Introduction to R for the Geoscience: Stratigraphic & Palaeo Data

Gavin Simpson

30th April — 3rd May, 2013

1 Rates of Change

In this part of the practical you will compute some rate of change estimates for a short core sequence from the Round Loch of Glenhead covering the last 140 years. There aren't any canned functions for computing rates of change in R so you'll perform all the neccessary steps yourself — don't worry, they aren't too onerous and you'll learn more about R coding as you go along.

Start by loading the rioja package and the dataset needed

```
> require("rioja")
> ## load data and extract
> data(RLGH)
> spp <- RLGH$spec
> age <- RLGH$depth$Age
```

1.1 Ordination-based rates of change

The first method you'll use is the ordination based rate of change, for which we need to smooth and interpolate the species data to a common time interval. This can be done using the `interp.dataset()` function

```
> ## interpolate new dataset to every 5 years
> ## using a smoothing splint
> x.new <- seq(0, max(age), by=5)
> sp.interp <- interp.dataset(y=spp, x=age, xout=x.new,
+                             method = "sspline")
> rownames(sp.interp) <- x.new
```

Next you need to ordinate the data. For consistency with the lecture example, use DCA, but you could try other ordination techniques should you wish.

```
> ## ordinate
> ord <- decorana(sp.interp)
> ord

Call:
decorana(veg = sp.interp)

Detrended correspondence analysis with 26 segments.
Rescaling of axes with 4 iterations.

                  DCA1    DCA2     DCA3     DCA4
Eigenvalues     0.1312 0.02112 0.007314 0.007082
Decorana values 0.1316 0.01130 0.005090 0.003085
Axis lengths    1.1056 0.53068 0.427959 0.374999
```

```
> ## plot of points showing ages
> plot(ord, display = "sites", type = "text")
```

How many axes do you want to retain?

Finally, to compute the rate of change we need to extract the axis scores we need and compute the distance between observations in ordination space. This has to be done by hand for the most part.

```
> ## get scores
> scrs <- scores(ord, display = "sites", choices = 1:2)
> roc <- as.matrix(dist(scrs))
> roc <- roc[row(roc) == col(roc) + 1] ## extract off-diagonal
> roc
```

```
 [1] 0.145791683 0.038388466 0.056476075 0.096765764 0.141675748 0.089316724
 [7] 0.148886370 0.065164981 0.123568620 0.180866597 0.135311608 0.072936429
[13] 0.009012417 0.054150418 0.058022625 0.065137156 0.081900194 0.093647074
[19] 0.102022525 0.106469426 0.102384652 0.095557955 0.078070381 0.054041243
[25] 0.024817872 0.004879166 0.010482557 0.017234419
```

Plot the rates of change for the core sequence

```
> ## plot roc against age
> plot(roc ~ head(x.new, -1), type = "l", ylab = "Rate of Change", xlab = "Age")
> points(roc ~ head(x.new, -1), type = "h")
```

Identify the periods with greatest rates of change. You'll compare these with those identified using the dissimilarity-based approach in the next section.

1.2 Dissimilarity-based rates of change

Now you'll calculate rates of change for the same samples using the raw dissimilarity method. In this method the data don't need to be interpolate, instead we work with the actual species data themselves. Start by computing the squared chord distances between all samples and extract the off-diagonal elements (the pairwise dissimilarities)

```
> roc2 <- as.matrix(paldist(spp/100, dist.method="sq.chord"))
> roc2 <- roc2[row(roc2) == col(roc2) + 1] ## extract off-diagonal
```

Next scale these by the number of years between each sample. We can compute the sample intervals in years using the `diff()` function

```
> intervals <- diff(age)
> roc2 <- roc2 / intervals
```

Plot the rates of change for the core sequence

```
> ## plot roc2 against age
> plot(roc2 ~ head(age, -1), type = "l", ylab = "Rate of Change", xlab = "Age")
> points(roc2 ~ head(age, -1), type = "h")
```

Identify the periods with greatest rates of change. How do these compare with the rates extract using the ordination-based approach? Can you think why there might be differences? Which method would you trust most? Why?

To aid you in your comparison, you can plot both rates of change on the same device

```
> ## compare plots
> layout(matrix(1:2, nrow = 2))
> ## plot roc against new age
> xlim <- range(age, x.new)
> plot(roc ~ head(x.new, -1), type = "l", ylab = "Rate of Change", xlab = "Age",
+      main = "Ordination-based", xlim = xlim)
```

```
> points(roc ~ head(x.new, -1), type = "h")
> ## plot roc2 against age
> plot(roc2 ~ head(age, -1), type = "l", ylab = "Rate of Change", xlab = "Age",
+      main = "Dissimilarity-based", xlim = xlim)
> points(roc2 ~ head(age, -1), type = "h")
> layout(1)
```

2 Transfer functions

We need to load the analogue package to start:

```
> library("analogue")
```

The data we will use today are:

1. Diatom counts and lake water pH from the Surface Waters Acidification Project diatom:pH training set
2. Diatom counts from a sediment core from the Round Loch of Glenhead (RLGH)

The data are available in analogue and are loaded with the `data()` function

```
> data(swapdiat, swappH, rlgh, package = "analogue")
```

3 MAT transfer functions

MAT transfer functions are a k-NN method and work by determining how similar samples are to one another and then use the k-most similar samples to a test sample to infer a unobserved property for the test sample. To do this we need to calculate dissimilarities between out training set samples and our test samples, and to do this, the training set and test set need to have the same set of species(variables). So ensure this is the case, we merge the two diatom datasets using `join()`

```
> dat <- join(swapdiat, rlgh, verbose = TRUE)
```

```
Summary:

            Rows Cols
Data set 1:  167  277
Data set 2:  101  139
Merged:      268  277
```

As the data are percentages we convert these to proportions as we subset the merged data set, by dividing the values by 100

```
> train <- dat$swapdiat / 100
> fossil <- dat$rlgh / 100
```

MAT models are fitted using the `mat()` function, which has a formula interface as well as a more standard interface. We use the formula interface and fit a MAT model to the SWAP data

```
> swap.mat <- mat(swappH ~ ., data = train, method = "chord")
```

A summary of the fitted model is obtained

```
> swap.mat
```

```
        Modern Analogue Technique

Call:
mat(formula = swappH ~ ., data = train, method = "chord")

Percentiles of the dissimilarities for the training set:

   1%    2%    5%   10%   20%
0.645 0.690 0.758 0.817 0.903

Inferences based on the mean of k-closest analogues:

  k   RMSEP      R2 Avg Bias Max Bias
  1  0.4227  0.7139  -0.0254  -0.3973
  2  0.3741  0.7702  -0.0493  -0.4689
  3  0.3387  0.8088  -0.0379  -0.4034
  4  0.3282  0.8200  -0.0335  -0.4438
  5  0.3136  0.8356  -0.0287  -0.4124
  6  0.3072  0.8444  -0.0386  -0.4152
  7  0.3167  0.8364  -0.0481  -0.4179
  8  0.3065  0.8474  -0.0433  -0.4130
  9  0.3049  0.8495  -0.0436  -0.4111
 10  0.3015  0.8548  -0.0473  -0.4083

Inferences based on the weighted mean of k-closest analogues:

  k   RMSEP      R2 Avg Bias Max Bias
  1  0.4227  0.7139  -0.0254  -0.3973
  2  0.3724  0.7720  -0.0485  -0.4651
  3  0.3380  0.8096  -0.0382  -0.4064
  4  0.3276  0.8208  -0.0341  -0.4437
  5  0.3140  0.8353  -0.0292  -0.4166
  6  0.3074  0.8440  -0.0378  -0.4203
  7  0.3156  0.8372  -0.0466  -0.4214
  8  0.3056  0.8480  -0.0420  -0.4167
  9  0.3041  0.8499  -0.0422  -0.4156
 10  0.3009  0.8548  -0.0458  -0.4130
```

The main part of the output shows the performance statistics for models containing 1 to 10 analogues. There are results for the unweighted and weighted model. The weights are the inverse of the dissimilarity, $1/d_{jk}$. The RMSEP values shown are leave-one-out errors. A more detailed summary of the fitted model can be produced using the `summary` method

```
> summary(swap.mat)
```

The output above uses only $k = (1, \ldots, 10)$ analogues to save space. To find the value of k that gives the lowest RMSEP we use the `getK()` extractor function

```
> getK(swap.mat)

[1] 10
attr(,"auto")
[1] TRUE
attr(,"weighted")
[1] FALSE
```

The `auto` means this value was selected automatically. Also, this is the unweighted version of the model.

A `plot` method is also available, modelled on the base function `plot.lm()`.

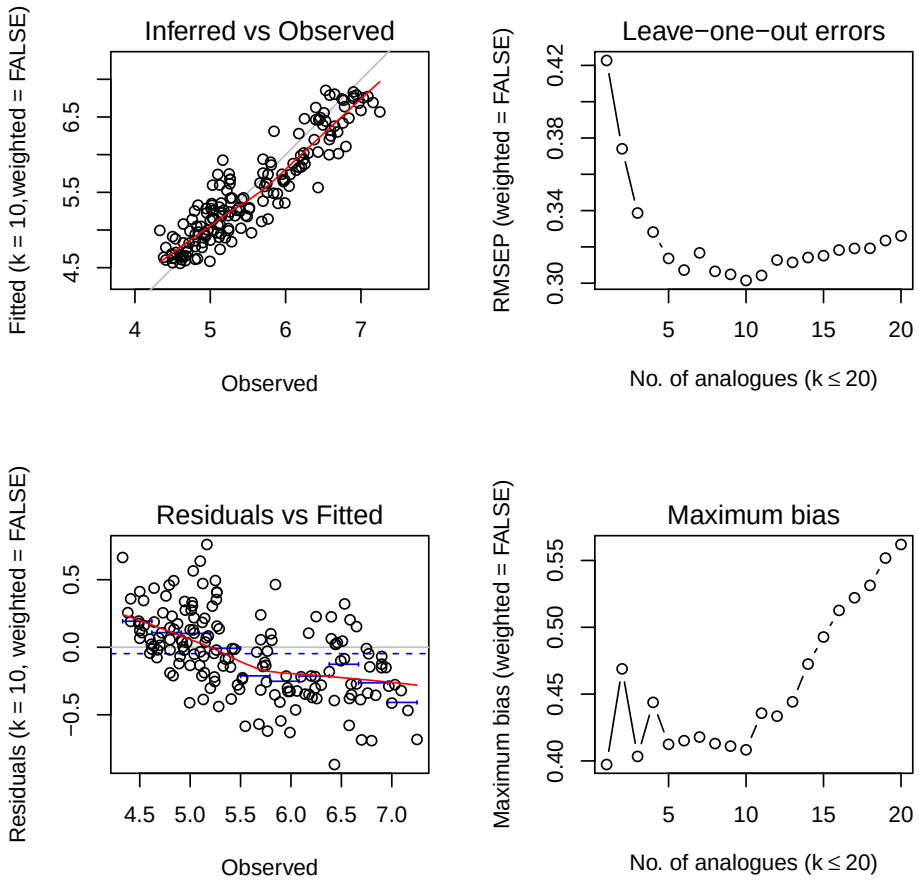

Figure 1: Summary diagram of the results of a MAT model applied to predict lake water pH from the SWAP diatom data set — see analogue paper for details.

```
> layout(matrix(1:4, ncol = 2))
> plot(swap.mat)
> layout(1)
```

The first and third lines in the snippet i) partition the plotting device into 4 (2 by 2) plotting regions, and ii) reset the defaults after drawing the plot.

To reconstruct pH values for the RLGH core we use the `predict` method

```
> rlgh.mat <- predict(swap.mat, fossil, k = 10)
> rlgh.mat

        Modern Analogue Technique predictions

Dissimilarity: chord
k-closest analogues: 10,        Chosen automatically? FALSE
Weighted mean: FALSE
Bootstrap estimates: FALSE

Model error estimates:
    RMSEP r.squared  avg.bias  max.bias
  0.30150   0.85478  -0.04729  -0.40833
```

```
Predicted values:
000.3 000.8 001.3 001.8 002.3 002.8 003.3 003.8 004.3 004.8 005.3 006.3 007.3
4.824 4.793 4.830 4.780 4.793 4.823 4.793 4.836 4.793 4.874 4.844 4.896 4.876
008.3 009.3 010.3 011.8 013.3 014.3 015.3 016.3 017.3 018.3 019.3 020.3 022.3
5.013 5.100 5.118 5.129 5.256 5.497 5.434 5.426 5.364 5.320 5.362 5.368 5.404
024.3 025.3 026.3 027.3 028.3 030.5 032.5 036.5 040.5 044.5 048.5 052.5 056.5
5.503 5.362 5.368 5.484 5.362 5.362 5.466 5.572 5.362 5.546 5.626 5.746 5.718
060.5 064.5 068.5 072.5 076.5 080.5 084.5 088.5 092.5 096.5 100.5 104.5 108.5
5.569 5.423 5.443 5.625 5.747 5.505 5.513 5.350 5.514 5.471 5.427 5.460 5.409
112.5 118.5 120.5 124.5 128.5 130.5 132.5 134.5 136.5 138.5 140.5 142.5 144.5
5.484 5.313 5.508 5.600 5.658 5.396 5.255 5.410 5.412 5.447 5.491 5.427 5.014
146.5 148.5 150.5 152.5 154.5 156.5 158.5 160.5 162.5 164.5 166.5 168.5 170.5
5.229 5.273 5.272 5.270 5.270 5.270 5.314 5.270 5.274 5.246 5.284 5.284 5.233
172.5 174.5 176.5 178.5 180.5 182.5 184.5 188.5 192.5 196.5 200.5 204.5 208.5
5.235 5.270 5.270 5.270 5.195 5.233 5.198 5.233 5.297 5.233 5.195 5.208 5.214
212.5 216.5 220.5 224.5 228.5 244.5 248.5 252.5 254.5 256.5
5.223 5.203 5.195 5.566 5.605 5.588 5.585 5.608 5.688 5.619
```

The `reconPlot` method can be used to plot the reconstructed values as a time series-like plot

```
> reconPlot(rlgh.mat, use.labels = TRUE, ylab = "pH", xlab = "Depth (cm.)")
```

The argument `use.labels = TRUE` instructs the function to take the names component of the predicted values as the values for the x-axis. Here depth is a surrogate for time.

Bootstrapping (or bagging) is a computer intensive method of producing point estimates and associated errors via resampling with replacement from the original training set. It is understood that the $\text{RMSEP}_{\text{boot}}$ is a more realistic estimate of the uncertainty in reconstructed values, and one way of determining sample-specific errors for fossil samples.

To bootstrap a MAT model, we use the `bootstrap` function, here drawing 100 bootstrap samples

```
> swap.boot <- bootstrap(swap.mat, n.boot = 100)
> swap.boot

        Bootstrap results for palaeoecological models

Model type: MAT
Weighted mean: FALSE
Number of bootstrap cycles: 100

Leave-one-out and bootstrap-derived error estimates:

           k  RMSEP     S1     S2 r.squared avg.bias max.bias
LOO       10 0.3015      -      -    0.8548 -0.04729  -0.4083
Bootstrap 12 0.3321 0.1206 0.3094    0.9219 -0.05550  -0.4545
```

Notice that the bootstrap model suggests that $k = 11$ analogues provides the lowest RMSEP. The quote RMSEP is a function of the s_1 and s_2 values shown

$$\text{RMSEP}_{\text{boot}} = \sqrt{s_1^2 + s_2^2}$$

A more traditional RMSEP value is returned by the `RMSEP` function

```
> RMSEP(swap.boot, type = "standard")

[1] 0.3094198
```

To produce sample specific errors for the RLGH core samples, we reuse the `predict` method, again drawing 100 bootstrap samples

```
> rlgh.boot <- predict(swap.mat, fossil, bootstrap = TRUE, n.boot = 100)
```

We can redraw our plot of reconstructed values using `reconPlot`, displaying the sample-specific errors via error bars

```
> reconPlot(rlgh.boot, use.labels = TRUE, ylab = "pH", xlab = "Depth (cm.)", display.error = "bars",
```

An alternative measure of reliability of reconstructed values is to determine if each fossil sample has good close modern analogues in the training set. A close modern analogue is defined as being within a distance x of a training set sample that is less than or equal to a low percentile of the distribution of pair-wise dissimilarity values from the training set. We calculate the minimum distance between a training set sample and each fossil sample using `minDC` and plot the results

```
> rlgh.mdc <- minDC(rlgh.mat)
> plot(rlgh.mdc, use.labels = TRUE, xlab = "Depth (cm.)")
```

There are several core samples whose minimum distance to a training set sample exceed the 5^{th} or even the 10^{th} percentile, indicating that reconstructions for these periods of the record may be less reliable.

4 Weighted Averaging transfer functions

WA transfer functions are fitted in much the same way as the MAT models of the previous section. The `wa()` function is used and again it has a formula and a standard interface. We use the formula interface to fit a WA model to the SWAP data, print the results and drawn some model diagnostics plots

```
> swap.wa <- wa(swappH ~ ., data = swapdiat, deshrink = "inverse")
> swap.wa

        Weighted Averaging Transfer Function

Call:
wa(formula = swappH ~ ., data = swapdiat, deshrink = "inverse")

Deshrinking  : Inverse
Tolerance DW : No
No. samples  : 167
No. species  : 277

Performance:
     RMSE  R-squared  Avg. Bias  Max. Bias
   0.2756     0.8717     0.0000    -0.1933
```

```
> opar <- par(mfrow = c(1,2))
> plot(swap.wa)
> par(opar)
```

Note the RMSE here is an apparent statistic and is a gross under-estimate of the true error as this value has been generated by testing the model with the same data used to fit the model.

The bootstrap RMSEP is a more realistic estimator of the true model error. Again we can use the `bootstrap()` function to perform the calculations

```
> swap.waboot <- bootstrap(swap.wa, n.boot = 100)
> swap.waboot
```

How does this RMSEP value compare to the MAT $\text{RMSEP}_{\text{boot}}$ calculated earlier?

WA reconstructed values can be produced using the `predict` method. Again we use bootstrapping to determine sample-specific errors and plot the reconstructed values and errors using `reconPlot()`

```
> rlgh.wapred <- predict(swap.wa, rlgh, CV = "bootstrap", n.boot = 100)
> rlgh.wapred
> reconPlot(rlgh.wapred, use.labels = TRUE, ylab = "pH", xlab = "Depth (cm.)")
```

5 Chronological clustering

In this section you'll use functionality in the rioja and mvpart packages to perform chronological clustering or zonation of the RLGH core sequece. Begin by loading rioja

```
> require("rioja")
```

`chclust()` implements the CONISS and CONSLINK methods, with CONISS being the default. To use `chclust()` a dissimilarity matrix is required. Load the RLGH core data and compute the chord distance between samples in the core

```
> data(RLGH)
> diss <- dist(sqrt(RLGH$spec / 100))  ## / 100 because data are %
```

Now use `diss` with `chclust()` to perform the clustering

```
> clust <- chclust(diss)
> clust
```

```
Call:
chclust(d = diss)

Cluster method   : coniss
Distance         : euclidean
Number of objects: 20
```

The clustering can be visualised with the `plot()` method

```
> plot(clust)
```

The number of zones can be indicated via comparison with the broken stick distribution. In rioja this can be achieved via the `bstick()` function.

```
> bst <- bstick(clust, 10)
> bst
```

```
       nGroups dispersion    bstick
Stick1       2  0.6202329 0.5758494
Stick2       3  0.3627392 0.4135349
Stick3       4  0.1784789 0.3323777
Stick4       5  0.1748117 0.2782729
Stick5       6  0.1475450 0.2376943
Stick6       7  0.1438951 0.2052314
Stick7       8  0.1325117 0.1781790
Stick8       9  0.1247732 0.1549912
Stick9      10  0.1204636 0.1347019
```

This also produces a plot with the observed reduction in within-group sums of sqaures plotted in red and the broken stick distribution in red. The `10` in the call limits the number of groups up to which the analysis is performed.

How many groups are suggested by the analysis?

6 Principal Curves

In this section of the practical you'll explore fitting a principal curve to the Abernethy Forest pollen data set. The functions you need are available in the analogue package. Load it, and ignore the warnings for now.

```
> require(analogue)
> data(abernethy)
```

A simple summary of the data can be produced using the `Stratiplot()` function

```
> (plt <- Stratiplot(Age ~ . - Depth, data =
+                    chooseTaxa(abernethy, max.abun = 15, n.occ = 10),
+                    type = c("h","g","l"), sort = "wa"))
```

Next, remove two variables from the data set (the samples depths and ages) and then proceed to fit a PCA and a CA, as well as the principal curve to the data

```
> abernethy2 <- abernethy[, -(37:38)]
> aber.pca <- rda(abernethy2)
> aber.ca <- cca(abernethy2)
> aber.pc2 <- prcurve(abernethy2, method = "ca", trace = TRUE, plotit = TRUE,
+                     vary = TRUE, penalty = 1.4)

--------------------------------------------------------------------------------
Initial curve: d.sq: 103233.4502
Iteration   1: d.sq: 4283.4308
Iteration   2: d.sq: 4312.2976
Iteration   3: d.sq: 4340.6911
Iteration   4: d.sq: 4355.3876
Iteration   5: d.sq: 4366.4975
Iteration   6: d.sq: 4369.9444
--------------------------------------------------------------------------------
PC Converged in 6 iterations.
--------------------------------------------------------------------------------
```

The arguments to `prcurve()` tell it to start from a CA solution, show it working by printing to the console and plotting the curve as it converges, taxa are allowed differeing complexity smoothers and the penalty per degree of freedom is increased by 40% to help avoid over-fitting.

Look at the fitted object. How much of the variance in the dataset is explained by the principal curve?

```
> aber.pc2

        Principal Curve Fitting

Call: prcurve(X = abernethy2, method = "ca", vary = TRUE, trace = TRUE,
plotit = TRUE, penalty = 1.4)

Algorithm converged after 6 iterations

           SumSq Proportion
Total     103234      1.000
Explained  98864      0.958
Residual    4370      0.042

Fitted curve uses 218.3391 degrees of freedom.
```

Compare that with the variance explained by the PCA and CA

```
> varExpl(aber.pca)

      PC1
0.4649883

> varExpl(aber.ca)

      CA1
0.3098955
```

How well has the principal curve done compared to the more traditional techniques?

Plot the principal curve

```
> ## Plot the fitted curve
> plot(aber.pc2, abernethy2)
```

Can you see why the principal curve performs much better than the other two ordination techniques?

A longer chunk of code is required to draw rates of change and compare the various methods

```
> ## local variables for ease of reference
> Depth <- abernethy$Depth
> Age <- abernethy$Age
> ## split plot window into 3 chunks, fig 2 uses columns 2 & 3
> layout(matrix(c(1,2,2), ncol = 3))
> nseg <- nrow(abernethy2) - 1
> RoC <- abs(unclass(diff(gradientDist(aber.pc2))) / diff(Age))
> ## First plot - as rate of change
> plot(y = Age[-length(Age)],
+      x = RoC * 1000, type = "n",
+      ylim = rev(range(Age)),
+      ylab = "Age (Radiocarbon years BP)",
+      xlab  = expression("Rate of Change" ~ (kyr^{-1})),
+      main = "a", cex.main = 1.5)
> segments(x0 = rep(0, nseg), y0 = Age[-length(Age)],
+          x1 = RoC * 1000, y1 = Age[-length(Age)])
> ## plot comparison figure
> plot(gradientDist(aber.pc2), orderBy = Age,
+      xlim = rev(range(Age)),
+      type = "o", flipAxes = TRUE, xlab = "Age (Radiocarbon years BP)",
+      main = "b",
+      cex = 0.8, pch = 21, col = "black", bg = "black", cex.main = 1.5)
> lines(gradientDist(aber.pca), orderBy = Age,
+       lty = "dashed", flipAxes = TRUE, type = "o",
+       cex = 0.8, pch = 22, col = "forestgreen", bg = "forestgreen")
> lines(1 - gradientDist(aber.ca), orderBy = Age,
+       lty = "dotdash", flipAxes = TRUE, type = "o",
+       cex = 0.8, pch = 23, col = "red", bg = "red")
> legend("topright", bty = "n", pch = 21:23,
+        legend = c(expression(PCurve), expression(PCA[1]), expression(CA[1])),
+        lty = c("solid","dashed","dotdash"),
+        col = c("black","forestgreen","red"),
+        pt.bg = c("black","forestgreen","red"), inset = 0.01, cex = 1.4,
+        seg.len = 4)
> layout(1)
```

The resulting plot is shown in Figure 2.

The individual curves fitted to each taxon can be visualised, again with a long chunk of code. Here, only the most abundant taxa are shown

```
> taxaWant <- tran(chooseTaxa(abernethy2, max.abun = 25, n.occ = 10),
+                  method = "pcent2prop")
> ylim <- c(0, max(apply(taxaWant, 2, max)))
> p <- with(aber.pc2, data.frame(gradient = seq(min(lambda), max(lambda),
+                                length = 1000)))
> interpCurve <- approxfun(with(aber.pc2, lambda[tag]), Age)
> p <- transform(p, AgeGradient = interpCurve(gradient))
> layout(matrix(1:NCOL(taxaWant), ncol = 3))
> op <- par(mar = c(2,1,4.1,2.1), oma = c(3.1,3.1,0,0), las = 1)
```

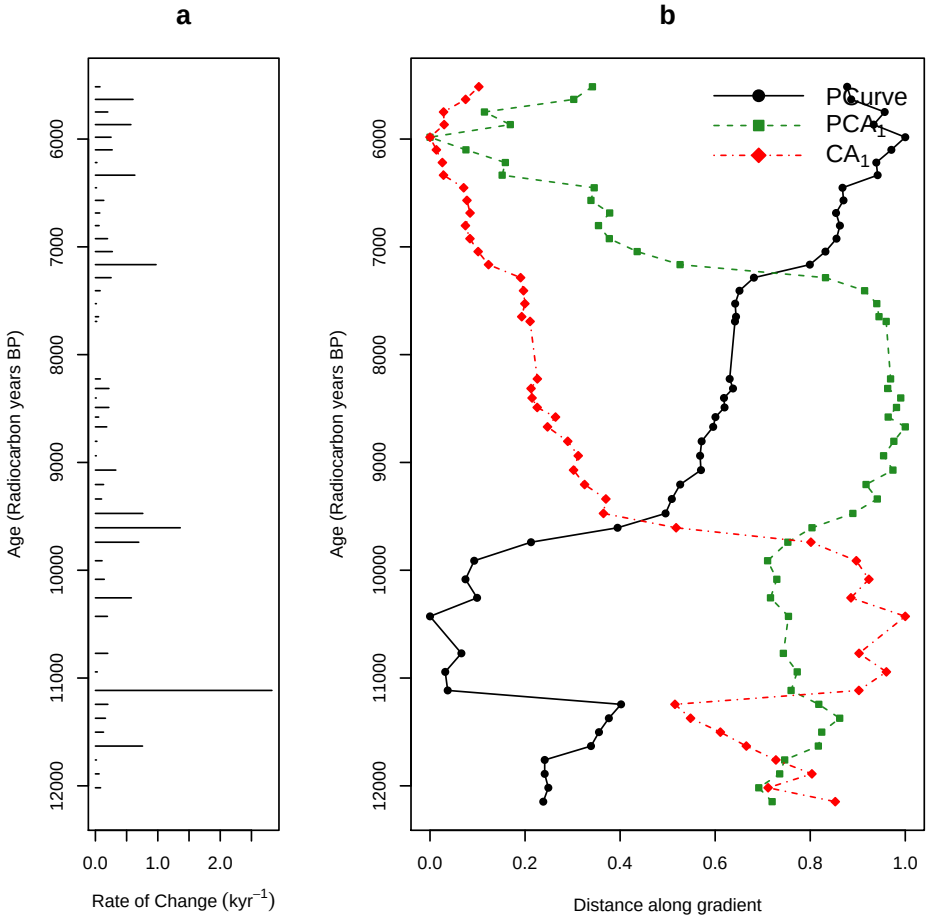

Figure 2: Comparison of methods summarising change in the Abernethy Forest sequence.

```
> for(i in seq_len(NCOL(taxaWant))) {
+     d <- with(aber.pc2, data.frame(gradient = lambda[tag],
+                                    abundance = taxaWant[,i],
+                                    AgeGradient = interpCurve(lambda[tag])))
+     mod <- with(d,
+                 smooth.spline(AgeGradient,##gradient,
+                               abundance,
+                               control.spar = list(low = 0.2),
+                               penalty = 1.4,
+                               df = aber.pc2$complexity[names(taxaWant)[i]]))
+     with(aber.pc2, plot(interpCurve(lambda[tag]), #lambda[tag],
+                         taxaWant[,i],
+                         main = colnames(taxaWant)[i],
+                         ylim = ylim, ylab = NA, xlab = NA,
+                         type = "n"))
+     fit <- pred <- predict(mod, p[,"AgeGradient"])$y##[,1]
+     with(aber.pc2, points(interpCurve(lambda[tag]), #lambda[tag],
+                                      taxaWant[,i]))
+     with(p, lines(AgeGradient, fit, lwd = 2, col = "black"))
+ }
> title(xlab = "Distance along the Principle Curve", outer = TRUE,
```

```
+       cex.lab = 1.3, line = 1)
> title(ylab = "Proportional Abundance", outer = TRUE,
+       cex.lab = 1.3, line = 2)
> layout(1)
> par(op)
```

The resulting plot is shown in Figure 3.

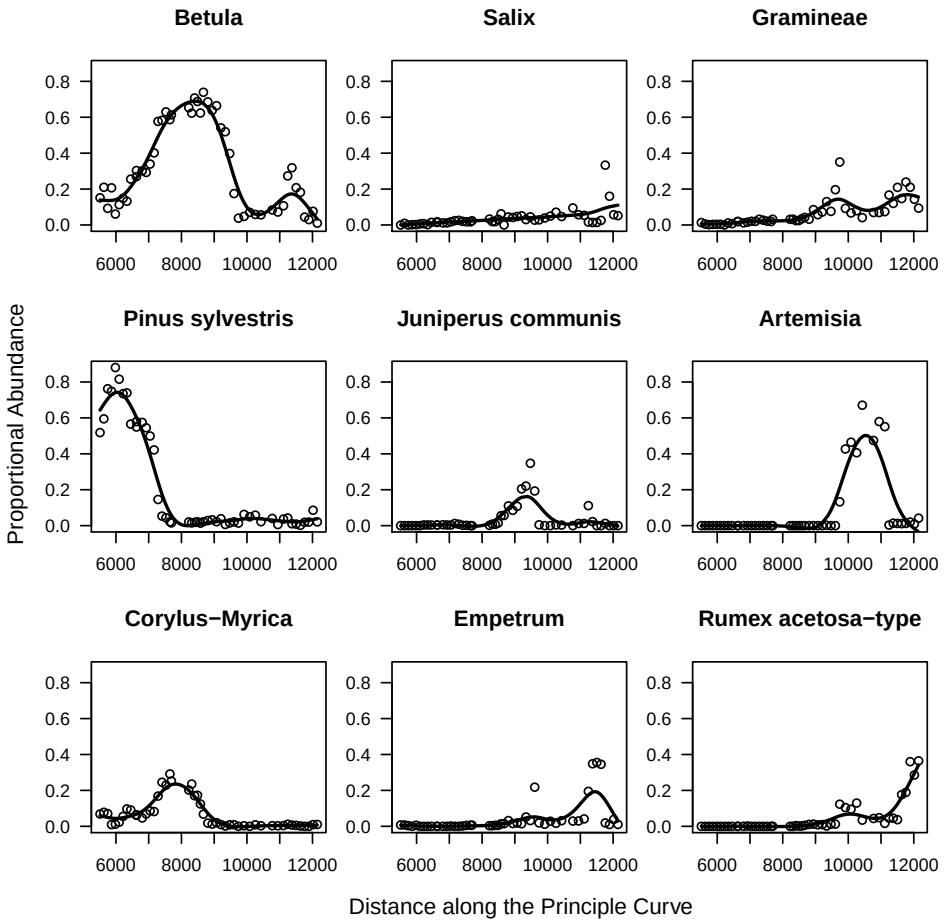

Figure 3: The individual splines fitted to each taxon. Only the most abundant species are shown.

Using the two plots, do you feel the principal curve has done a good job of describing floristic change in the Abernethy core?