Supplementary materials for: Modelling palaeoecological time series using generalized additive models

Gavin L. Simpson

August 15, 2018

1 Introduction

This document is an annotated version of the R code used to fit the GAMs and related analyses to the Small Water and Braya-Sø example data sets.

The following packages are required: *mgcv*, *scam*, *ggplot2*, *cowplot*, and *tidyr*.

```
library("mgcv")
```

```
#> Loading required package: nlme
```

```
#> This is mgcv 1.8-24. For overview type 'help("mgcv-package")'.
```

```
library("scam")
```

```
#> This is scam 1.2-2.
```

```
library("ggplot2")
library("cowplot")
```

```
#>
#> Attaching package: 'cowplot'
```

```
#> The following object is masked from 'package:ggplot2':
#>
#>     ggsave
```

```
library("tidyr")
```

In addition, the *schoenberg* package is required; this in-development package is not on CRAN but can be installed directly from GitHub using functions from the *devtools* package. To install the package, install *devtools* and then use `devtools::install_github()` to install *schoenberg*, as shown below:

```
## schoenberg is not on CRAN, install from github:
install.packages("devtools")
devtools::install_github("gavinsimpson/schoenberg")
```

Assuming that the installation of *schoenberg* completes without error, the package can be loaded as usual

```r
library("schoenberg") # need to change the name of the package
```

The final environment-preparation step is to set the default *ggplot* theme, which the loading of *cowplot* has over-ridden. Here I use the more-minimal black-and-white theme (`theme_bw()`)

```r
## Default ggplot theme
theme_set(theme_bw())
```

2 Load the data sets

The example data sets are also stored on GitHub; https://github.com/gavinsimpson/frontiers-palaeo-additive-modelling. Once downloaded the data are read in and processed a little

```r
## source Small Water data
small <- readRDS("./data/small-water/small-water-isotope-data.rds")
head(small)
```

```
#>   Depth   d13C  TotalC d15N TotalN DryWeight     Year
#> 1   0.2 -27.57  806.49 3.05  64.21       8.2 2007.254
#> 2   0.4 -27.67  949.33 3.01  73.26       7.6 2006.510
#> 3   0.8 -27.63 1305.52 2.93  93.25      11.6 2004.941
#> 4   1.2 -27.62 1136.04 2.33  86.09       9.6 2003.269
#> 5   1.6 -27.48 1028.27 2.09  93.80      10.9 2001.496
#> 6   2.0 -27.39  809.91 2.66  79.98       9.9 1999.626
```

```r
## load braya so data set
braya <- read.table("./data/braya-so/DAndrea.2011.Lake Braya So.txt",
                    skip = 84)
## clean up variable names
names(braya) <- c("Depth", "DepthUpper", "DepthLower", "Year", "YearYoung",
                  "YearOld", "UK37")
## add a variable for the amount of time per sediment sample
braya <- transform(braya, sampleInterval = YearYoung - YearOld)
head(braya)
```

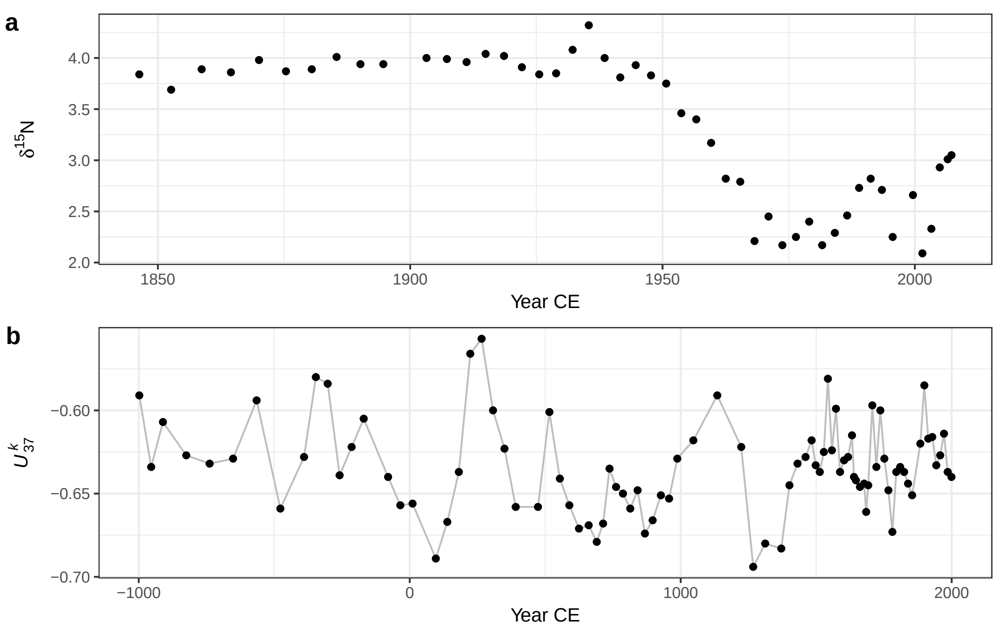
```
#>   Depth DepthUpper DepthLower     Year YearYoung YearOld   UK37
#> 1  0.25        0.0        0.5 1999.125   2006.00 1992.25 -0.640
#> 2  0.75        0.5        1.0 1985.375   1992.25 1978.50 -0.637
#> 3  1.25        1.0        1.5 1971.525   1978.50 1964.55 -0.614
#> 4  1.75        1.5        2.0 1957.575   1964.55 1950.60 -0.627
#> 5  2.25        2.0        2.5 1943.150   1950.60 1935.70 -0.633
#> 6  2.75        2.5        3.0 1928.250   1935.70 1920.80 -0.616
```

```
#>   sampleInterval
#> 1          13.75
#> 2          13.75
#> 3          13.95
#> 4          13.95
#> 5          14.90
#> 6          14.90
```

```r
## plot labels
d15n_label <- expression(delta^{15}*N)
braya_ylabel <- expression(italic(U)[37]^{italic(k)})
```

Plots of the two data sets are prepared using *ggplot2*

```r
## plot Small Water data
small_plt <- ggplot(small, aes(x = Year, y = d15N)) +
    geom_point() +
    labs(y = d15n_label, x = "Year CE")

## Generate a plot of the data
braya_plt <- ggplot(braya, aes(x = Year, y = UK37)) +
    geom_line(colour = "grey") +
    geom_point() +
    labs(y = braya_ylabel, x = "Year CE")

## Recreate plot from manuscript
plot_grid(small_plt, braya_plt, ncol = 1, labels = "auto", align = "hv",
          axis = "lr")
```



3 Fitting GAMs

Instead of following the structure of the paper exactly, the subsequent sections focus on the two example data sets in turn, beginning with Small Water.

3.1 Small Water

A GAM is typically fitted using the `gam()` function from the *mgcv* package. The typical call includes a formula describing the response variable and the linear predictor separated by the ~ (tilde) symbol. The linear predictor part of the model contains the smooth function of the time variable in the data set, and is indicated using the `s()` function. Unless other options are needed, we only need specify where the variables in the formula can be found via the `data` argument, and that we wish to use REML smoothness selection.

Putting this together we have the following function call to fit a simple GAM to the Small Water isotope data

```
m <- gam(d15N ~ s(Year, k = 15), data = small, method = "REML")
```

As discussed in the paper, this model assumes that residuals[1] are independent. To account for residual temporal autocorrelation we can include a continuous-time first-order autoregressive (CAR(1)) process in the model residuals for GAMs fitted to data that are conditionally distributed Gaussian.

The GAM plus CAR(1) process is fitted to the Small Water data set using the `gamm()` function. This fits GAMs as mixed effects models via the *nlme* package, which allows the use of correlation structures in the model residuals via the `correlation` argument. Here, the `corCAR1()` function is used to select the CAR(1) process and we specify the ordering of samples via the `Year` variable in `small`.

```
## fit small water GAM using gamm() with a CAR(1)
mod <- gamm(d15N ~ s(Year, k = 15), data = small,
            correlation = corCAR1(form = ~ Year), method = "REML")
```

The estimated value of ϕ for the CAR(1) can be extracted from the fitted model via the `$lme` component. Here we just extract the correlation structure component.

```
## estimate of phi and confidence interval
smallPhi <- intervals(mod$lme, which = "var-cov")$corStruct
smallPhi
```

```
#>          lower      est.     upper
#> Phi 0.2811107 0.6026967 0.8547543
#> attr(,"label")
#> [1] "Correlation structure:"
```

[1] or equivalently that the observations, conditional upon the model, are independent.

The object returned by `gamm()` includes both the linear mixed model and GAM faces of the model, and as a result we need to access the separate elements (`lme` and `gam` respectively) when proceeding to explore the model fit. The model summary is prepared from the `$gam` component of the fitted model

```r
## summary object
summary(mod$gam)
```

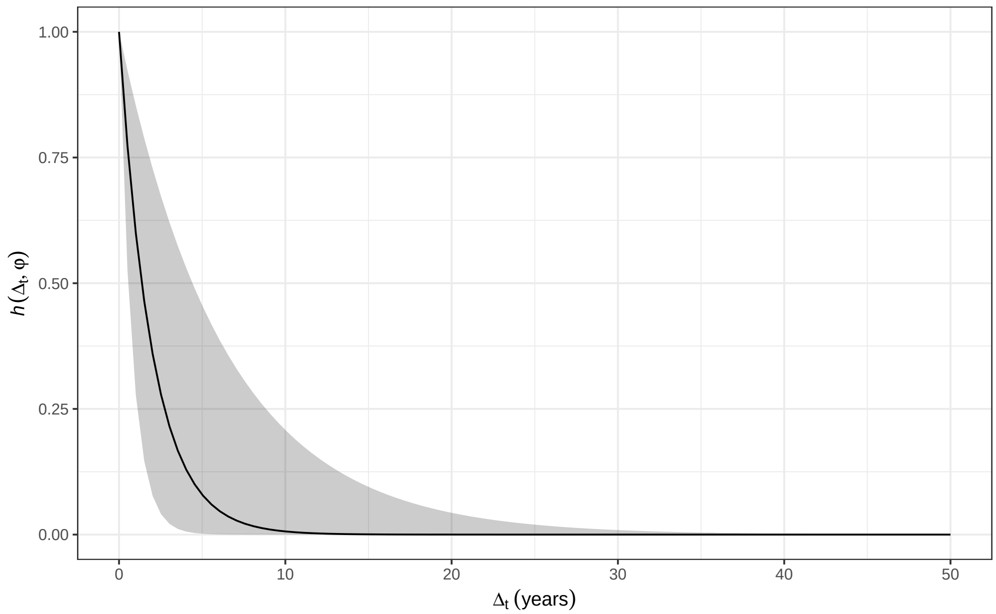
```
#>
#> Family: gaussian
#> Link function: identity
#>
#> Formula:
#> d15N ~ s(Year, k = 15)
#>
#> Parametric coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  3.30909    0.03489   94.84   <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Approximate significance of smooth terms:
#>           edf Ref.df     F p-value
#> s(Year) 7.954  7.954 47.44  <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> R-sq.(adj) =  0.929
#>   Scale est. = 0.037268  n = 48
```

The output shows the estimated complexity of the fitted smooth, expressed in terms of the effective degrees of freedom of the spline. An associated F statistic and test of the null hypothesis of no trend (effect) are also shown. Here the estimated trend provides strong evidence against this null.

The CAR(1) process plotted in Figure 10 of the manuscript was prepared using

```r
## plot CAR(1) process
S <- seq(0, 50, length = 100)
car1 <- setNames(as.data.frame(t(outer(smallPhi, S, FUN = `^`)[1, , ])),
                 c("Lower","Correlation","Upper"))
car1 <- transform(car1, S = S)

car1Plt <- ggplot(car1, aes(x = S, y = Correlation)) +
    geom_ribbon(aes(ymax = Upper, ymin = Lower),
                fill = "black", alpha = 0.2) +
    geom_line() +
```

```
    ylab(expression(italic(h) * (list(Delta[t], varphi)))) +
    xlab(expression(Delta[t] ~ (years)))
car1Plt
```

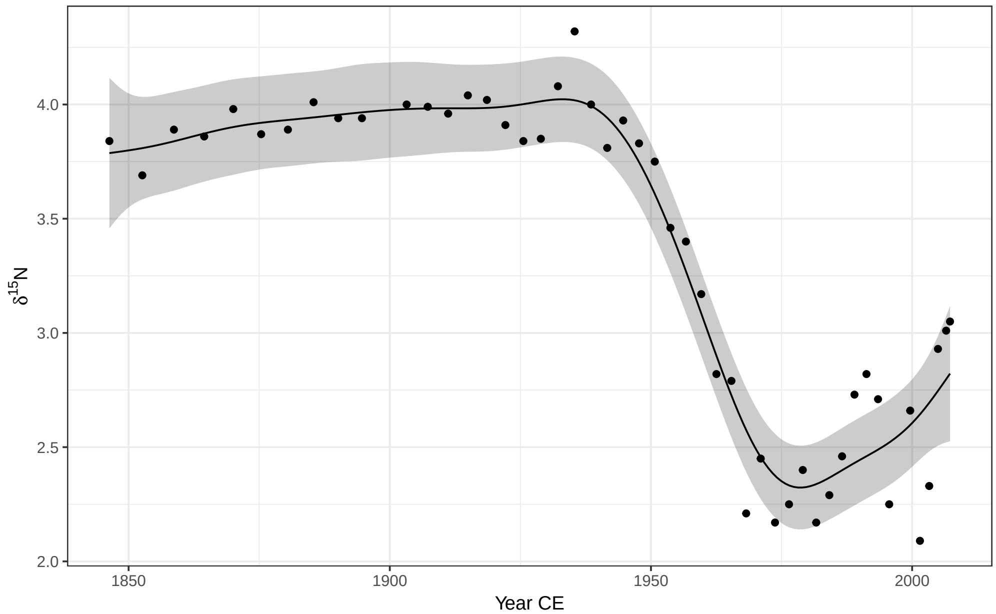


The exponential decline in correlation with increasing separation is evident here; once samples are ~10 years apart, there is little estimated dependence between them.

The next code chunk prepares a plot of the fitted GAMS. The general idea is to predict from the fitted model for a fine grid of points over the range of the time variable. Below I plot the trend for Small Water with an approximate 95% point-wise confidence interval that assumes asymptotic normality

```
N <- 300   # number of points at which to evaluate the smooth

## create new data to predict at; 200 evenly-spaced values over `Year`
newYear <- with(small, data.frame(Year = seq(min(Year), max(Year),
                                             length.out = 200)))
## Predict from the fitted model; note we predict from the $gam part
newYear <- cbind(newYear,
                 data.frame(predict(mod$gam, newYear, se.fit = TRUE)))
## Create the confidence interval
crit.t <- qt(0.975, df = df.residual(mod$gam))
newYear <- transform(newYear,
                     upper = fit + (crit.t * se.fit),
                     lower = fit - (crit.t * se.fit))

## Plot estimated trend
small_fitted <- ggplot(newYear, aes(x = Year, y = fit)) +
    geom_ribbon(aes(ymin = lower, ymax = upper, x = Year), alpha = 0.2,
```

```
                 inherit.aes = FALSE, fill = "black") +
    geom_point(data = small, mapping = aes(x = Year, y = d15N),
                inherit.aes = FALSE) +
    geom_line() +
    labs(y = d15n_label, x = "Year CE")
small_fitted
```



3.2 Braya-Sø

The same model is fitted to the Braya-Sø data set. Note however that in order to even fit the model with both a smooth and the CAR(1) process, I have had to change the default optimiser used to estimate the model parameters, and reduce the basis dimension to a small number. We also fit the model using GCV, which is the defaut, hence no `method` argument

```
## fit the car(1) model --- needs optim as this is not a stable fit!
## also needs k setting lower than default
braya.car1 <- gamm(UK37 ~ s(Year, k = 5), data = braya,
                   correlation = corCAR1(form = ~ Year),
                   method = "REML",
                   control = list(niterEM = 0, optimMethod = "BFGS",
                                  opt = "optim"))

## fit model using GCV
braya.gcv <- gam(UK37 ~ s(Year, k = 30), data = braya)

## estimate of phi and confidence interval
brayaPhi <- intervals(braya.car1$lme)$corStruct
```

```
brayaPhi
```

```
#>              lower      est. upper
#> Phi 1.611049e-22 0.2000156     1
#> attr(,"label")
#> [1] "Correlation structure:"
```

Note the wide confidence interval — effectively 0–1 — on ϕ. If you were to increase the value of `k` to be `k = 10` in the `s(Year)` above, the model will fit but a warning message will be emitted when trying to extract ϕ due to a non-positive definite model covariance matrix, indicating problems with the model.

For Braya-Sø, we repeat the process, but we do so for both models (GAMM + CAR(1) and GCV), and use a critical value from the t distribution to form the confidence interval

```
N <- 300   # number of points at which to evaluate the smooth

## data to predict at
newBraya <- with(braya, data.frame(Year = seq(min(Year), max(Year),
                                              length.out = N)))
## add predictions from GAMM + CAR(1) model
newBraya <- cbind(newBraya,
                  data.frame(predict(braya.car1$gam, newBraya,
                                     se.fit = TRUE)))
crit.t <- qt(0.975, df = df.residual(braya.car1$gam))
newBraya <- transform(newBraya,
                      upper = fit + (crit.t * se.fit),
                      lower = fit - (crit.t * se.fit))

## add GAM GCV results
fit_gcv <- predict(braya.gcv, newdata = newBraya, se.fit = TRUE)
crit.t <- qt(0.975, df.residual(braya.gcv))
newGCV <- data.frame(Year   = newBraya[["Year"]],
                     fit    = fit_gcv$fit,
                     se.fit = fit_gcv$se.fit)
newGCV <- transform(newGCV,
                    upper = fit + (crit.t * se.fit),
                    lower = fit - (crit.t * se.fit))
newBraya <- rbind(newBraya, newGCV)               # bind on GCV results
## Add indicator variable for model
newBraya <- transform(newBraya,
                      Method = rep(c("GAMM (CAR(1))", "GAM (GCV)"),
                                   each = N))

## plot CAR(1) and GCV fits
braya_fitted <- ggplot(braya, aes(y = UK37, x = Year)) +
```

```
    geom_point() +
    geom_ribbon(data = newBraya,
                mapping = aes(x = Year, ymax = upper, ymin = lower,
                              fill = Method),
                alpha = 0.3, inherit.aes = FALSE) +
    geom_line(data = newBraya,
              mapping = aes(y = fit, x = Year, colour = Method)) +
    labs(y = braya_ylabel, x = "Year CE") +
    scale_color_manual(values = c("#5e3c99", "#e66101")) +
    scale_fill_manual(values = c("#5e3c99", "#e66101")) +
    theme(legend.position = "right")
braya_fitted
```

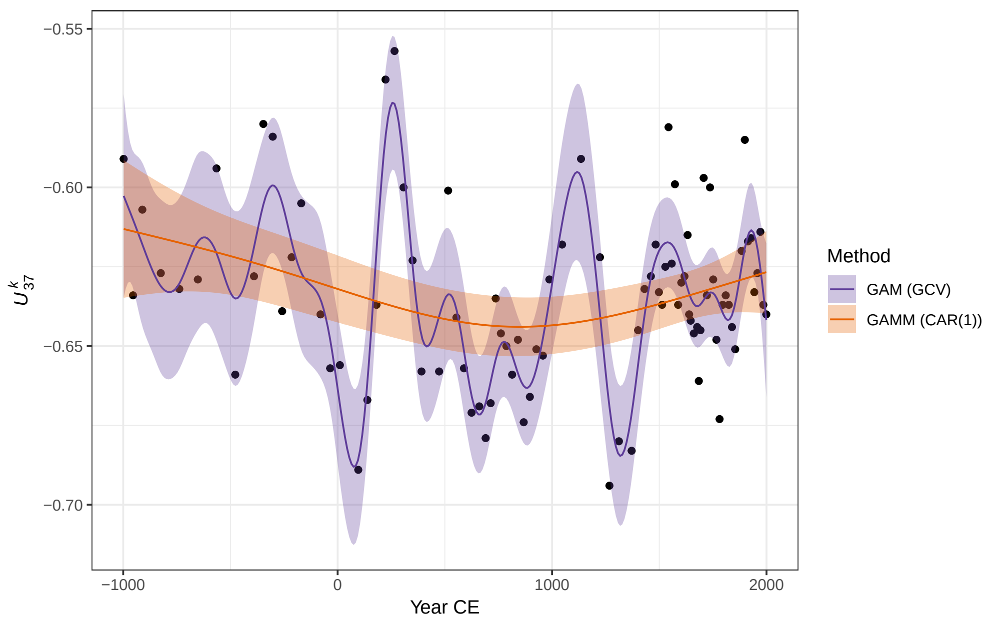


Figure 5 in the manuscript was produced using:

```
plot_grid(small_fitted, braya_fitted, ncol = 1, labels = "auto",
          align = "hv", axis = "lr")
```

3.2.1 Checking if the size of the basis expansion is sufficient

Clearly we are not able to identify both the wiggly trend and the CAR(1) process in a single model. We might proceed by fitting a simple GAM via `gam()` with REML smoothness selection and moderate number of basis functions — here we set `k = 15` for illustration.

```
braya_low_k <- gam(UK37 ~ s(Year, k = 15), data = braya, method = "REML")
```

Before proceeding, we should perform a check to determine if the number of basis functions requested is sufficient to capture the wiggliness in the data. This test is provided by `gam.check()`

```r
gam.check(braya_low_k)
```

```
#>
#> Method: REML   Optimizer: outer newton
#> full convergence after 6 iterations.
#> Gradient range [-4.468978e-08,8.696688e-10]
#> (score -187.2524 & scale 0.000689802).
#> Hessian positive definite, eigenvalue range [0.5525272,43.5153].
#> Model rank =  15 / 15
#>
#> Basis dimension (k) checking results. Low p-value (k-index<1) may
#> indicate that k is too low, especially if edf is close to k'.
#>
#>            k'   edf k-index p-value
#> s(Year) 14.00  2.62    0.56  <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In the output, the column labelled `k'` is the size of the basis used (the number of basis functions) — note that it is 14 and not the requested 15 functions as the constant basis function has been removed as it is confounded with the model constant term, the intercept. The column labelled `edf` indicates the *effective degrees of freedom* for the estimated smooth. The `k-index` column contains the value of a test statistic for a test of sufficient number of basis functions — ideally, we'd like the value of `k-index` to be close to or greater than 1. The quoted *p* value measures the support for the null hypothesis that enough basis functions were available. In practial terms, the low `k-index` and very low *p* value suggest that the requested number of basis functions used to fit the model was too low.

The solution is to increase `k`, say by doubling the original value, and refit the model and perform the basis dimension check using `gam.check()`, repeating this process until `k-index` is close to or greater than 1 and the *p* value is larger than a desired threshold. In my experience, it may not be possible with some data sets to increase `k` large enough to result in a `k-index` > 1 and a *p* value > 0.05, given the available. It is important to remember that the test provided by `gam.check()` is only an heuristic one and shouldn't be seen as infallible. If you find yourself increasing `k` to large values relative to the number of samples in your data set without achieving a `k-index` > 1 or *p* > 0.05, then note the value in the `edf` column as you increase `k`. If the `edf` of the model changes little as you continue to increase `k`, this may be evidence that the test is failing in this particular instance and that the estimated smooth is not dependent on the value of `k` chosen.

3.3 Accounting for heteroscedasticity due to time averaging

To proceed with the Braya-Sø example, we need to increase the basis dimension (`k = 45`), fit using `method = "REML"`. As discussed in the manuscript, we should also account for the non-constant variance, or *heteroscedasticity*, that may arise due to variation in the amount of time (or “lake years”) that each sample represents. All else equal, we expect that samples that average more time have smaller variance than those that average a smaller amount of time. To include information about the expected heteroscedasticity in the GAM we can use observational weights[2]. Here I use the `sampleInterval` variable that I created earlier as the measure of lake years per sample, and to avoid changing the model likelihood, the weights use are the values of `sampleInterval` divided by the mean of `sampleInterval`:

```
braya_reml <- gam(UK37 ~ s(Year, k = 45), data = braya,
                  method = "REML",
                  weights = sampleInterval / mean(sampleInterval))
summary(braya_reml)
```

```
#>
#> Family: gaussian
#> Link function: identity
#>
#> Formula:
#> UK37 ~ s(Year, k = 45)
#>
#> Parametric coefficients:
#>              Estimate Std. Error t value Pr(>|t|)
#> (Intercept) -0.633734   0.001901  -333.3   <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Approximate significance of smooth terms:
#>           edf Ref.df     F p-value
#> s(Year) 28.49  33.99 7.755  <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> R-sq.(adj) =  0.751   Deviance explained = 83.2%
#> -REML = -175.68  Scale est. = 0.00024036  n = 89
```

If we now check if 44 (`k = 45`) basis functions is sufficient

```
gam.check(braya_reml)
```

```
#>
#> Method: REML   Optimizer: outer newton
#> full convergence after 7 iterations.
#> Gradient range [-2.210576e-06,-5.181457e-08]
#> (score -175.684 & scale 0.0002403604).
#> Hessian positive definite, eigenvalue range [2.524305,48.11146].
```

[2]This simple solution is for a Gaussian GAM. For models with other distributions, the exact interpretation of observational weights will vary. For such data, where the conditional mean and variance of the observations are linked, the use of location-scale, or distributional, models may be more apporpriate and informative.

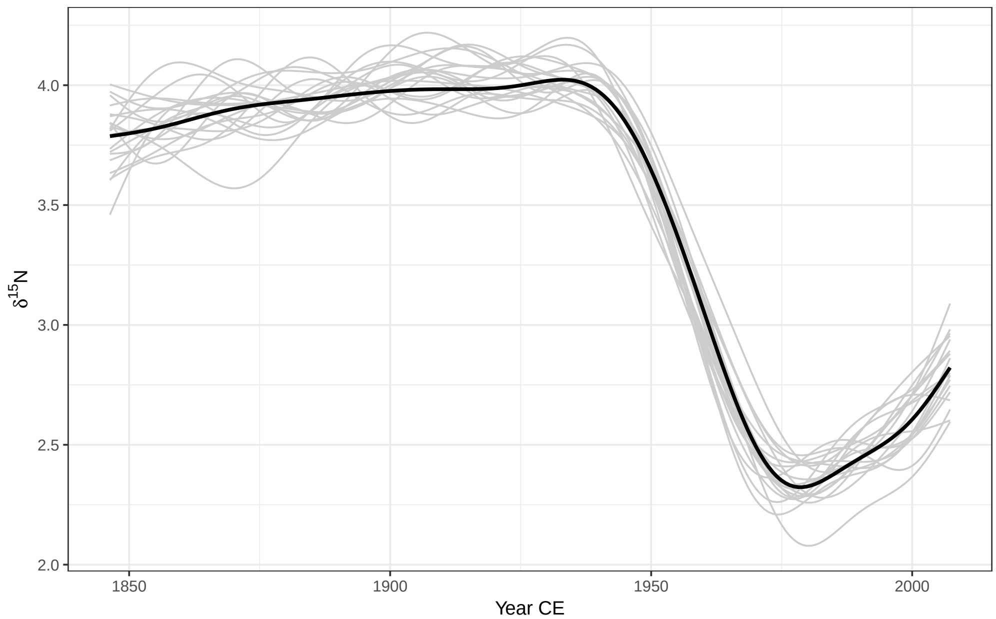
```
#> Model rank =  45 / 45
#>
#> Basis dimension (k) checking results. Low p-value (k-index<1) may
#> indicate that k is too low, especially if edf is close to k'.
#>
#>           k'  edf k-index p-value
#> s(Year) 44.0 28.5    1.27    0.99
```

we note that the value of `k-index` is now greater than 1 and the *p* value suggests there is very little evidence against the null hypothesis. Also note that the `edf` of the estimated smooth is considerably below the maximum (`k'`). This is the effect of the smoothness penalty removing the excessive wiggliness possible with 44 basis functions. If the `edf` were close to `k'`, you might consider increasing `k` by a modest amount (in this instance by 5 or 10 additional basis functions) to assure yourself that you have sufficient basis functions. This has the additional advantage of allowing a richer set of smooths of a given complexity (`edf`) to be represented using the basis functions, which may improve the model fit to the data without changing the `edf` too much.

4 Posterior simulation

Samples from the posterior distribution of a GAM can be drawn using the `simulate()` methods from the *schoenberg* package.

```
set.seed(1) # set the random seed to make this reproducible
nsim <- 20  # how many simulations to draw

## do the simulations
sims <- simulate(mod, nsim = nsim, newdata = newYear, unconditional = TRUE)

## rearrange the output into a long/tidy format
colnames(sims) <- paste0("sim", seq_len(nsim))
sims <- setNames(stack(as.data.frame(sims)), c("simulated", "run"))
sims <- transform(sims, Year = rep(newYear$Year, nsim),
                  simulated = simulated)

## Plot simulated trends
smallSim.plt <- ggplot(newYear, aes(x = Year, y = fit)) +
    geom_line(data = sims,
              mapping = aes(y = simulated, x = Year, group = run),
              colour = "grey80") +
    geom_line(lwd = 1) +
    labs(y = d15n_label, x = "Year CE")
smallSim.plt
```



We repeat the same simulation for Braya-Sø

```
## posterior simulation
## need to reset-up newBraya
newBraya <- with(braya,
                 data.frame(Year = seq(min(Year), max(Year),
                                       length.out = N)))
braya_pred <- cbind(newBraya,
                    data.frame(predict(braya_reml, newBraya,
                                       se.fit = TRUE)))

## simulate
set.seed(1)
sims2 <- simulate(braya_reml, nsim = nsim, newdata = newBraya,
                  unconditional = TRUE)
colnames(sims2) <- paste0("sim", seq_len(nsim))
sims2 <- setNames(stack(as.data.frame(sims2)),
                      c("simulated", "run"))
sims2 <- transform(sims2, Year = rep(newBraya$Year, nsim),
                       simulated = simulated)

brayaSim.plt <- ggplot(braya_pred, aes(x = Year, y = fit)) +
    geom_line(data = sims2,
              mapping = aes(y = simulated, x = Year, group = run),
              colour = "grey80") +
    geom_line(lwd = 1) +
    labs(y = braya_ylabel, x = "Year CE")
brayaSim.plt
```

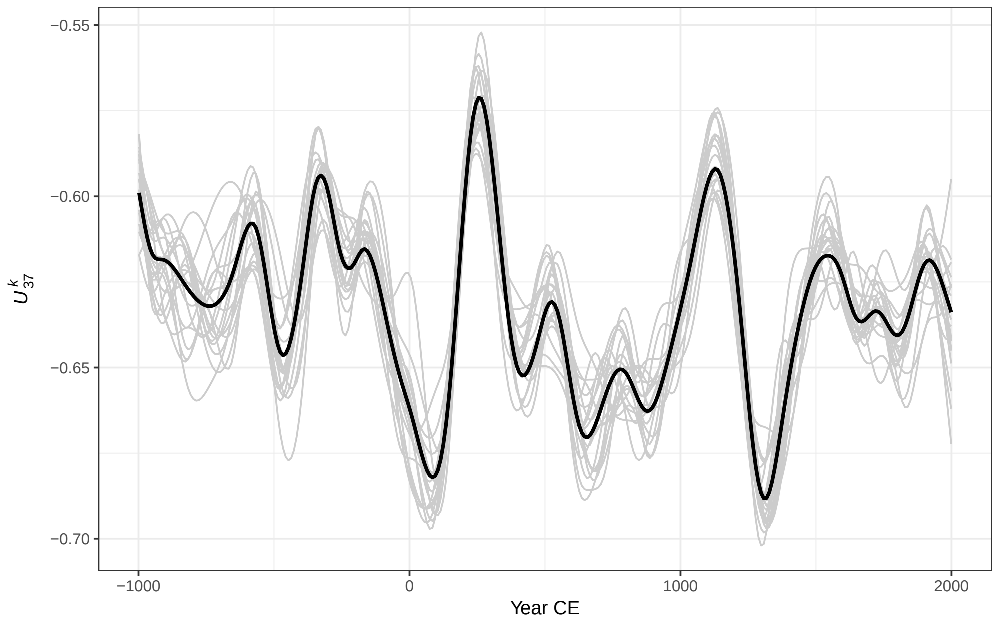


Figure 7 in the manuscript was prepared using

```
plot_grid(smallSim.plt, brayaSim.plt, ncol = 1, labels = "auto",
          align = "hv", axis = "lr")
```

5 Confidence and simultaneous intervals

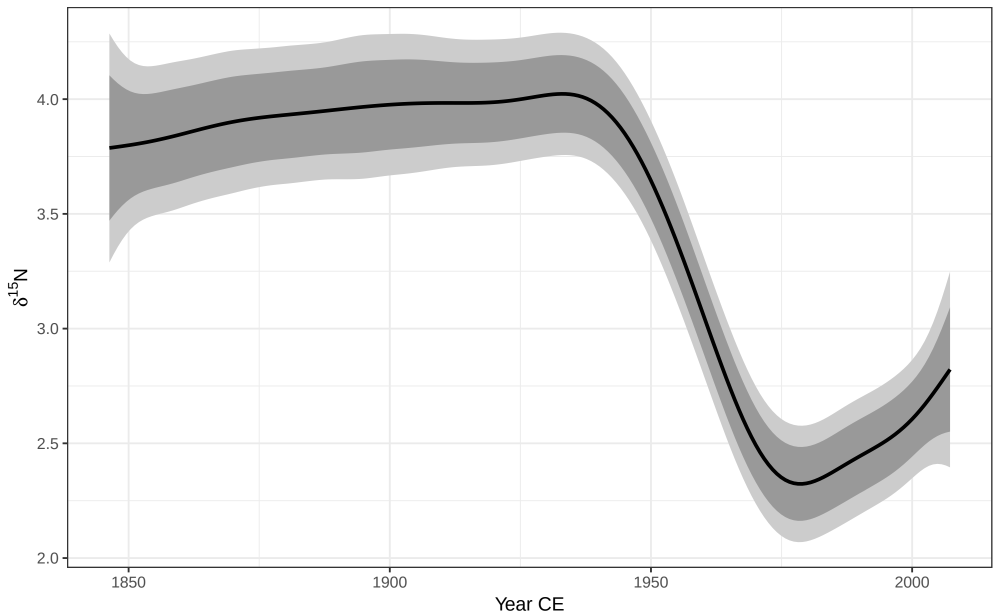
Across-the-function and simultaneous confidence intervals are computed using the `confint()` method. The type of interval required is given via the `type` argument with options `"confidence"` and `"simultaneous"`. Fro example, for Small Water we would use

```
sw.cint <- confint(mod, parm = "Year", newdata = newYear,
                   type = "confidence")
sw.sint <- confint(mod, parm = "Year", newdata = newYear,
                   type = "simultaneous")
head(sw.sint)
```

```
#>     smooth     Year      est         se     crit    lower    upper
#> 1 s(Year) 1846.319 3.787215 0.1616144 3.087688 3.288200 4.286230
#> 2 s(Year) 1847.128 3.789747 0.1509100 3.087688 3.323784 4.255710
#> 3 s(Year) 1847.937 3.792304 0.1411439 3.087688 3.356496 4.228112
#> 4 s(Year) 1848.745 3.794914 0.1324961 3.087688 3.385807 4.204020
#> 5 s(Year) 1849.554 3.797602 0.1251070 3.087688 3.411310 4.183893
#> 6 s(Year) 1850.363 3.800394 0.1190552 3.087688 3.432788 4.167999
```

The `confint()` methods return data frames suitable for subsequent plotting with **ggplot**. The columns labelled `est` and `se` are the estimate values of the smooth and its standard error, respectively. The variables `lower` and `upper` contain the values of the lower and upper bounds on the requested interval. The intervals can be plotted as follows

```
smallInt.plt <- ggplot(sw.cint, aes(x = Year, y = est)) +
    geom_ribbon(data = sw.sint,
                mapping = aes(ymin = lower, ymax = upper, x = Year),
                fill = "grey80", inherit.aes = FALSE) +
    geom_ribbon(mapping = aes(ymin = lower, ymax = upper, x = Year),
                fill = "grey60", inherit.aes = FALSE) +
    geom_line(lwd = 1) +
    labs(y = d15n_label, x = "Year CE")
smallInt.plt
```



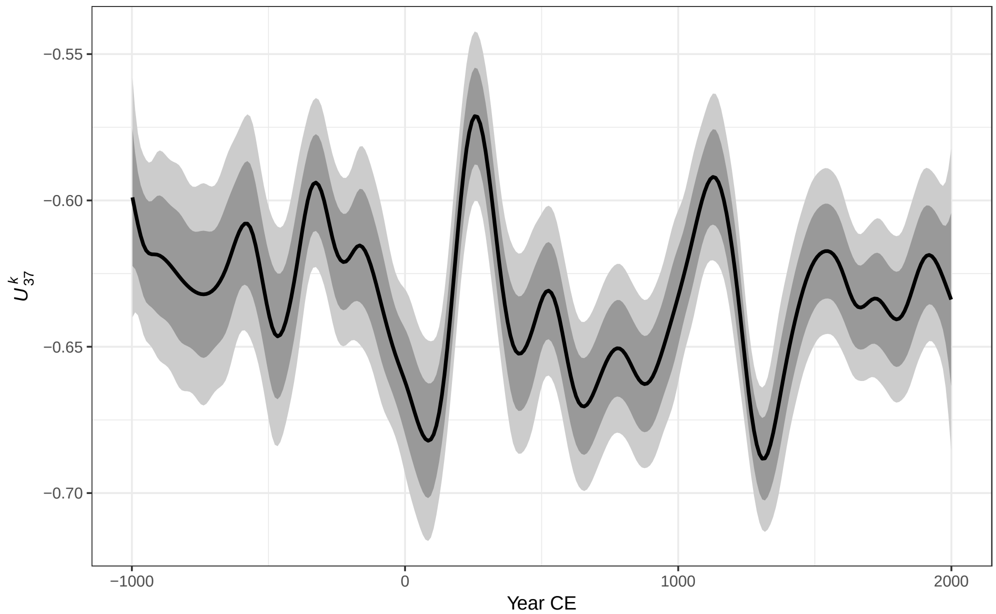
The intervals for Braya-Sø are created

```
bs.cint <- confint(braya_reml, parm = "Year", newdata = newBraya,
                   type = "confidence")
bs.sint <- confint(braya_reml, parm = "Year", newdata = newBraya,
                   type = "simultaneous")
```

and plotted

```
brayaInt.plt <- ggplot(bs.cint, aes(x = Year, y = est)) +
    geom_ribbon(data = bs.sint,
                mapping = aes(ymin = lower, ymax = upper, x = Year),
                fill = "grey80", inherit.aes = FALSE) +
    geom_ribbon(mapping = aes(ymin = lower, ymax = upper, x = Year),
                fill = "grey60", inherit.aes = FALSE) +
    geom_line(lwd = 1) +
    labs(y = braya_ylabel, x = "Year CE")
```

```
brayaInt.plt
```



in the same way

Figure 8 in the manuscript was prepared using

```
plot_grid(smallInt.plt, brayaInt.plt, ncol = 1, labels = "auto",
          align = "hv", axis = "lr")
```

6 Derivatives of the estimated trend

The first derivative of the estimated trend is calculated using finite differences using the `fderiv()` function. There is also a `confint()` method for objects produced by `fderiv()`. The first derivatives and a 95% simultaneous confidence interval for the Small Water trend we computed and plotted using

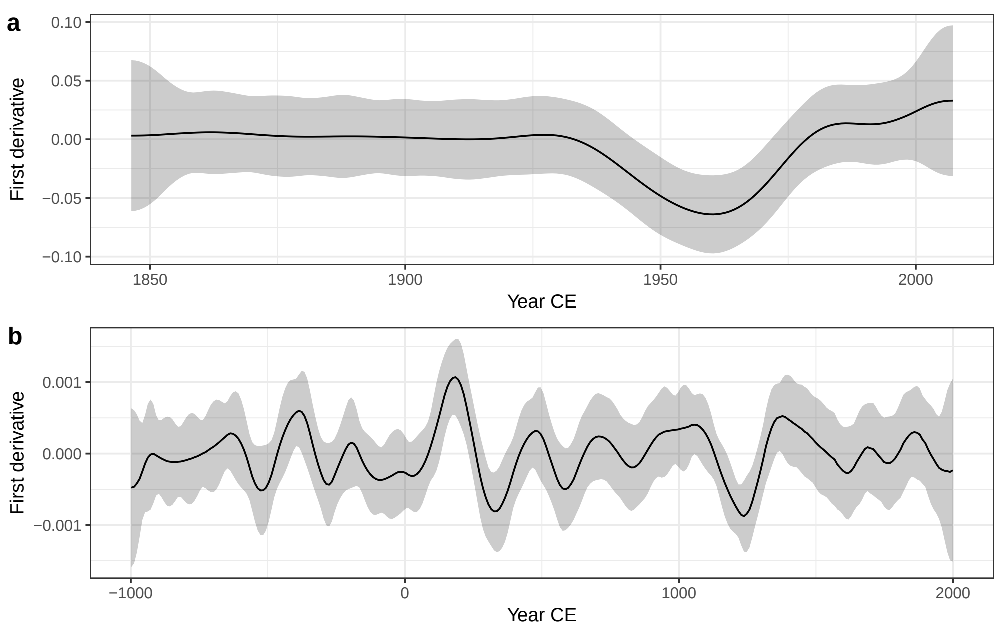
```
small.d <- fderiv(mod, newdata = newYear, n = N)
small.sint <- with(newYear,
                   cbind(confint(small.d, nsim = nsim,
                                 type = "simultaneous"),
                         Year = Year))

small_deriv_plt <- ggplot(small.sint, aes(x = Year, y = est)) +
    geom_ribbon(aes(ymin = lower, ymax = upper), alpha = 0.2,
                fill = "black") +
    geom_line() +
    labs(x = "Year CE", y = "First derivative")
```

whilst for Braya-Sø, the following was used

```
braya.d <- fderiv(braya_reml, newdata = newBraya, n = N)
braya.sint <- with(newBraya,
                   cbind(confint(braya.d, nsim = nsim,
                                 type = "simultaneous"),
                         Year = Year))

braya_deriv_plt <- ggplot(braya.sint, aes(x = Year, y = est)) +
    geom_ribbon(aes(ymin = lower, ymax = upper),
                alpha = 0.2, fill = "black") +
    geom_line() +
    labs(x = "Year CE", y = "First derivative")
```

Figure 9 in the manuscript was prepared using

```
plot_grid(small_deriv_plt, braya_deriv_plt, ncol = 1, labels = "auto",
          align = "hv", axis = "lr")
```



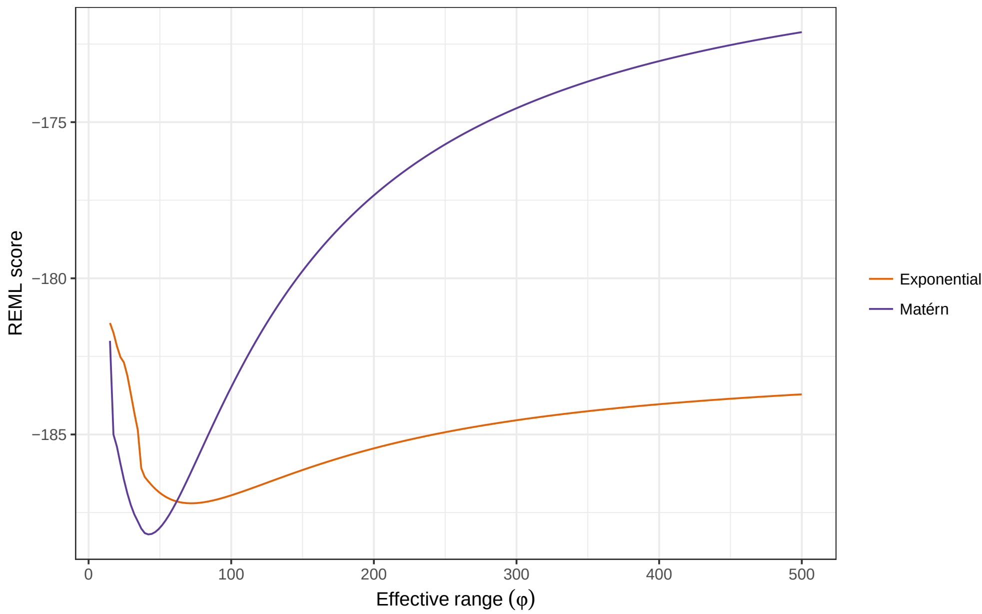
7 Gaussian process smooths

For the Gaussian process smooth to fit within the GAM framework described in the manuscript, we need to supply the value of ϕ for the effective range of the correlation function. To estimate ϕ, we need to repeatedly fit the required GAM using a range of plausible values for ϕ, which we do using a loop. In the chunk below I fit 200 models with ϕ in the range 15–500. For value of ϕ I fit the required GAM and extract the REML score (from component `gcv.ubre`) and store it in the numeric vectors `Mat` or `SEx`, for Matérn and Squared Exponential correlation functions, respectively. The final line of the chunk prepares the REML scores for the two correlation functions in long or tidy format suitable for plotting with *ggplot2*.

```
nn <- 200    # number of points at which to evaluate profile likelihood
dseq <- seq(15, 500, length.out = nn)  # effective ranges to fit at
Mat <- SEx <- numeric(length = nn)     # object to hold model fits

## loop over the sequence of deparation distances and fit the models
for (i in seq_along(dseq)) {
    ## iterate over dseq, fit GP GAM w Matérn covariance
    Mat[i] <- gam(UK37 ~ s(Year, k = 45, bs = "gp", m = c(3, dseq[i])),
                  weights = sampleInterval / mean(sampleInterval),
                  data = braya, method = "REML",
                  family = gaussian())[["gcv.ubre"]]
    ## fit squared exponential
    SEx[i] <- gam(UK37 ~ s(Year, k = 45, bs = "gp", m = c(2, dseq[i], 1)),
                  weights = sampleInterval / mean(sampleInterval),
                  data = braya, method = "REML",
                  family = gaussian())[["gcv.ubre"]]
}

## extract the REML score into ggplot-friendly object
reml.scr <- data.frame(cor = rep(c("Matérn","Exponential"), each = nn),
                       effrange = rep(dseq, 2),
                       reml = c(Mat, SEx))
```

The REML scores for the models are plotted with *ggplot2* using

```
## profile-likelihood plot
proflik.plt <- ggplot(reml.scr, aes(x = effrange, y = reml, colour = cor)) +
    geom_line() +
    scale_colour_manual(name = "", values = c("#e66101","#5e3c99")) +
    labs(y = "REML score", x = expression(Effective ~ range ~ (varphi)))
proflik.plt
```



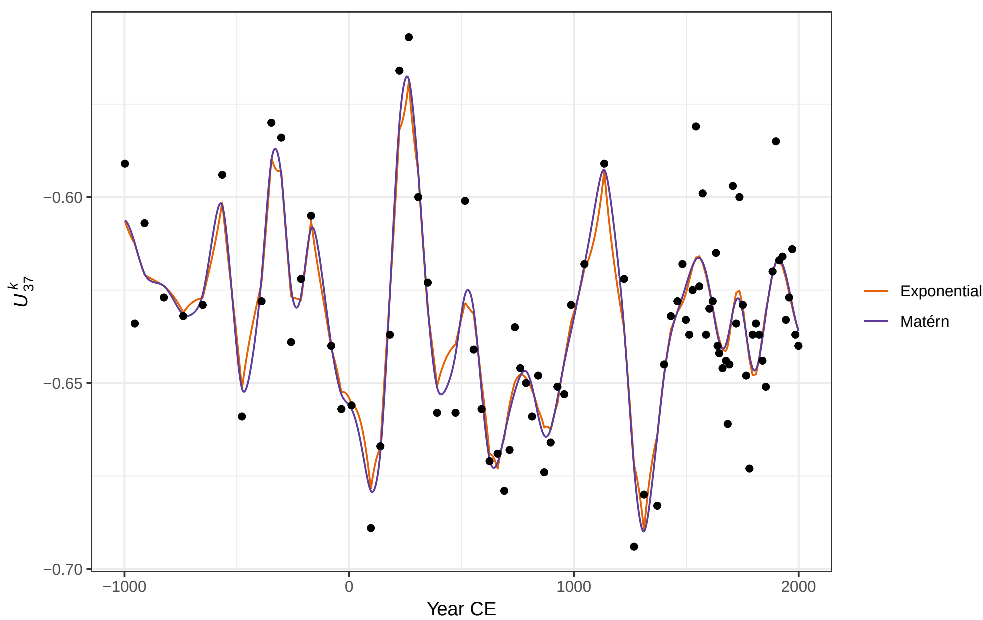
Next we extract the minimum of the REML scores for the two correlation functions and refit those models (we threw away all the models in the `for ()` loop earlier to avoid storing lots of model objects). Then we fit GAMs with Gaussian process smooths using the values of ϕ that produced the minimum REML scores, and predict using the fitted models to visualize the trends.

```
## effective range minima from profile likelihood
effRange2 <- with(subset(reml.scr, cor == "Matérn"), dseq[which.min(reml)])
effRange3 <- with(subset(reml.scr, cor == "Exponential"), dseq[which.min(reml)])

## Refit these models: Matern
gp2 <- gam(UK37 ~ s(Year, k = 45, bs = "gp", m = c(3, effRange2)),
           data = braya,
           method = "REML", weights = sampleInterval / mean(sampleInterval))
## Refit these models: Power exponential
gp3 <- gam(UK37 ~ s(Year, k = 45, bs = "gp", m = c(2, effRange3, 1)),
           data = braya,
           method = "REML", weights = sampleInterval / mean(sampleInterval))

## data to predict at
newd <- with(braya, data.frame(Year = seq(min(Year), max(Year),
                               length.out = 1000)))
## create predictions on response scale for both covariance functions
p.gp2 <- transform(newd,
                   fitted = predict(gp2, newdata = newd, type = "response"),
                   effRange = round(effRange2))
p.gp3 <- transform(newd,
                   fitted = predict(gp3, newdata = newd, type = "response"),
```

```r
                     effRange = round(effRange3))
## joint the two sets of predictions together
pred <- rbind(p.gp2, p.gp3)
## add some categorical variables for plotting
pred <- transform(pred,
                  effRange = factor(effRange),
                  cor = rep(c("Matérn", "Exponential"), each = nrow(newd)))
```

The estimated Gaussian process trends are plotted using

```r
gp.plt2 <- ggplot(pred, aes(x = Year, y = fitted, colour = cor)) +
    geom_line() + theme(legend.position = "right") +
    geom_point(aes(x = Year, y = UK37), data = braya, inherit.aes = FALSE) +
    scale_colour_manual(name = "", values = c("#e66101","#5e3c99")) +
    labs(y = braya_ylabel, x = "Year CE")
gp.plt2
```



whilst Figure 12 in the manuscript was prepared using

```r
plot_grid(proflik.plt, gp.plt2, ncol = 1, labels = c("a","b"),
          align = "hv", axis = "lr")
```

8 Adaptive smooths

The adaptive smooth was fitted to the Braya-Sø data by adding `bs = "ad"` to the `s()` term in the model formula. The other aspects of the fit are as previously used for the other models, REML smoothness selection and observational weights:

```r
## Adaptive spline, weights as sampleInterval
mod_ad <- gam(UK37 ~ s(Year, k = 45, bs = "ad"), data = braya,
              method = "REML",
              weights = sampleInterval / mean(sampleInterval))
```

9 Comparing trends

For the model comparison, I refitted all the models for consistency; the code to fit each of the

1. Thin plate regression spline (TPRS),
2. Gaussian process spline (Matérn correlation functions), and
3. Adaptive smoother

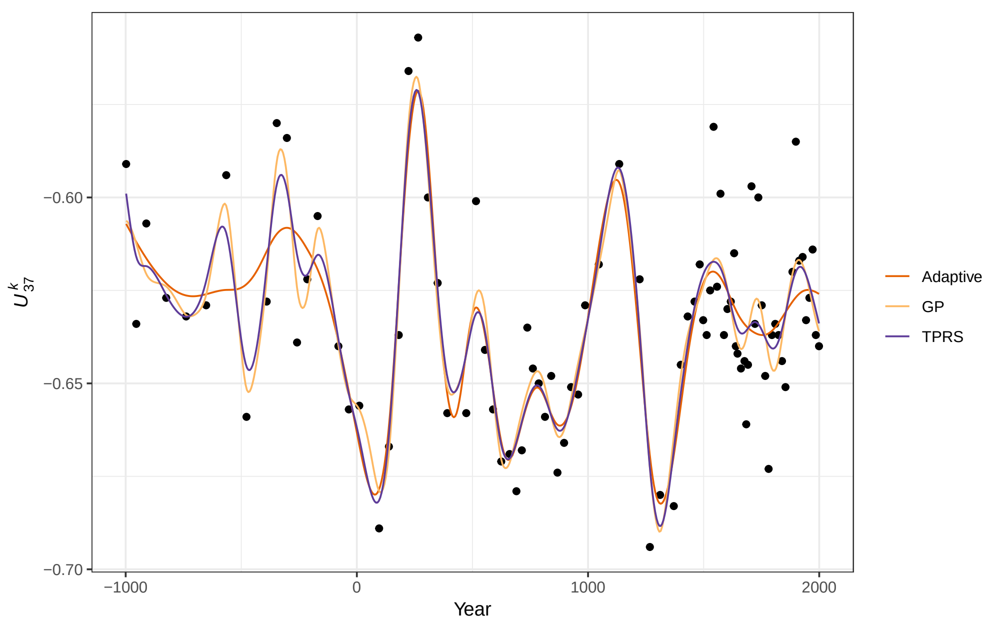
is shown below.

```r
## effective range parameter from profile-likelihood of Matern model
effRange <- effRange2

## TPRS, weights as sampleInterval
mod_tprs <- gam(UK37 ~ s(Year, k = 45, bs = "tp"), data = braya,
                method = "REML",
                weights = sampleInterval / mean(sampleInterval))

## Gaussian process, Matern, kappa = 1.5, weights as sampleInterval
mod_gp <- gam(UK37 ~ s(Year, k = 45, bs = "gp", m = c(3, effRange)),
              data = braya,
              method = "REML",
              weights = sampleInterval / mean(sampleInterval))

## Adaptive spline, weights as sampleInterval
mod_ad <- gam(UK37 ~ s(Year, k = 45, bs = "ad"), data = braya,
              method = "REML",
              weights = sampleInterval / mean(sampleInterval))
```

We write a small function to predict from each model over the range of `Year` and return the data in tidy format for plotting. The plot produce reproduces figure 13 in the manuscript.

```r
## wrap this in a function that will return all the plots & derived objects
processGAM <- function(mod) {
    ## Predict from model
    N <- 500
    newYear <- with(braya,
                    data.frame(Year = seq(min(Year), max(Year),
                                          length.out = N)))
    newYear <- cbind(newYear,
```

```r
                        data.frame(predict(mod, newYear, se.fit = TRUE)))

    out <- list(objects = newYear)
    out
}

plts_gp   <- processGAM(mod = mod_gp) # Gaussian process smooth with weights
plts_ad   <- processGAM(mod = mod_ad) # Adaptive smooth with weights
plts_tprs <- processGAM(mod = mod_tprs) # TPRS with weights

pltData <- do.call("rbind", lapply(list(plts_gp, plts_ad, plts_tprs),
                                   `[[`, "objects"))
pltData <- transform(pltData, Model = rep(c("GP", "Adaptive", "TPRS"),
                                          each = nrow(plts_gp$objects)))

allFits <- ggplot(pltData, aes(x = Year, y = fit)) +
    geom_point(aes(x = Year, y = UK37), data = braya) +
    geom_line(aes(colour = Model)) + labs(y = braya_ylabel, x = "Year") +
    theme(legend.position = "right") +
    scale_colour_manual(name = "",
                        values = c("#e66101", "#fdb863", "#5e3c99"))
allFits
```



10 Accounting for age-model uncertainty

The manuscript proposed to simulate from the posterior distribution of the fitted age model as a way to account for age-model uncertainty. The first step in the process is to fit the age model from which to simulate new age models. This was done using the *scam* package for a *shape-constrained GAM*, with the age-model spline constrained to be monotonic decreasing (`bs = "mpd"`).

To make this section self-contained, I refitted the Small Water GAM plus CAR(1) model

```
knots <- with(small, list(Year = seq(min(Year), max(Year), length = 14)))
mod <- gamm(d15N ~ s(Year, k = 15), data = small, method = "REML",
            correlation = corCAR1(form = ~ Year),
            knots = knots)
```

and then we load the ^{210}Pb dating results for the dated core sections.

```
swAge <- read.csv("./data/small-water/small1-dating.csv")
```

before fitting the shape-constrained GAM. Currently, *scam* can only fit models using GCV smoothness selection. I used the `gamma` argument here to add a larger penalty for more-complex models. Each effective degree of freedom used by the spline is counted as 1.4 degrees of freedom in the GCV score.

```
## monotonic spline age-depth model
swAge$Error[1] <- 1.1
swAgeMod <- scam(Date ~ s(Depth, k = 5, bs = "mpd"), data = swAge,
                 weights = 1 / swAge$Error, gamma = 1.4)
```

Note that I added a small amount of error to the surface sample age as the model cannot be fitted if an observation has `0` weight.

Next, predict from the estimated age model, and draw 25 samples from the posterior distribution using `simulate()`. The results are tidied into a format suitable for further processing and plotting. Note that the posterior samples here are only used for plotting.

```
## predict from the age model for a smooth set of points in `Depth`
newAge <- with(swAge, data.frame(Depth = seq(min(Depth), max(Depth),
                                             length.out = 200)))
newAge <- transform(newAge,
                    fitted = predict(swAgeMod, newdata = newAge,
                                     type = "response"))
newSims <- as.data.frame(simulate(swAgeMod, nsim = 25, newdata = newAge))
newSims <- cbind(Depth = newAge$Depth, newSims)
newSims <- gather(newSims, Simulation, Age, -Depth)
```

In the next code chunk, I draw 100 samples from the posterior distribution of the age model, but notice that I pass in the `small` data to `newdata` in the call to `simulate()` as the locations I want new age estimates for are the depths for which we have δ^{15}N values. A small function (`fitSWModels`) is written to prepare each simulation for fitting and then actually fit the GAM plus CAR(1) model using the updated age information.

```
## simulate from age model; each column is a simulation
ageSims <- simulate(swAgeMod, nsim = 100, newdata = small, seed = 42)
ageSims <- as.data.frame(ageSims)

fitSWModels <- function(x, y, knots) {
    dat <- data.frame(d15N = y, Year = x)
    m <- gamm(d15N ~ s(Year, k = 15), data = dat, method = "REML",
              correlation = corCAR1(form = ~ Year), knots = knots)
}

## generate new trends using draws from age-model posterior
simTrendMods <- lapply(ageSims, fitSWModels, y = small$d15N, knots = knots)


## function wrapper to predict new trends at locations over the
## range of `Year`
predSWModels <- function(mod, newdata) {
    predict(mod$gam, newdata = newdata, type = "response")
}

## predict from fitted model to produce a smooth trend for each posterior
## sample
simTrends <- lapply(simTrendMods, predSWModels, newdata = newYear)

## arrange in a tidy format form plottings
simTrends <- data.frame(Year  = with(newYear, rep(Year, length(simTrends))),
                        Trend = unlist(simTrends),
                        Group = rep(seq_along(simTrends),
                                    times = lengths(simTrends)))
```

The next chunk does the final step in the process. For each of the models we just fitted to include age model uncertainty, we simulate 50 draws from the model posterior distribution. We start with a wrapper function around the `simulate()` code we want to run on each model, then do the actual posterior draws for each model using `lapply()`. The final step just arranges data for plotting.

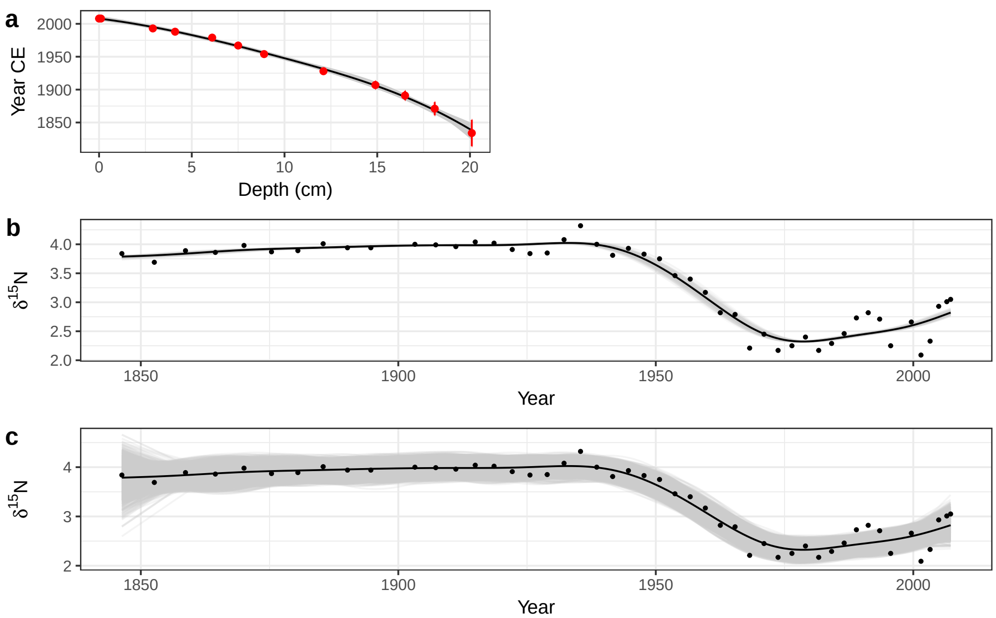
```
## wrapper to simulate from a fitted GAM with the required arguments
simulateSWModels <- function(mod, newdata, nsim, seed = 42) {
    sims <- simulate(mod, nsim = nsim, newdata = newdata, seed = seed)
    as.vector(sims)
}

## now do the posterior simulation
```

```r
NSIM <- 50      # number of posterior samples *per* model
simSimulate <- lapply(simTrendMods, simulateSWModels, newdata = newYear,
                      nsim = NSIM, seed = 42)

## arrange in a tidy format
simSimulate <-
  data.frame(Year  = with(newYear,
                          rep(Year, times = NSIM * length(simSimulate))),
             Trend = unlist(simSimulate),
             Group = rep(seq_len(NSIM * length(simSimulate)),
                         each = nrow(newYear)))
```

Each of the steps is visualized using the plot code shown below.

```r
plt1 <- ggplot(swAge, aes(y = Date, x = Depth)) +
    geom_line(data = newSims,
              mapping = aes(y = Age, x = Depth, group = Simulation),
              alpha = 1, colour = "grey80") +
    geom_line(data = newAge, mapping = aes(y = fitted, x = Depth)) +
    geom_point(size = 1.5, colour = "red") +
    geom_errorbar(aes(ymin = Date - Error, ymax = Date + Error, width = 0),
                  colour = "red") +
    labs(y = "Year CE", x = "Depth (cm)")

plt2 <- ggplot(simTrends, aes(x = Year, y = Trend, group = Group)) +
    geom_line(alpha = 0.1, colour = "grey80") +
    geom_line(data = newYear,
              mapping = aes(x = Year, y = fit), inherit.aes = FALSE) +
    geom_point(data = small,
               mapping = aes(x = Year, y = d15N),
               inherit.aes = FALSE, size = 0.7) +
    labs(x = "Year", y = d15n_label)

plt3 <- ggplot(simSimulate, aes(x = Year, y = Trend, group = Group)) +
    geom_line(alpha = 0.2, colour = "grey80") +
    geom_point(data = small,
               mapping = aes(x = Year, y = d15N),
               inherit.aes = FALSE,
               size = 0.7) +
    geom_line(data = newYear,
              mapping = aes(x = Year, y = fit),
              inherit.aes = FALSE) +
    labs(x = "Year", y = d15n_label)

## align all plots vertically
```

```r
plots <- align_plots(plt1, plt2, plt3, align = 'v', axis = 'l')


## create the two rows of figures, from `plots`
top_row <- plot_grid(plots[[1]], NULL, ncol = 2, labels = "a")
bot_row <- plot_grid(plots[[2]], plots[[3]], ncol = 1, labels = c("b", "c"))

## combine the two rows, top row has 1 plot row, bottom row has 2, hence
## the rel_heights to even this out
plot_grid(top_row, bot_row, ncol = 1, rel_heights = c(0.5, 1))
```



This reproduces figure 14 from the manuscript.

11 Session information

```r
devtools::session_info()
```

```
#> Session info -------------------------------------------------------------
#>  setting  value
#>  version  R version 3.4.4 Patched (2018-03-17 r74422)
#>  system   x86_64, linux-gnu
#>  ui       X11
#>  language (EN)
#>  collate  en_CA.UTF-8
#>  tz       America/Regina
```

```
#>  date     2018-05-14
#> Packages -----------------------------------------------------------------
#>  package    * version date       source
#>  backports    1.1.2   2017-12-13 CRAN (R 3.4.3)
#>  base       * 3.4.4   2018-03-18 local
#>  codetools    0.2-15  2016-10-05 CRAN (R 3.4.4)
#>  colorspace   1.3-2   2016-12-14 CRAN (R 3.4.1)
#>  compiler     3.4.4   2018-03-18 local
#>  cowplot    * 0.9.2   2017-12-17 CRAN (R 3.4.3)
#>  datasets   * 3.4.4   2018-03-18 local
#>  devtools     1.13.5  2018-02-18 CRAN (R 3.4.4)
#>  digest       0.6.15  2018-01-28 CRAN (R 3.4.4)
#>  evaluate     0.10.1  2017-06-24 CRAN (R 3.4.1)
#>  ggplot2    * 2.2.1   2016-12-30 CRAN (R 3.4.1)
#>  glue         1.2.0   2017-10-29 CRAN (R 3.4.2)
#>  graphics   * 3.4.4   2018-03-18 local
#>  grDevices  * 3.4.4   2018-03-18 local
#>  grid         3.4.4   2018-03-18 local
#>  gtable       0.2.0   2016-02-26 CRAN (R 3.4.1)
#>  htmltools    0.3.6   2017-04-28 CRAN (R 3.4.1)
#>  knitr        1.20    2018-02-20 CRAN (R 3.4.4)
#>  labeling     0.3     2014-08-23 CRAN (R 3.4.1)
#>  lattice      0.20-35 2017-03-25 CRAN (R 3.4.4)
#>  lazyeval     0.2.1   2017-10-29 CRAN (R 3.4.2)
#>  magrittr     1.5     2014-11-22 CRAN (R 3.4.1)
#>  MASS         7.3-49  2018-02-23 CRAN (R 3.4.4)
#>  Matrix       1.2-14  2018-04-09 CRAN (R 3.4.4)
#>  memoise      1.1.0   2017-04-21 CRAN (R 3.4.1)
#>  methods      3.4.4   2018-03-18 local
#>  mgcv       * 1.8-23  2018-01-21 CRAN (R 3.4.4)
#>  munsell      0.4.3   2016-02-13 CRAN (R 3.4.1)
#>  nlme       * 3.1-137 2018-04-07 CRAN (R 3.4.4)
#>  pillar       1.2.1   2018-02-27 CRAN (R 3.4.4)
#>  plyr         1.8.4   2016-06-08 CRAN (R 3.4.1)
#>  purrr        0.2.4   2017-10-18 CRAN (R 3.4.2)
#>  Rcpp         0.12.16 2018-03-13 CRAN (R 3.4.4)
#>  rlang        0.2.0   2018-02-20 CRAN (R 3.4.4)
#>  rmarkdown  * 1.9     2018-03-01 CRAN (R 3.4.4)
#>  rprojroot    1.3-2   2018-01-03 CRAN (R 3.4.3)
#>  scales       0.5.0   2017-08-24 CRAN (R 3.4.1)
#>  scam       * 1.2-2   2017-09-24 CRAN (R 3.4.3)
#>  schoenberg * 0.0-6   2018-04-12 Github (gavinsimpson/schoenberg@f9b8a84)
#>  splines      3.4.4   2018-03-18 local
#>  stats      * 3.4.4   2018-03-18 local
```

```
#>  stringi       1.1.7   2018-03-12 CRAN (R 3.4.4)
#>  stringr       1.3.0   2018-02-19 CRAN (R 3.4.4)
#>  tibble        1.4.2   2018-01-22 CRAN (R 3.4.3)
#>  tidyr       * 0.8.0   2018-01-29 CRAN (R 3.4.4)
#>  tools         3.4.4   2018-03-18 local
#>  utils       * 3.4.4   2018-03-18 local
#>  withr         2.1.2   2018-03-15 CRAN (R 3.4.4)
#>  yaml          2.1.18  2018-03-08 CRAN (R 3.4.4)
```