatiques (BOREA) Université de Caen Normandie UNICAEN, Sorbonne Université, MNHN, UPMC Univ Paris 06, UA, CNRS 8067, IRD, Esplanade de la paix, F-14032 Caen, France

[b] CSLN, Cellule de Suivi du Littoral Normand (CSLN), 53 rue de Prony, 76600 Le Havre, France

[c] Ifremer, DYNECO/DHYSED, F-29280 Plouzané, France.

* Corresponding author: amelie.lehuen@gmail.com

Data from HMS model: 10.12770/8f5ec053-52c8-4120-b031-4e4b6168ff29

Biological data from report: https://www.seine-aval.fr/publication/macrobenthos-en-estuaire-et-baie-de-seine-mabes/

# ANNEX 1 : HMS DATA TREATMENT

Loading the netcdf file, defining matlab variable names
ncchoice is a character array corresponding to on Mar3D netcdf file (one 'Hyd' and one 'Sed' per year). anstr is the year treated in a character array, type is the type of file, either 'Hyd' or 'Sed', wdworknc the work directory

```
ncid = netcdf.open(ncchoice, 'NC_NOWRITE');
ncid_info = ncinfo(ncchoice)
% File dimensions, variables and attributes
[ndims,nvars] = netcdf.inq(ncid);
% Extracting the list of variable names
Varnames = [];
for i = 1:nvars; Varnames{i,1} = netcdf.inqVar(ncid,i-1); end
for i = 1:nvars; Varnames{i,2} = strrep(ncid_info.Variables(i).Attributes ...
    (strcmp({ncid_info.Variables(i).Attributes.Name},'long_name')).Value,'_',' '); end %
for i = 1:nvars; Varnames{i,3} = strrep(ncid_info.Variables(i).Attributes ...
    (strcmp({ncid_info.Variables(i).Attributes.Name},'units')).Value,'_',' '); end %
for i = 1:nvars; [~,fill] = netcdf.inqVarFill(ncid,i-1); Varnames{i,4} = abs(fill); end
for i = 1:nvars; Varnames{i,5} = size(ncid_info.Variables(i).Size,2); end %
for i = 1:nvars; Varnames{i,6} = i; end %
Varnames{ismember(Varnames(:,1),{'SAL','SALINITE'}),3} = 'u.s.i.';
Varnames{ismember(Varnames(:,1),{'TEMP','TEMPERATURE'}),3} = 'degC'; %
if strcmp(type,'Sed')
  Varnames(contains(Varnames(:,1),'MES'),3) = {'kg.m-3'};
  Varnames{strcmp(Varnames(:,1),'DZS'),2} = 'Layer thickness';
  Varnames{strcmp(Varnames(:,1),'EPTOT'),2} = 'Total sediment thickness';
  Varnames{strcmp(Varnames(:,1),'H0'),2} = 'Bathymetry relative to the mean level';
  Varnames{strcmp(Varnames(:,1),'Nbniv'),2} = 'Number of levels';
  Varnames{strcmp(Varnames(:,1),'SALINITE'),2} = 'Salinity';
  Varnames{strcmp(Varnames(:,1),'TEMPERATURE'),2} = 'Temperature';
  Varnames{strcmp(Varnames(:,1),'TENFON'),2} = 'Bed shear stress';
  Varnames{strcmp(Varnames(:,1),'surf'),2} = 'Mesh size';
end
timeorig =
datetime(ncid_info.Variables(find(strcmp('time',Varnames(:,1)))).Attributes(8).Value);
timeCh = 0; timelong = ncid_info.Variables(find(strcmp('time',Varnames(:,1)))).Size;
if strcmp(type,'Sed'); min_lev = 0; max_lev = 3;
else; min_lev = 0; max_lev = 2;% ncid_info.Variables(strcmp('level',Varnames)).Size;
end
Varnames_long=Varnames; % Save the global list of variables...
Varnames = sortrows(Varnames,5);

Varnames_List = [{'level'},... #1D ,{'time'},{'SIG'}
  {'H0'},{'surf'},... #2D ,{'latitude'},{'longitude'}
  {'U'},{'V'},{'XE'},{'EPTOT'},{'Nbniv'},{'TENFON'},{'ubot'},... #3D
  {'SAL'},{'SALINITE'},{'TEMP'},{'TEMPERATURE'},{'UZ'},{'VZ'},{'DZS'},... #4D
  {'MES_mediumsand'},{'MES_lightsand'},{'MES_mud'}]';
  if strcmp(type,'Sed')
    Varnames_List = [Varnames_List;{'MES_coarsesand'};{'MES_gravel'}];
  end
% Selection of extracted vars
Varnames = Varnames(ismember(Varnames(:,1),Varnames_List),:);
varch = 1:size(Varnames,1);
```

NetCdf data extraction and management of missing data

Extraction is performed by variable type, one for 1 and 2D data, one for 3D and one for 4D. For 3 and 4D, the function extracts the data, saves the var to the wdworknc folder and deletes it from the workspace.
1D Vectors

```
varid = netcdf.inqVarID(ncid,'time');
tmp = netcdf.getVar(ncid,varid,timeCh,timelong); %,timefilter
tmp(abs(tmp) == Varnames_long{varid+1,4}) = NaN;
TIMEdt = seconds(tmp)+timeorig; % Time conversion in seconds to datetime
eval(['Var1D2D.' type '_' 'TIME_' anstr ' = tmp;']);
eval(['Var1D2D.' type '_' 'TIMEdt_' anstr ' = TIMEdt;']);
fprintf([etudenc ' ' anstr ' \n time_start = ' datestr(min(TIMEdt)) ...
  ' \n time_end = ' datestr(max(TIMEdt)) '\n'])
clear TIMEdt;
```

2D Vectors Longitude & Latitude

```
varid = netcdf.inqVarID(ncid,'longitude'); lon_nc = netcdf.getVar(ncid,varid)'; % X
 lon_nc(abs(lon_nc) == Varnames_long{varid+1,4}) = NaN;
varid = netcdf.inqVarID(ncid,'latitude'); lat_nc = netcdf.getVar(ncid,varid)'; % Y
 lat_nc(abs(lat_nc) == Varnames_long{varid+1,4}) = NaN;
lon_nc_tot = lon_nc;
lon_nc = lon_nc(jminid(fz)+1:jmaxid(fz)+1,iminid(fz)+1:imaxid(fz)+1);
eval(['Var1D2D.' type '_' 'lon_nc_' anstr ' = lon_nc;']);
lat_nc_tot = lat_nc;
lat_nc = lat_nc(jminid(fz)+1:jmaxid(fz)+1,iminid(fz)+1:imaxid(fz)+1);
eval(['Var1D2D.' type '_' 'lat_nc_' anstr ' = lat_nc;']);
```

Loop on variables

```
for vc=1:length(varch)
  varid = netcdf.inqVarID(ncid,Varnames{vc,1});
  dim=Varnames{vc,5};
  if dim==1
    tmp = netcdf.getVar(ncid,varid); %
    [~,fill] = netcdf.inqVarFill(ncid,varid); tmp(abs(tmp) == abs(fill)) = NaN;
    eval(['Var1D2D.' type '_' Varnames{vc,1} '_' anstr '= tmp;']);
  elseif dim==2
    tmp = netcdf.getVar(ncid,varid,[iminid(fz) jminid(fz)],[range_i(fz) range_j(fz)]); %
    [~,fill] = netcdf.inqVarFill(ncid,varid); tmp(abs(tmp) == abs(fill)) = NaN;
    if strcmp(Varnames{vc,1},'HO'); tmp=tmp'; end
    eval(['Var1D2D.' type '_' Varnames{vc,1} '_' anstr '= tmp;']);
  elseif dim==3 && (Simple == 3 || Simple == 34)
    tmp = [];
    try
      tmp = netcdf.getVar(ncid,varid,[iminid(fz) jminid(fz) timeCh], ...
        [range_i(fz) range_j(fz) timelong]); %,[1 1 timefilter]
      [~,fill] = netcdf.inqVarFill(ncid,varid); tmp(abs(tmp) == abs(fill)) = NaN;
      tmp = permute(tmp,[2,1,3]);
    catch tmp = NaN(size(lon_nc,1),size(lon_nc,2),timelong);
    end
    eval(['Var3D.' type '_' Varnames{vc,1} '= tmp;']);
    save ([wdworknc, etudenc,'_Var3D_',anstr], 'Var3D');
  elseif dim==4 && (Simple == 4 || Simple == 34)
    tmp_lv = []; tmp = [];
    try
```

```
      for idx = min_lev:max_lev
        tmp = netcdf.getVar(ncid,varid,[iminid(fz) jminid(fz) idx timeCh], ...
          [range_i(fz) range_j(fz) 1 timelong]); %,[1 1 1 timefilter]
        [~,fill] = netcdf.inqVarFill(ncid,varid); tmp(abs(tmp) == abs(fill)) = NaN;
        tmp_lv = cat(3,tmp_lv,tmp);
      end
      tmp_lv = median(tmp_lv,3,'omitnan');
      tmp_lv = permute(tmp_lv, [2 1 4 3]);
    catch tmp_lv = NaN(size(lon_nc,1),size(lon_nc,2),timelong);
    end
    eval([type '_' Varnames{vc,1} '= tmp_lv;']);
    save ([wdworknc, etudenc,'_' type '_' Varnames{vc,1},'_',anstr],[type '_'
Varnames{vc,1}]);
    clear tmp tmp_lv
    eval(['clear ' type '_' Varnames{vc,1}]); %
  end
end % Loop Varnames
clear tmp
netcdf.close(ncid) % Closing the file, VERY IMPORTANT
clear ncid_info
Varnames(:,1)=strcat([type '_'],Varnames(:,1));
if job==1
  Varnames_all=Varnames;
else
  Varnames_all=[Varnames_all; Varnames];
end
```

# ANNEX 2: QUANTILE REGRESSION FUNCTION

Df is the dataframe containing all biologic data and the associated abiotic predictors, biolo the type of biologic response chosen (biomass), sdmod is the kind of model run (which predictors couples), taus the vector of quantiles processed, and typ the character vector describing the quantile regression ("nli")

```
f.gauss2d <- function(x1,x2,A,mu1,sigma1,mu2,sigma2) {
 A*exp(-(((x1-mu1)^2/(2*sigma1^2))+((x2-mu2)^2/(2*sigma2^2))))}

f.rq_nLin <- function(df,biolo,sdmod,taus,typ) {
 typetxt<-paste0("RQ",length(sdmod$id),typ[2])
 sdmname<-sprintf("%s_%g%g%g%s%s",typetxt,sp,sai,
         biolo$id,
         paste0(sdmod$id,collapse = ""),
         ifelse(length(sdmod$id)==1,"0","") )

 yt<- biolo$Var
 yl<- biolo$whole
 xt<-paste(sdmod$Var, collapse=typ[1])
 xl<-paste(sdmod$whole, collapse=paste0(" ",typ[1]," "))
 dfrq <-df[,c(yt,sdmod$Var)] %>%
  drop_na() %>%
  setnames(old=c(yt,sdmod$Var),
      new=c("y",paste0("x",1:length(sdmod$Var))) ) %>%
  mutate(across(everything(),jitter))
 dftmp<-df %>%
  drop_na(all_of(c(yt,sdmod$Var))) %>%
  select(all_of(c("Zone","Tidal_level","Period","Season","Annee","Mois","NINJ")))
 formumod<-as.formula(ifelse(length(sdmod$id)==1,"y~f.gaussf(x1,A,mu,sigma)",
               "y~f.gauss2d(x1,x2,A,mu1,sigma1,mu2,sigma2)"))

 lci <- if(length(sdmod$id)==1){
  list(A=quantile(dfrq$y,FactA_CI), mu=median(dfrq$x1), sigma=sd(dfrq$x1))
 } else {
  list(A=quantile(dfrq$y,FactA_CI), mu1=median(dfrq$x1), sigma1=sd(dfrq$x1),
    mu2=median(dfrq$x2), sigma2=sd(dfrq$x2))}

 # Model grid definition
 xmod<-map( #x1x2mod
  map(as.list(names(dfrq %>% select(-y))),
    ~{ dfrq %>% select(-y) %>%
      summarise(across(everything(),
             list(~min(.x, na.rm = TRUE), ~max(.x, na.rm = TRUE)),
             .names = "{.fn}.{.col}")) %>%
      select(contains(.x)) %>%
```

```
      unlist(., use.names=FALSE) } ) %>%
   setNames(names(dfrq %>% select(-y))) ,
  ~seq(.[1],.[2],length.out=graphfine) ) %>%
  bind_cols()
gridx <- expand.grid(xmod)
if(length(sdmod$Var)==2){
 gridxmat <- mesh(xmod$x1,xmod$x2)
} else{ gridxmat<-NULL }
mod_grid<-list(xmod=xmod,gridx=gridx,gridxmat=gridxmat)

modt<-map(as.list(taus), \(taut) { #taut=taus[[4]]
 tryCatch({
  # nlRQ PARAMETER CONTROL TO AVOID R ABORT WHEN SUMMARY(MODEL) : InitialStepSize=0 (ex=1)
  cc<-nlrq.control(maxiter=100, k=2,
          InitialStepSize = 0,
          big=1e+20, eps=1e-06, beta=0.97)
  modelq0<-rq(y ~ 1, data=dfrq, tau=taut)

  modelq<-nlrq(formumod, data=dfrq,
        start = lci, tau=taut,
        control=cc, method="L-BFGS-B")
  smrq <- summary(modelq) #, se="boot"
  meta <- list(type=typ[1],typetxt=typetxt,
        Sp=espece,Season=saison[sai,2],
        reponse=yt,reponset=yl,
        unit=biolo$Unit,
        predict=xt,predictt=xl,
        predictl=sdmod$whole,
        predfile=paste(sdmod$Var, collapse="_"),
        lci=lci)
  valid <- list(AICc=round(f.AICc_nl(modelq,smrq),1),
         Rone=round((1 - modelq$m$rho/modelq0$rho),5))
  smrq_t <- smrq$coefficients %>%
   as.data.frame %>%
   mutate(Var=rownames(.)) %>%
   relocate(Var) %>%
   remove_rownames(.) %>%
   mutate(tau=taut, taust=sprintf("tau= %s",tau),
      sdmname=sdmname,
      formula=paste0(format(formumod),collapse=""),
      type=typ[1],typetxt=typetxt,
      Sp=espece, Season=saison[sai,2],
      reponse=yt, reponset=yl, unit=biolo$Unit,
      predict=xt, predictt=xl,
      predfile=paste(sdmod$Var, collapse="_"))
  smrq_t<-smrq_t %>% append(valid)
  # Out of limits points calculation
```

```
    rqlim_t <- modelq %>%
     predict %>%
     as.data.frame %>%
     rename_all(~sprintf("%s",taut)) %>%
     mutate(across(everything(), function(x){replace(x, which(x<0), NA)})) %>%
     bind_cols(tibble(dfrq)) %>%
     bind_cols(tibble(dftmp)) %>%
     melt(id.vars=c(names(dfrq),names(dftmp)),
       variable.name = "tau",value.name = "RqLim") %>%
     mutate(sdmname=sdmname,
        taust=sprintf("tau= %s",tau),
        tau=as.numeric(as.character(tau))) %>%
     mutate(status=case_when(y>RqLim ~"over",y<=RqLim ~"under" ))
    # Surface model calculation
    mod_pred_t <- as.data.frame(predict(modelq,newdata=gridx)) %>%
     rename_all(~sprintf("%s",taut)) %>%
     mutate(across(everything(),
            function(x){replace(x, which(x<0), NA)})) %>%
     bind_cols(gridx)
    mod_pred_t <- mod_pred_t %>%
     rename_all(~c("RqMod",names(dfrq %>% select(-y))) ) %>%
     mutate(sdmname=sdmname,
        tau=taut,
        taust=sprintf("tau= %s",tau))

    modl<-list(sdmname=sdmname, meta=meta,
         modelq=modelq, smrq=smrq, smrq_t=smrq_t,
         rqlim_t=rqlim_t, mod_grid=mod_grid, mod_pred_t=mod_pred_t)
   },error = function(e) {print(e)})
  } ) # end map taus

  modt <- modt %>%
   setNames(sprintf("%.3f",taus)) %>%
   transpose %>%
   modify_at(.,c("sdmname","meta","mod_grid"),~.x[[1]]) %>%
   modify_at(.,c("smrq_t","rqlim_t","mod_pred_t"),~.x %>% bind_rows())
  return(modt)
 } # end func

```