ADMUR: Ancient Demographic Modelling Using Radiocarbon

Adrian Timpson

2020-09-18

UNDER CONSTRUCTION. COMPLETION DUE SEPT 2020

This vignette provides a comprehensive guide to modelling population dynamics using the R package ADMUR, and accompanies the publication 'Directly modelling population dynamics in the South American Arid Diagonal using 14C dates', Philosophical Transactions B, 2020, A. Timpson et al. It assumes the user has some basic familiarity with programming in R.

In addition to some basic information on getting started, this vignette is divided into three main parts:

1. **Date calibration and SPDs:** Calibrate individual radiocarbon dates, generate Summed Probability Distributions, and automatically phase large datasets to adjust for ascertainment bias. Comparison with other software.
2. **Continuous Piecewise Linear (CPL) Modelling:** Parameter estimation using the likelihood function, and its application in MCMC for estimating credible intervals. Model selection using BIC, and the goodness-of-fit (GOF) test.
3. **SPD simulation testing:** Null hypothesis testing by simulating SPDs and comparing to the observed SPD. P-values generated using likelihoods, improving on the summary statistics originally devised in Shennan et al 2013 and Timpson et al 2014.

Installation

The ADMUR package can be installed directly from GitHub, after installing and loading the 'devtools' package on the CRAN:

```
install.packages('devtools')
library(devtools)
install_github('UCL/ADMUR')
```

The ADMUR package can then be locally loaded:

```
library(ADMUR)
```

14C datasets

A summary of the available help files and data sets included in the package can be browsed, which include a terrestrial anthropogenic ^{14}C dataset from the South American Arid Diagonal:

```
help(ADMUR)
help(arid)
```

Datasets must be structured as a data frame that include columns 'age' and 'sd', which represent the uncalibrated ^{14}C age and its error, respectively.

```
arid[1:5,1:8]
```

```
##   UniqID          site       lat      long  age sd      LabNo Material_D
## 1    237 Villa del Mar -17.62295 -71.34017 6360 60 Beta-71133       bone
## 2    240          Yara -17.51998 -71.36716 5020 60 Beta-80970       bone
## 3    505    El Ahogado -17.96667 -70.88333 3515 40    Pa-1769   charcoal
## 4    506    El Ahogado -17.96667 -70.88333 3535 60    Pa-1768   charcoal
## 5    507    El Ahogado -17.96667 -70.88333 3660 40    Pa-1789   charcoal
```

Part 1

Date calibration and SPDs

The algorithm used by ADMUR to calculate model likelihoods of a ^{14}C dataset uses several functions to first calibrate ^{14}C dates. These functions are also intrinsically useful for ^{14}C date calibration or for generating a Summed Probability Distribution (SPD).

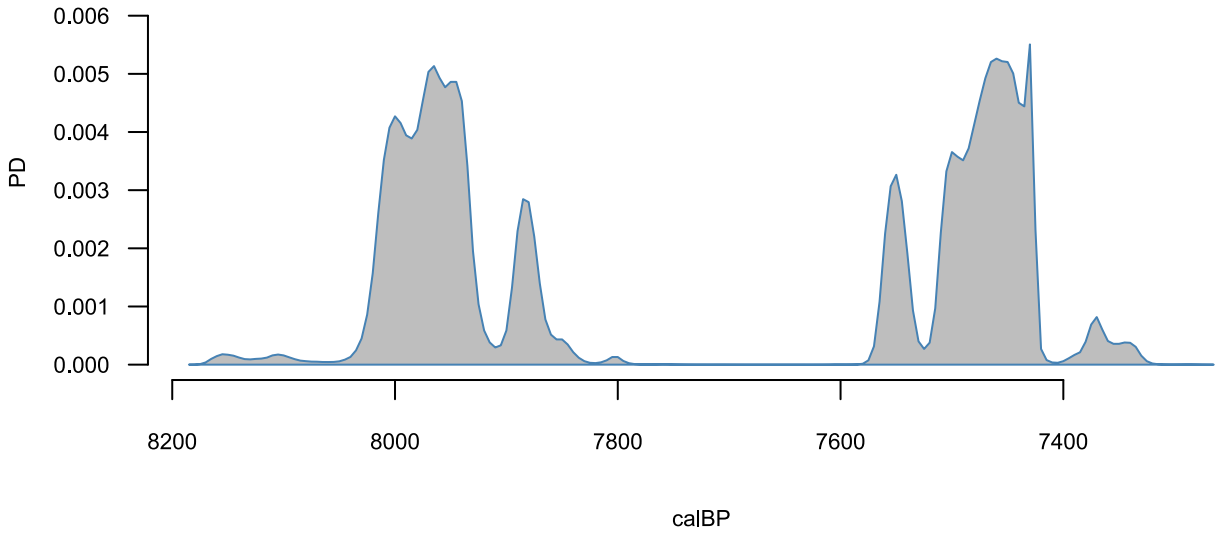
Calibrated 14C date probability distributions

Generating a single calibrated date distribution or SPD requires either a two-step process to give the user full control of the date range and temporal resolution, or a simpler one step process using a wrapper function that automatically estimates a sensible date range and resolution from the dataset, performs the two step process internally, and plots the SPD.

With the wrapper

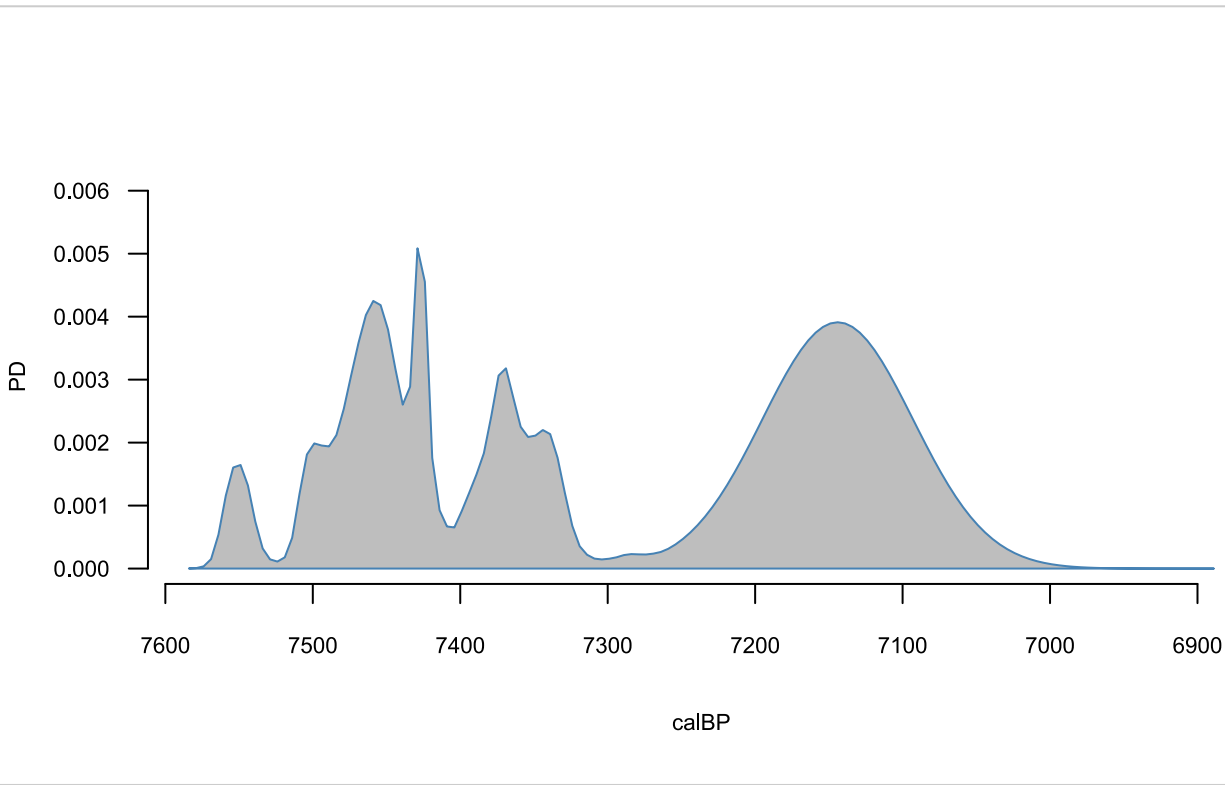
1. Use the function summedCalibratorWrapper()

```
data <- data.frame( age = c(6562,7144),  sd = c(44,51)  )
x <- summedCalibratorWrapper(data)
```



Notice the function automatically assumed the data provided were all ^{14}C dates. However, if you have other kinds of date such as thermoluminescence you can specify 'nonC14'. You can also specify a particular calibration curve:

```
data <- data.frame( age = c(6562,7144),  sd = c(44,51), datingType = c('C14','nonC14') )
x <- summedCalibratorWrapper(data, shcal20)
```



Without the wrapper

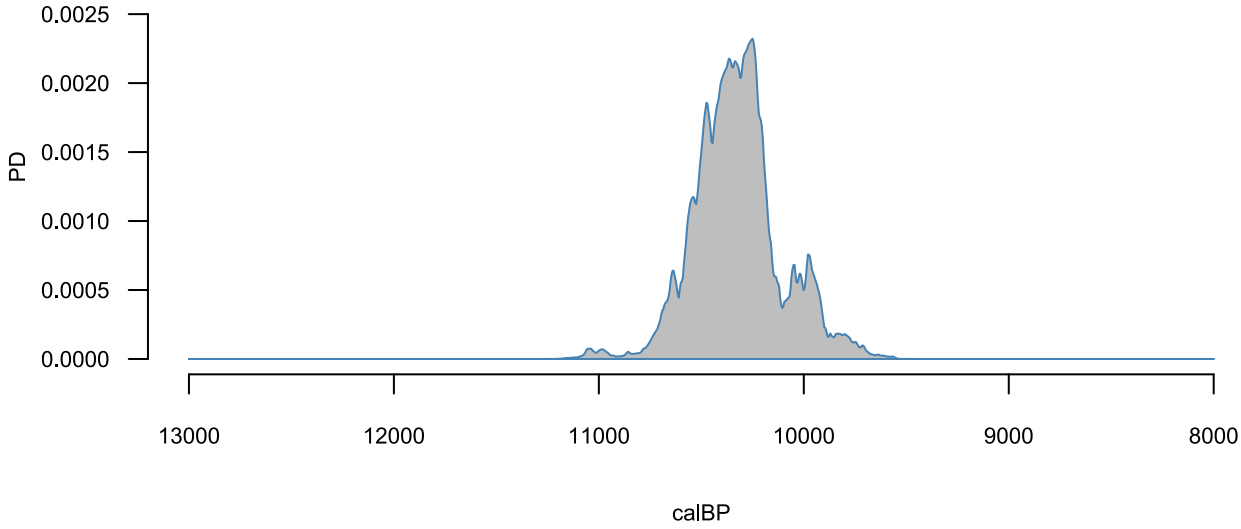
Generating the SPD without the wrapper gives you more control, and requires a two-step process:

1. Convert a calibration curve to a CalArray using the function makeCalArray()

2. Calibrate the ^{14}C dates through the CalArray using the function summedCalibrator().

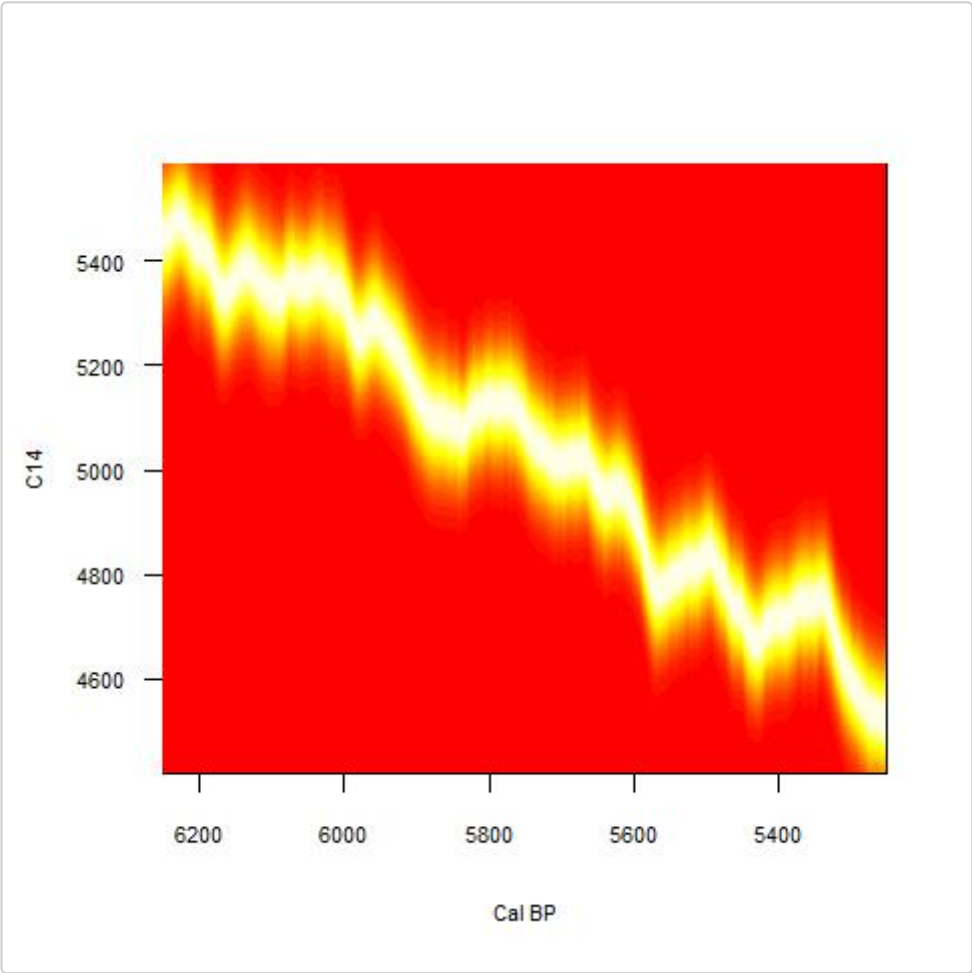
This is useful for improving computational times if generating many SPDs, for example in a simulation framework, since the CalArray needs generating only once.

```
data <- data.frame(age = c(9144), sd=c(151) )
CalArray <- makeCalArray( intcal13, calrange = c(8000,13000) )
cal <- summedCalibrator(data, CalArray)
plotPD(cal)
```



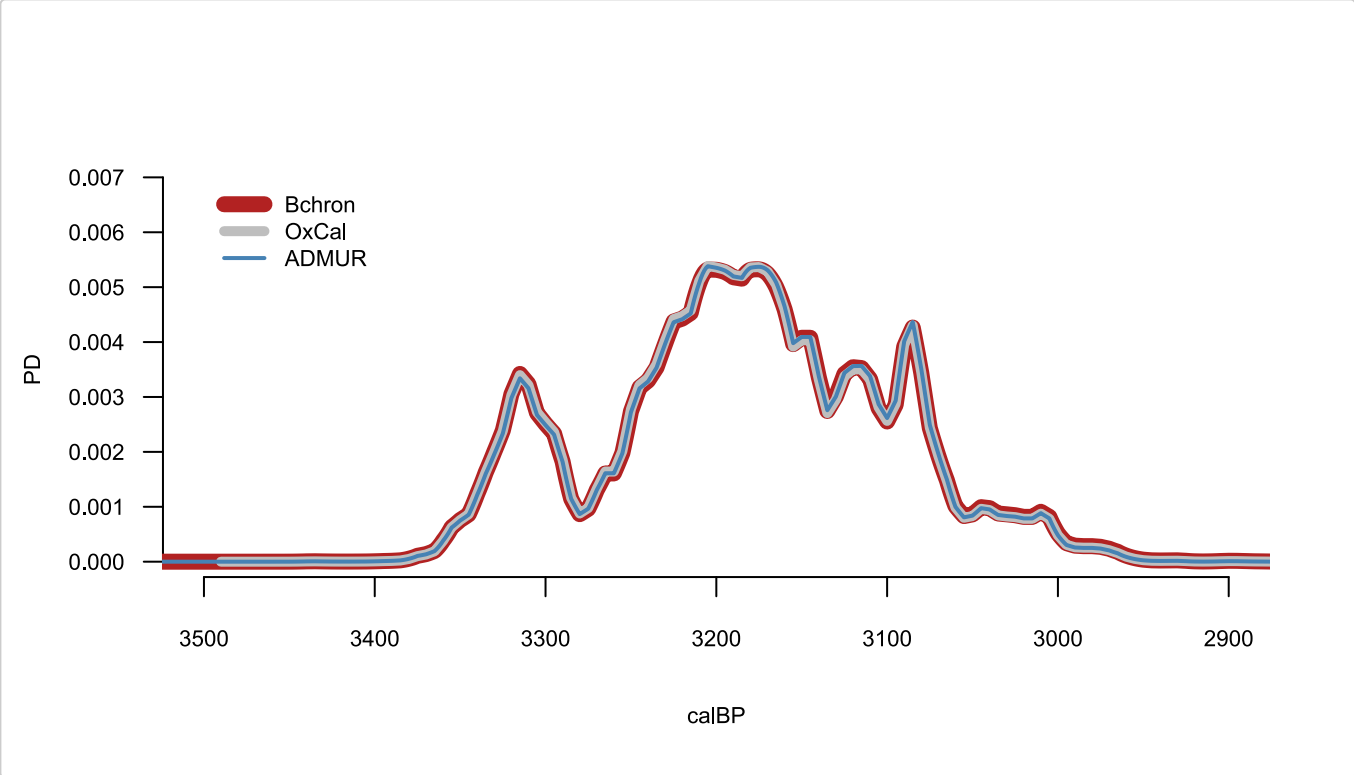
The CalArray is essentially a two-dimensional probability array of the calibration curve, and can be viewed using the plotCalArray() function. Calibration curves vary in their temporal resolution, and the preferred resolution can be specified using the parameter **inc** which interpolates the calibration curve. It would become prohibitively time and memory costly if analysing the entire 50,000 year range of the calibration curve at a 1 year resolution (requiring a 50,000 by 50,000 array) and in practice the default 5 year resolution provides equivalent results to 1 year resolution for study periods wider than c.1000 years.

```
x <- makeCalArray(shcal13, c(5500,6000), inc=1 )
plotCalArray(x)
```



Calibration comparison with other software

It is worth noting that the algorithm used by this package to calibrate ^{14}C dates gives practically equivalent results to those from OxCal generated using OxcAAR and Bchron

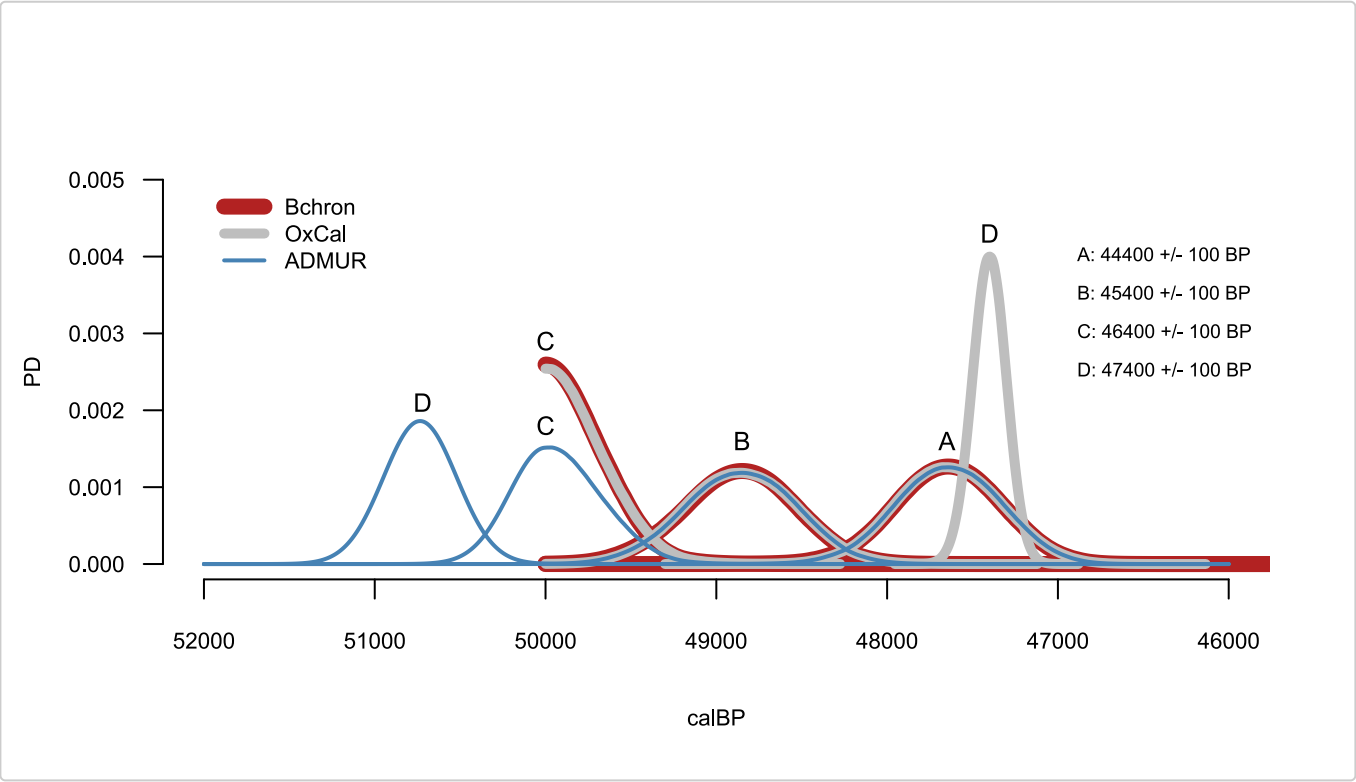


Comparison of calibration software for the ^{14}C date: 3000 +/- 50 BP calibrated through intcal13.

However, there are two fringe circumstances where these software programs differ substantially: at the border of the calibration curve; and if a date has a large error.

Border effects

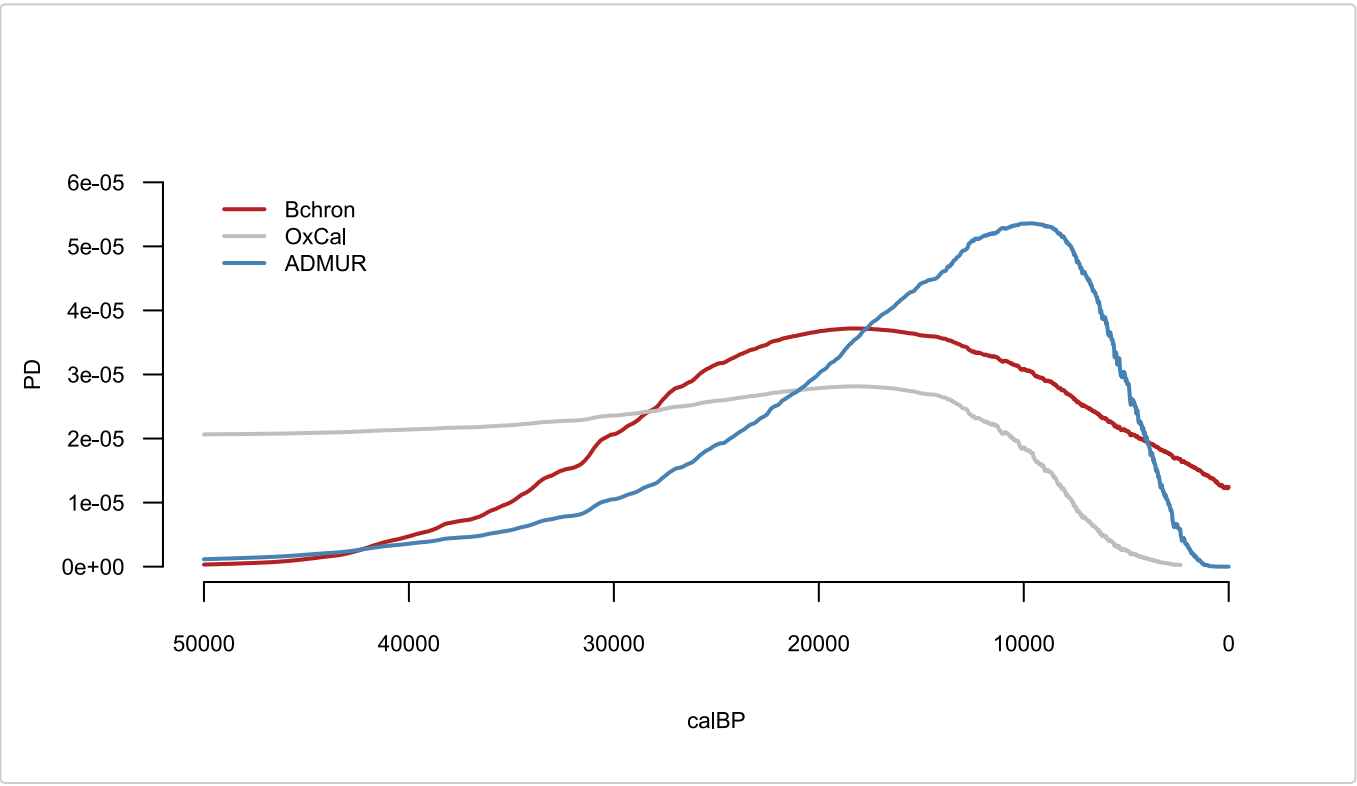
Consider the real ^{14}C date MAMS-13035 age: 50524 +/- 833 BP calibrated through intcal13, which only extends to 46401BP. Bchron throws an error, whilst OxCal applies a one-to-one mapping between Conventional Radiocarbon (CRA) time and calendar time for any date (mean) beyond the range of the calibration curve. The latter is in theory a reasonable way to mitigate the problem, however OxCal applies this in a binary manner that can create peculiarities. Instead ADMUR gradually fades the calibration curve to a one-to-one mapping between the end of the curve and 60,000 BP.



Comparison of calibration software at the limits of intcal13. OxCal and Bchron produce a truncated distribution for date C. Bchron cannot calibrate date D, and OxCal suggests date D is younger than dates A, B and C. ADMUR performs a soft fade at the limit of the calibration curve.

Large errors

A ^{14}C date is typically reported as a mean date with an error, which is generally interpreted as representing a symmetric Gaussian distribution before calibration. However, a Gaussian has a non-zero probability at all possible years (between $-\infty$ and $+\infty$), and therefore cannot fairly represent the date uncertainty which must be skewed towards the past. Specifically, if we consider the date in CRA time, it must have a zero probability of occurring in the future. Alternatively, if we consider the date as a 14C/12C ratio, it cannot be smaller than 1 (the present). Therefore ADMUR interprets a ^{14}C date as a lognormal distribution. This naturally skews the distribution away from the present. In practice, this difference is undetectably trivial, since a lognormal distribution approximates a normal distribution for typical radiocarbon errors, however, theoretically the differences can be large if considering dates with large errors that are close to the present.



Comparison of calibration software for the ^{14}C date 15000 +/- 10000 BP, using intcal13. Bchron assumes the CRA error is Normally distributed, resulting in a truncated curve with a substantial probability at present. OxCal produces a heavily skewed distribution with a low probability at present and a substantial probability at 50,000 BP that suddenly truncates to zero beyond this (total probability mass of all three curves equals 1). ADMUR assumes the CRA error is Lognormally distributed, such that the mean = 15000 and the variance = 10000^2.

Phased data: adjusting for ascertainment bias

A naive approach to generating an SPD as a proxy for population dynamics would be to sum all dates in the dataset, but a more sensible approach is to sum the SPDs of each phase. The need to bin dates into phases is an important step in modelling population dynamics to adjust for the data ascertainment bias of some archaeological finds having more dates by virtue of a larger research interest or budget.

Therefore phaseCalibrator() generates an SPD for each phase in a dataset, and includes a binning algorithm which provides a useful solution to handling large datasets that have not aready been phased. For example, consider the following 8 dates from 2 sites:

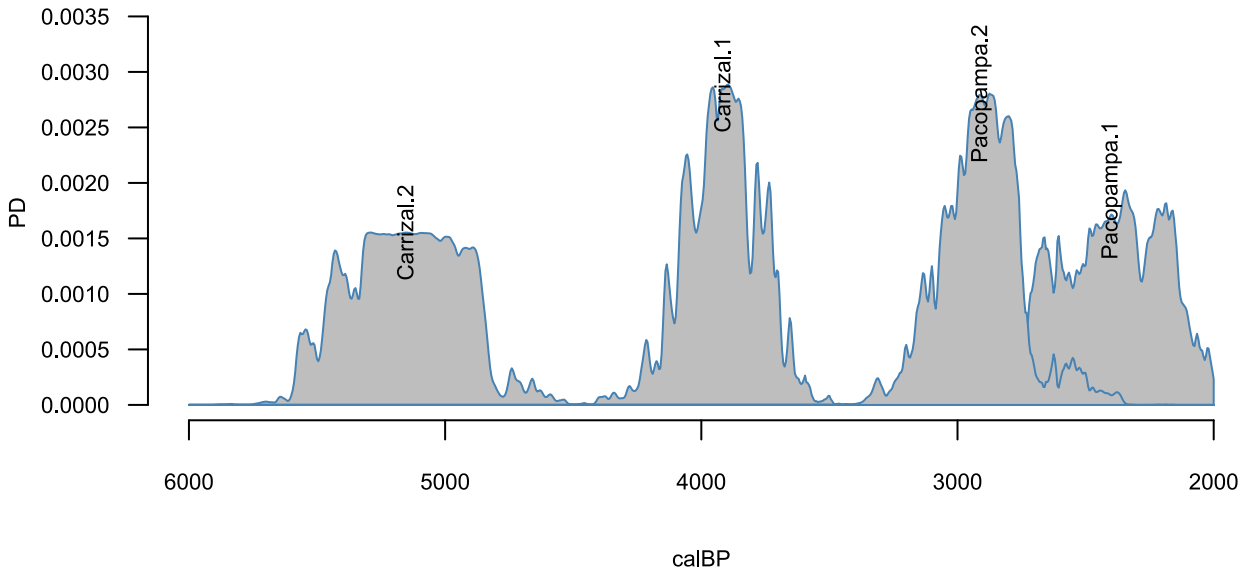
```
data <- subset(arid, site %in% c('Carrizal','Pacopampa'))
data[,2:7]
```

```
##           site       lat      long  age  sd      LabNo
## 1192  Carrizal -6.063056 -79.49806 3640 100 Beta-31075
## 1193  Carrizal -6.063056 -79.49806 4390 110 Beta-18920
## 1194  Carrizal -6.063056 -79.49806 4450 100 Beta-31073
## 1195  Carrizal -6.063056 -79.49806 4620 100 Beta-31074
## 1196  Carrizal -6.063056 -79.49806 4690 120 Beta-27417
## 1205 Pacopampa -6.200000 -79.01000 2385 155     SI-794
## 1206 Pacopampa -6.200000 -79.01000 2765 135     SI-792
## 1207 Pacopampa -6.200000 -79.01000 2855  95     SI-793
```

The data have not already been phased (do not include a column ‘phase’) therefore the default binning algorithm calibrates these dates into 4 phases. this is achieved by binning dates that have a mean ^{14}C

date within 200 ^{14}C years of any other date in that respective bin. Therefore Pacopampa.1 comprises samples 1207 and 1206, Pacopampa.2 comprises sample 1205, Carrizal.1 comprises samples 1196 and 1195 and 1194 and 1193, and Carrizal.2 comprises sample 1192:

```
CalArray <- makeCalArray(shcal20, calrange = c(2000,6000))
x <- phaseCalibrator(data, CalArray)
plotPD(x)
```



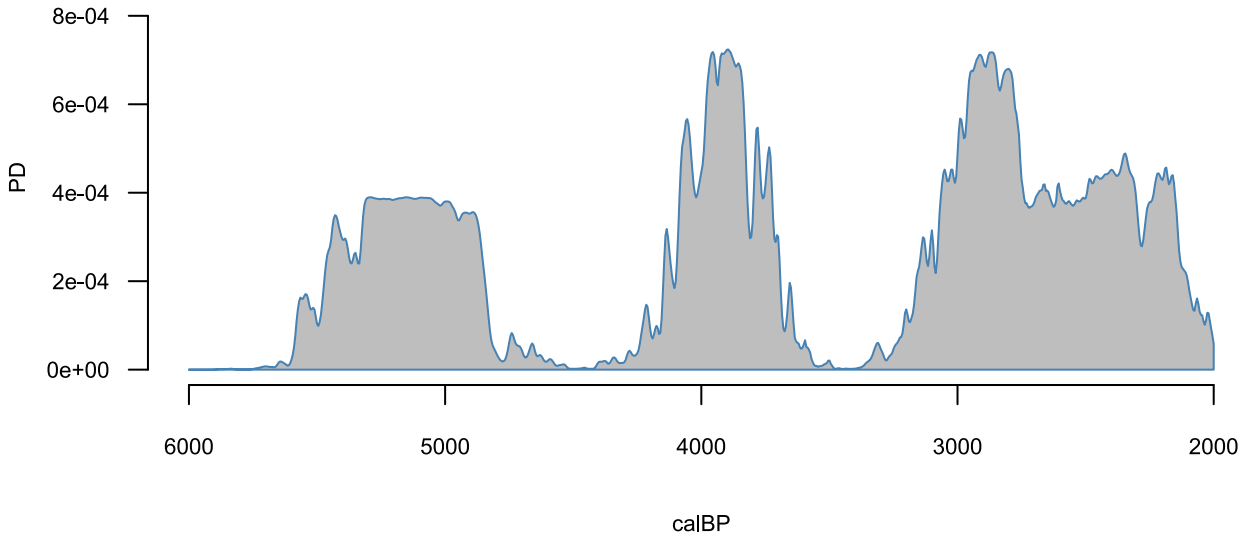
Finally, the distributions in each phase can be summed and normalised to unity. It is straight forward to achieve this directly from the dataframe created above:

```
SPD <- as.data.frame(rowSums(x))

# normalise
SPD <- SPD/(sum(SPD)*CalArray$inc)
```

Alternatively, the wrapper function summedPhaseCalibrator() will perform this entire workflow internally:

```
SPD <- summedPhaseCalibrator(data, shcal20, c(2000,6000))
plotPD(SPD)
```



Part 2

Continuous Piecewise Linear (CPL) Modelling

A CPL model lends itself well to the objectives of identifying specific demographic events. Its parameters are the (x,y) coordinates of the hinge points, which are the relative population size (y) and timing (x) of these events. Crucially, this package calculates model likelihoods (the probability of the data given some proposed parameter combination). This likelihood is used in a search algorithm to find the maximum likelihood parameters; to compare models with different numbers parameters to find the best fit without overfitting; in Monte-Carlo Markov Chain (MCMC) analysis to estimate credible intervals of those parameters; and in a goodness-of-fit test to check that the data is a typical realisation of the maximum likelihood model and its parameters.

Calculating likelihoods

Theoretically a calibrated date should be a continuous Probability Density Function (PDF), however in practice a date is represented as a discrete vector of probabilities corresponding to each calendar year, and therefore is a Probability Mass Function (PMF). Because calibration curves have a resolution of 5 years at best, this package generates a calibrated PMF with a default resolution of 5 years, although this can be adjusted to any preferred resolution (see makeCalArray()). This discretisation provides the advantage that numerical methods can be used to easily calculate likelihoods, provided the model is also discretised to the same time points.

A toy() model and population is provided to demonstrate how this achieved. Notice the toy population covers a slightly wider date range to ensure simulated ^{14}C dates are well represented around the edges. First, we simulate a plausible ^{14}C dataset and calibrate it:

```
# Simulate 99 archaeological samples in proportion to the toy population dynamics
pop.years <- as.numeric(row.names(toy$population))
mod.years <- as.numeric(row.names(toy$model))
set.seed(12345)
cal <- sample(x = pop.years, size = 99, replace = TRUE, prob = toy$population$individuals )

# Convert to 14C dates. 14C errors are borrowed from the real dataset 'arid' for realism.
age <- uncalibrateCalendarDates(cal, shcal20)
sd <- sample(x = arid$sd, size = 99, replace = TRUE)
data <- data.frame(age = age, sd = sd, phase = 1:99, datingType = 'C14')

# Calibrate each phase, taking care to restrict to the modelled date range
CalArray <- makeCalArray(shcal20, calrange = range(mod.years), inc = 1)
PD <- phaseCalibrator(data, CalArray)
```

Next we remove any calibrated dates with less than 50% of their PD within the modelled date range. This is a crucial step to avoid mischievous edge effects of dates outside the date range. Similarly, notice we constrained the CalArray to the modelled date range. These are important to ensure that we only model the population across a range that is well represented by data. To extend the model beyond the range of available data would be to assume the absence of evidence means evidence of absence. No doubt there may be occasions when this is reasonable (for example if modelling the first colonisation of an island that has been well excavated, and the period before arrival is evidenced by the absence of datable material), but more often the range of representative data is due to research interest, and therefore the logic of only including dates with at least 50% of their PD within the date range is that their true dates are more likely to be internal (within the date range) than external:

```
i <- colSums(PD) >= (0.5 / CalArray$inc)
PD <- PD[,i]
```

Finally we calculate the overall log likelihood of the model given the data:

```
loglik(PD, toy$model)
```

```
## [1] -657.7308
```

For comparison, we can calculate the overall likelihood of a uniform model given exactly the same data. Intuitively this should have a lower likelihood, since our dataset was randomly generated from the non-uniform toy population history:

```
uniform.model <- toy$model
uniform.model$pdf <- 1/nrow(uniform.model)
loglik(PD, uniform.model)
```

```
## [1] -673.518
```

And indeed the toy model is almost 5 million times more likely than the uniform model:

```
exp( loglik(PD, toy$model) - loglik(PD, uniform.model) )
```

```
## [1] 7182925
```

The anatomy of a CPL model

Having established how to calculate the likelihood of any proposed model given a dataset, we can use any out-of-the-box search algorithm to find the maximum likelihood model. This first requires us to describe the PDF of any population model in terms of a small number of parameters, rather than a vector of probabilities for each year.

We achieve this using the Continuous Piecewise Linear (CPL) model, which can be described using the (x,y) coordinates of its hinge points.

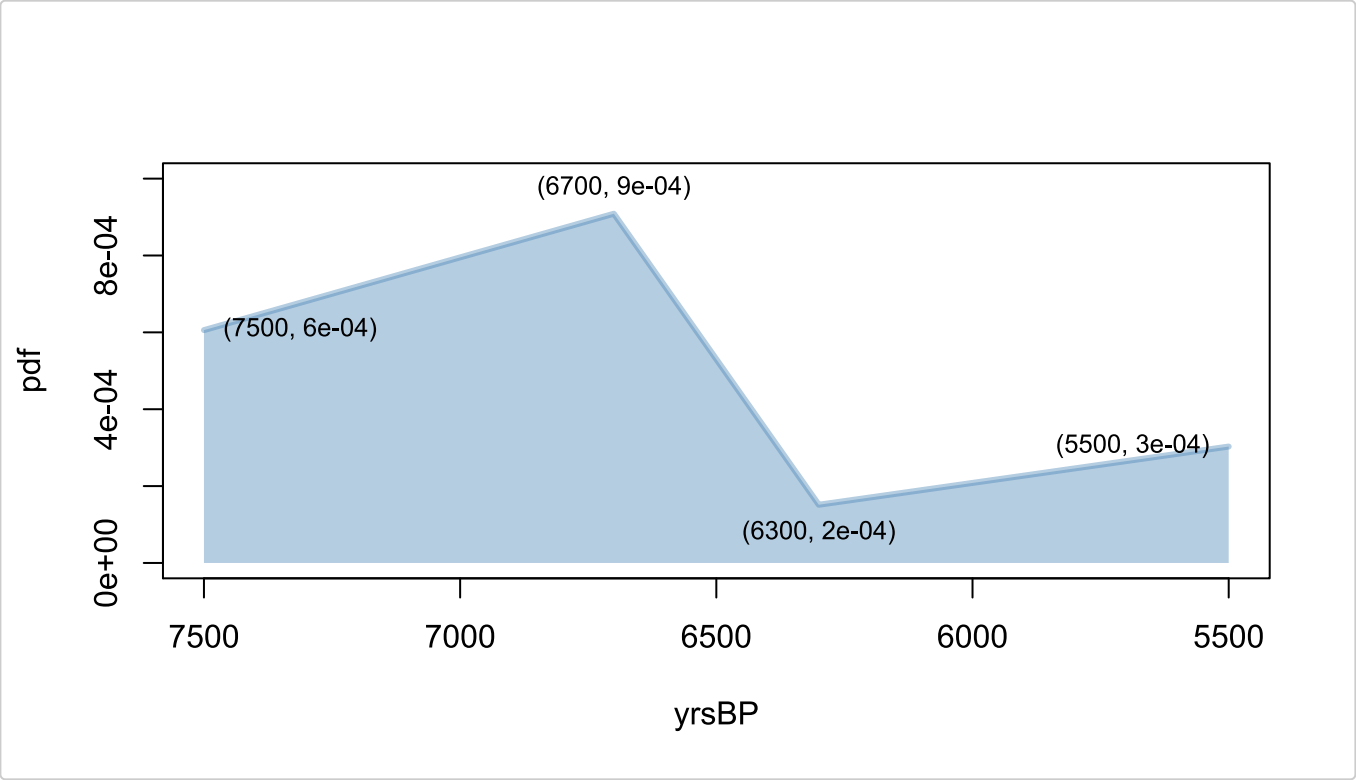


Illustration of the toy 3-CPL model PDF, described using just four coordinate pairs (hinges).

When performing a search for the best 3-CPL model coordinates (given a dataset), only five of these eight values are free parameters. The x-coordinates of the start and end (5500 BP and 7500 BP) are fixed by the choice of date range. Additionally, one of the y-coordinates must be constrained by the other parameters, since the total area of this PDF must equal 1. As a result, an n-CPL model will have 2n-1 free parameters.

Parameter space: The Area Breaking Process

We use the function convertPars() to map our search parameters to their corresponding PDF coordinates. This allows us to propose independent parameter values from a uniform distribution between 0 and 1, and convert them into coordinates that describe a corresponding CPL model PDF. This parameter-to-coordinate mapping is achieved using a modified stick breaking Dirichlet process.

The Dirichlet Process (not to be confused with the Dirichlet distribution) is an algorithm that can break a stick (the x-axis date range) into a desired number of pieces, ensuring all lengths are sampled evenly. The length (proportion) of remaining stick to break (at each iteration) is chosen by sampling from the Beta distribution, such that we use the Beta CDF (with α = 1 and β = the number of pieces still to be broken) to convert an x-parameter into its equivalent x-coordinate value. We extend this algorithm into two-dimensions by also sampling from a Gamma distribution on the y-axis, such that we use the Gamma CDF (with α = 1 and β = 1) to convert a y-parameter (between 0 and 1) into its equivalent coordinate value between 0 and $+\infty$.

The algorithm proceeds as follows:

1. At the first hinge (H1, x = 5500 BP) convert the first y-parameter into the H1 y-coordinate using the Gamma CDF.
2. At the next hinge (H2) convert the second y-parameter into the H2 y-coordinate using the Gamma CDF.
3. Use the above y-coordinates (y_{H1} and y_{H2}) to calculate the maximum possible H2 x-coordinate. This is achieved by assuming the PDF area so far (to the left of H2) equals 1. Or in other words by assuming the remaining hinges all have a y-coordinate equal to zero and the x-coordinate of H3 is infinitely close to H2. Convert the first x-parameter into x_{H2} by multiplying the Beta CDF by $\max(x_{H2})$.
4. Repeat steps 2 and 3 for subsequent hinges.

The algorithm continues until the penultimate hinge, which differs slightly since the maximum possible x-coordinate is constrained not just by the area so far (to the left) but also a minimum triangle area to the right. Therefore at the penultimate hinge we convert the last x-parameter first, by multiplying the Beta CDF by $\max(x_{Hpenultimate})$ Notice in the case of a 1-CPL model the first hinge is also the penultimate hinge, so in this special case the first y-parameter is converted into the H1 y-coordinate using a uniform distribution between 0 and 2.

Finally the coordinates of the last hinge are not free. This is because the x-coordinate is already defined as 7500 BP and the y-coordinate is calculated exactly using the other coordinates and the constraint that the total area equals 1.

The parameters must be provided single vector with an odd length, each between 0 and 1 (y,x,y,x,…y). For example, a randomly generated 6-CPL model will have 11 parameters and 7 hinges:

```
convertPars(pars = runif(11),years = 5500:7500)
```

```
##           x         y        area       stick     year          pdf
## 1 0.0000000 0.7009920 0.000000000 0.000000000 5500.000 0.0003503209
## 2 0.5152881 1.3132640 0.518961079 0.515288089 6530.576 0.0006563039
## 3 0.6350932 0.4001463 0.102637639 0.119805088 6770.186 0.0001999732
## 4 0.7772578 1.7175753 0.150532590 0.142164662 7054.516 0.0008583585
## 5 0.8303171 2.1613562 0.102906670 0.053059287 7160.634 0.0010801380
## 6 0.8315205 1.2476311 0.002051169 0.001203389 7163.041 0.0006235038
## 7 1.0000000 0.2114291 0.122910854 0.168479485 7500.000 0.0001056617
```

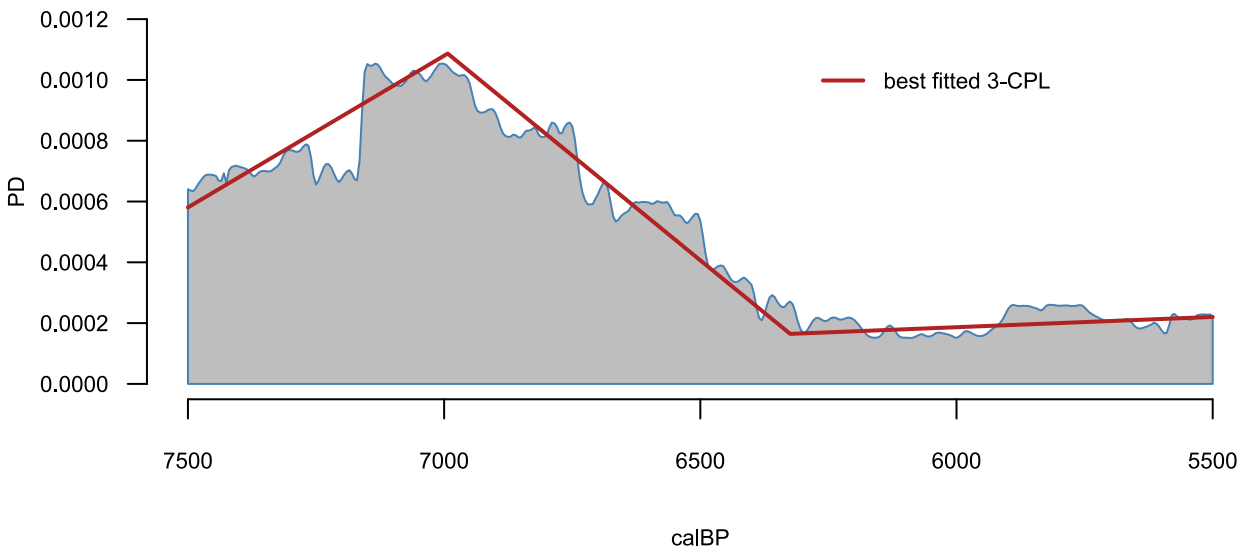
Note: The Area Breaking Algorithm is a heuristic that performs well and ensures the maximum likelihood parameters are always found, however its mapping of random parameters to PDF coordinates is not perfectly even, and we welcome ideas on improvements.

Maximum Likelihood parameter search

Any preferred search algorithm can be used. For example, the JDEoptim function from DEoptimR uses a differential evolution optimisation algorithm that performs very nicely for this application. We recommend increasing the default NP parameter to 20 times the number of parameters:

```
library(DEoptimR)
best <- JDEoptim(lower = rep(0,5),
    upper = rep(1,5),
    fn = objectiveFunction,
    PDarray = PD,
    type = 'CPL',
    NP = 100)
```

```
years <- as.numeric(row.names(PD))
CPL <- convertPars(best$par,years)
SPD <- summedPhaseCalibrator(data, shcal20, c(5500,7500))
plotPD(SPD)
lines(CPL$year, CPL$pdf, lwd=2, col='firebrick')
legend(x=6300, y=max(CPL$pdf), cex=0.7, lwd=2, col='firebrick', bty='n', legend= 'best fitted 3-
        CPL')
```



Credible interval parameter search using MCMC

Full workflow example

This section provides some full workflow examples that bring together several functions described so far, to generate a complete CPL model analysis.

Example 1:

Part 3

SPD simulation testing

Quote some text from paper about why this isnt a very sophisticated approach. Explain key improvement that the summary statistic is the likelihood.