Introduction to Functional Data Analysis

Mathematics for Big Data

Jeremy Williams

May 7, 2018

Contents

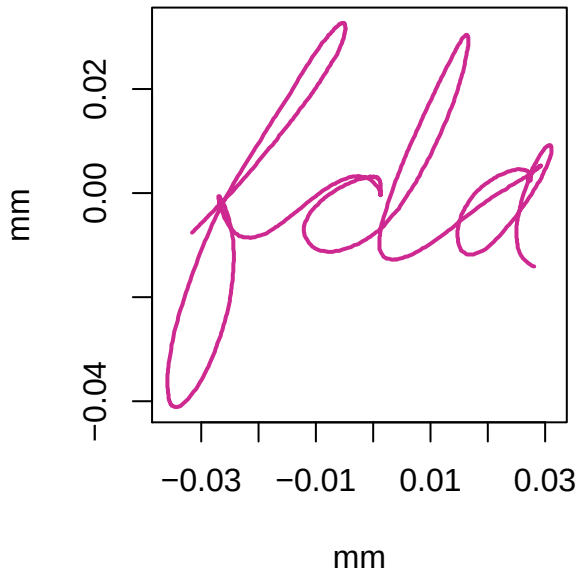
Introduction to Functional Data Analysis (Practical Work)

Practical 1

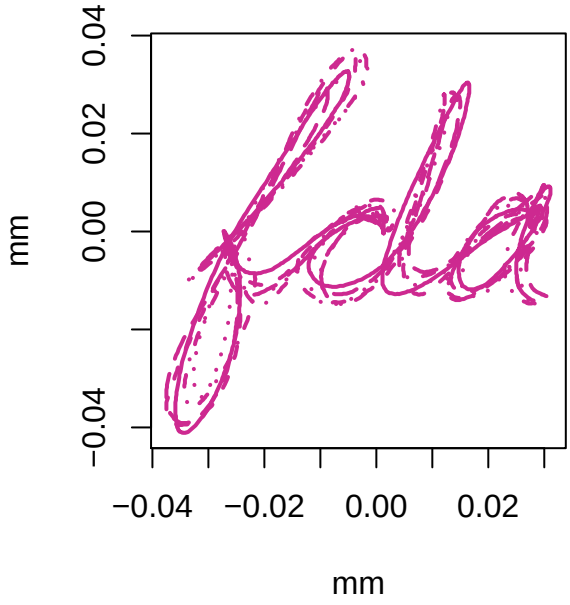
```
##### Adapted from Gilles Hooker
#####
#install.packages("fda");
suppressMessages(suppressWarnings(require(fda)))
#install.packages("fda.usc");
suppressMessages(suppressWarnings(require(fda.usc)))
## some plots
par(pty="s") # square plot

# handwritten data
data("handwrit")
?handwrit
```

```
## starting httpd help server ... done
```

```
plot(handwrit[, 1, 1], handwrit[, 1, 2], type="l", col="maroon3",xlab="mm",ylab="mm", lwd=2)
```




```
matplot(handwrit[, 1:5, 1], handwrit[, 1:5, 2], type="l", col="maroon3",xlab="mm",ylab="mm", lwd=2)
```




```
## take a look at the difference between plot and matplot


# Canadian Weather Data.
?CanadianWeather
data(CanadianWeather)
names(CanadianWeather)
```

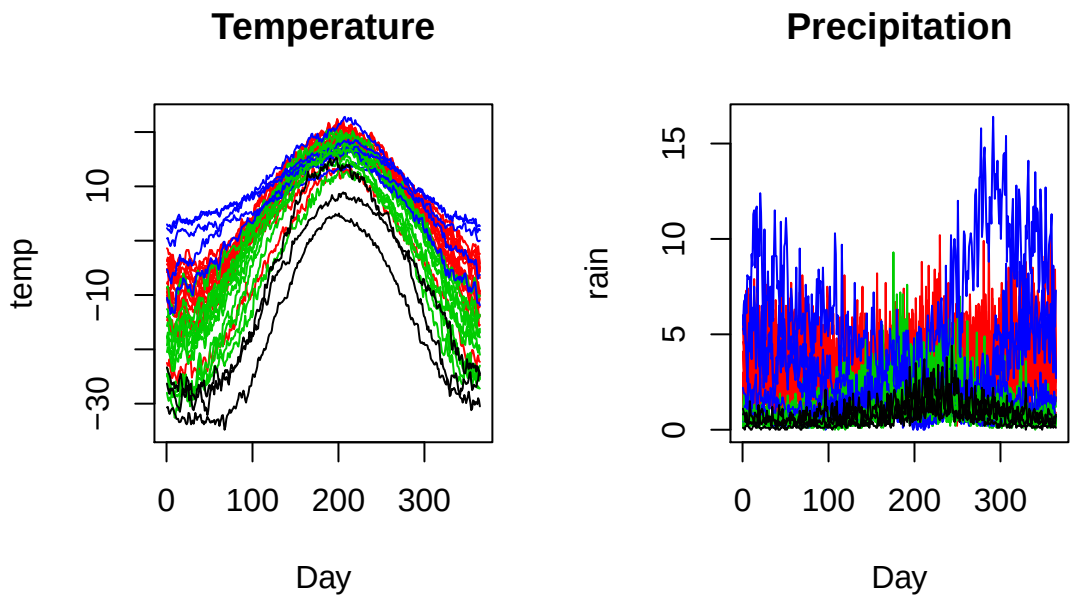
```
## [1] "dailyAv"       "place"         "province"      "coordinates"
## [5] "region"        "monthlyTemp"   "monthlyPrecip" "geogindex"
```

```
dim(CanadianWeather$dailyAv)
```

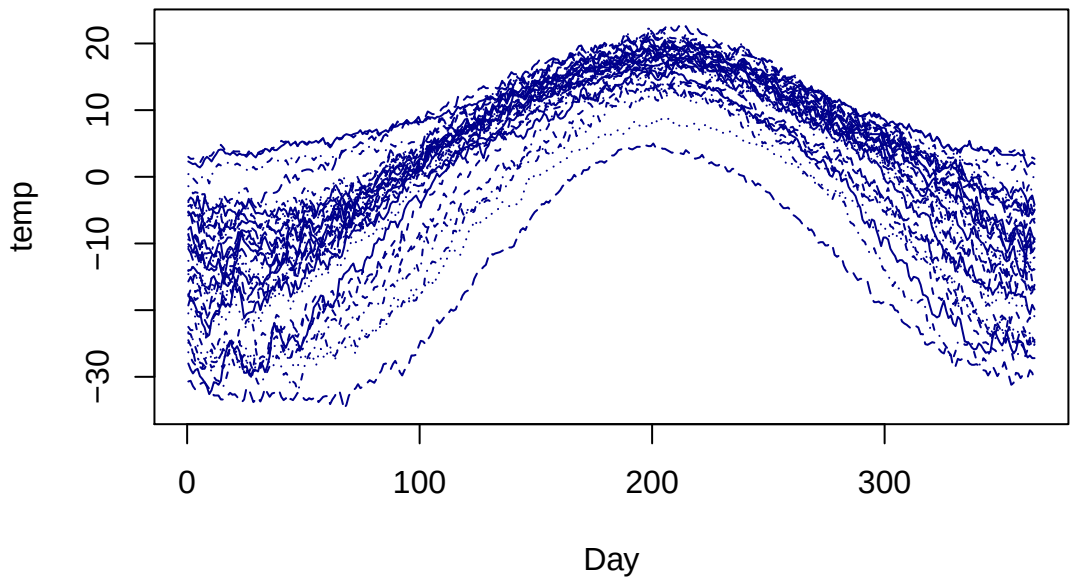
```
## [1] 365  35   3
```

```
# Temperature and precipitation are contained in the dailyAV element
#1: day of the year
#2: Monitoring Station
#3:Temperature.C Precipitation.mm      log10precip
temp = CanadianWeather$dailyAv[,,1]
precip = CanadianWeather$dailyAv[,,2]
# We need corresponding time points. Put them half-way through a day. This
# is because the period is over 0:365 and we'd like the 365 data points to be
# about symmetric in that period.

daytime = (1:365)-0.5

# This is a bit fine for plotting purposes, so we can also create a vector of
# points to use for plotting every 5 days.  Later to be used for b-spline basis.

day5 = seq(0,365,5)
```

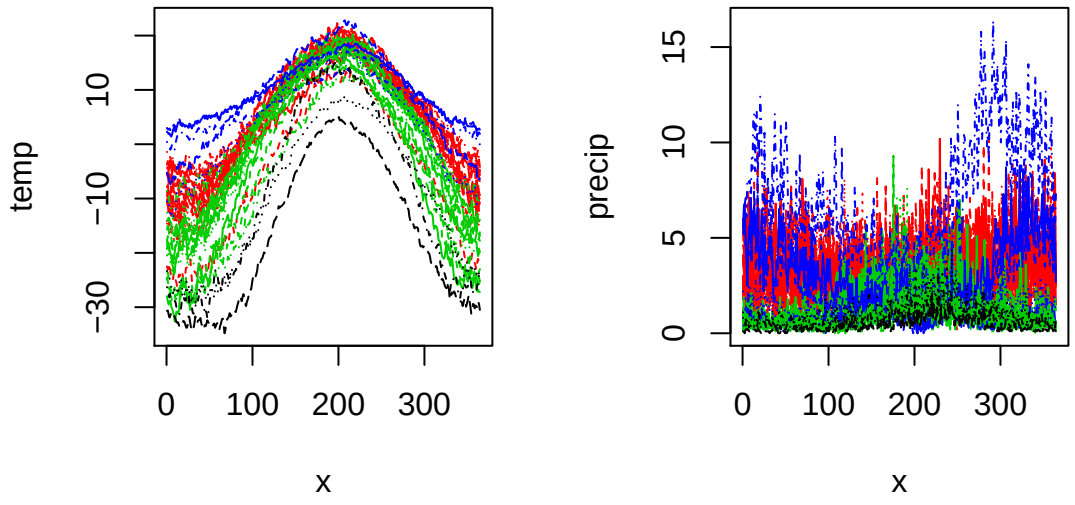
```r
### Some plots (for the course's notes)
par(mfrow=c(1,2))
par(pty="s")
matplot(daytime,temp,type='l', xlab="Day",ylab="temp", lty=1, main="Temperature",col=as.factor(Canadian
matplot(daytime,precip,type='l',xlab="Day",ylab="rain",lty=1, main= "Precipitation",col=as.factor(Canad
```




```r
par(mfrow=c(1,1));par(pty="m") #back to "normal" settings
matplot(daytime,temp,type='l', xlab="Day", col="darkblue")
```




```
# We can also plot by region; Atlantic, Pacific, Central and North.
par(mfrow=c(1,2))
par(pty="s")
matplot(daytime,temp,type='l',col=as.factor(CanadianWeather$region))
matplot(daytime,precip,type='l',col=as.factor(CanadianWeather$region))
```




```
# Fdata objects.  Beware, packages fda and fda.usc have slightly different syntaxis.

dim(CanadianWeather$dailyAv)
```

```
## [1] 365  35   3
```

```
######## The simplest Fdata object (with no smoothing)
tt=1:365
temp.fdata<-fdata(t(CanadianWeather$dailyAv[,,1]),tt) #from fda.usc , ask for help!
dim(temp.fdata) #  rows are functions as opposed to fda package
```

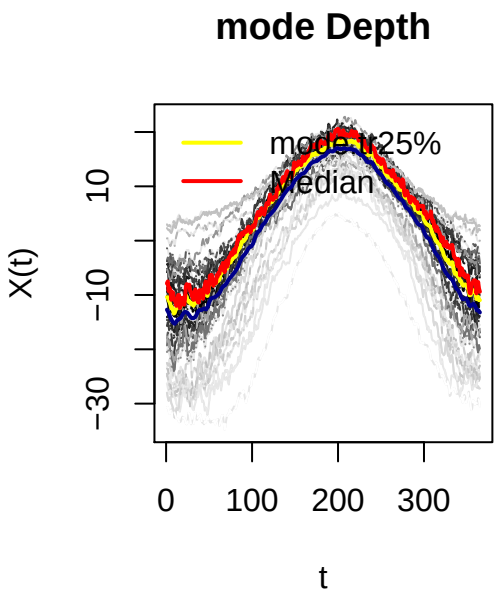
```
## [1]  35 365
```

```
class(temp.fdata)
```

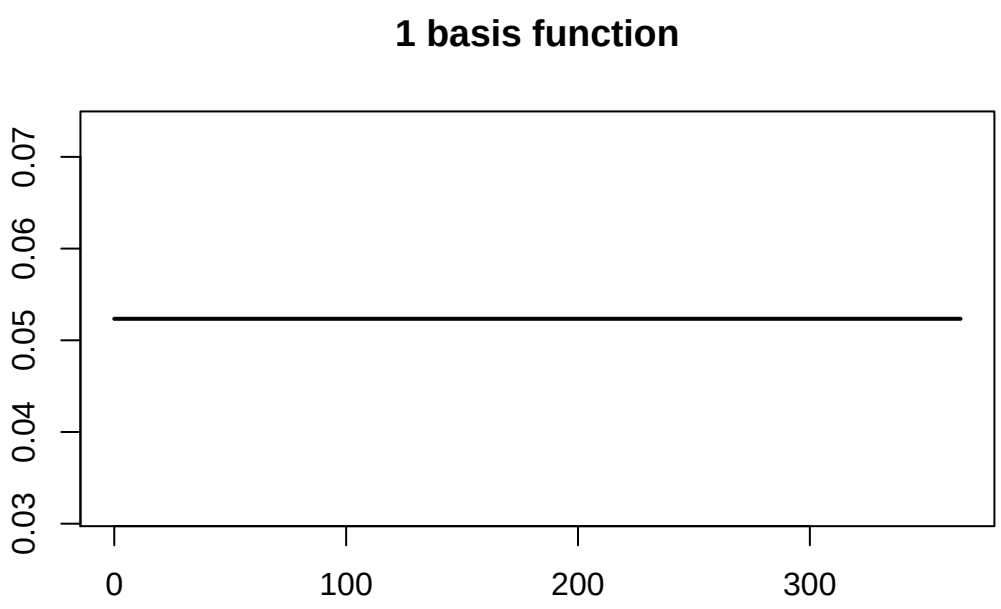
```
## [1] "fdata"
```

```
temp.fd=fdata2fd(temp.fdata)
depth.mode(temp.fdata,draw=T) # we'll talk about depths later
lines(mean.fd(temp.fd),col="darkblue",lwd=2)


################################################################################
###                            DEFINING A BASIS SYSTEM                       ####
################################################################################

# We'll always need the range (in fda.usc it is called rangeval)
# fda.usc mostly relyes on basis systems created by fda, see help
# for both libraries.

dayrng = c(0,365)
```

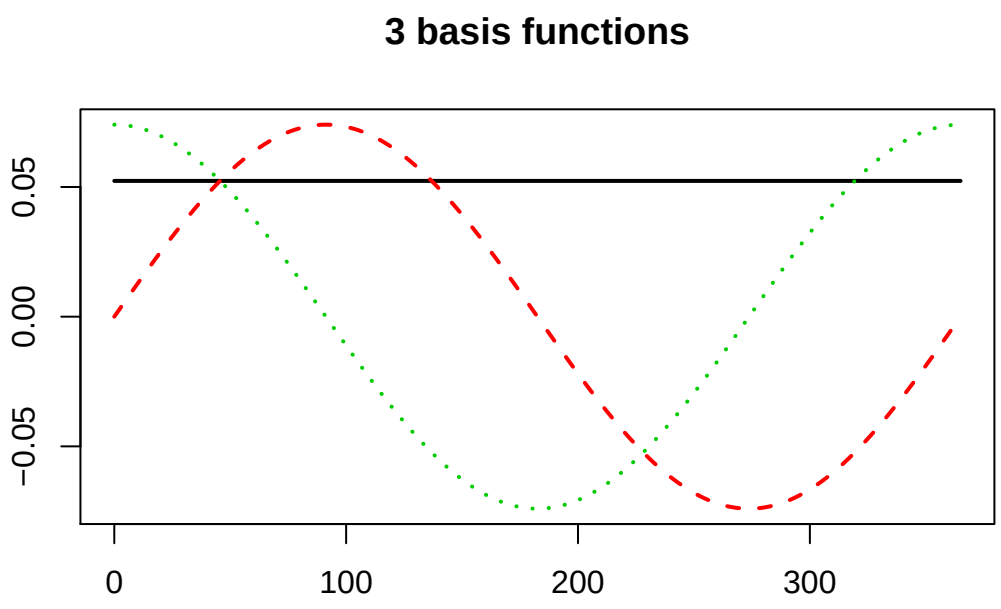
```
#### 1. Fourier Basis with 365 basis functions

fbasis = create.fourier.basis(dayrng,365) #from fda
par(mfrow=c(1,1))
```



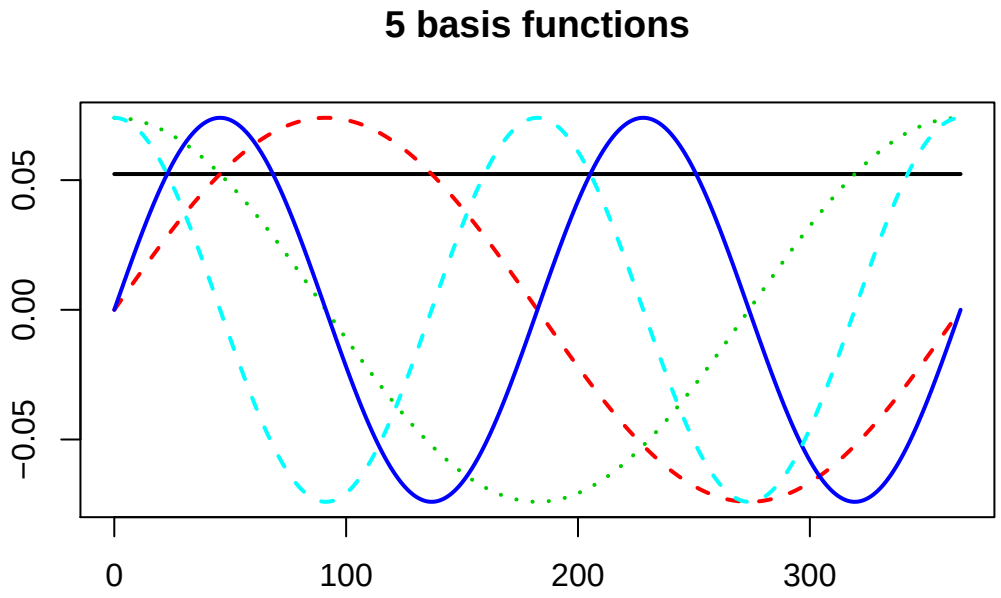

```
par(pty="m")
# Plot a basis with just 5 components  nbasis must be odd, otherwise one more is added
plot(create.fourier.basis(dayrng,1),lwd=2,main="1 basis function")
```



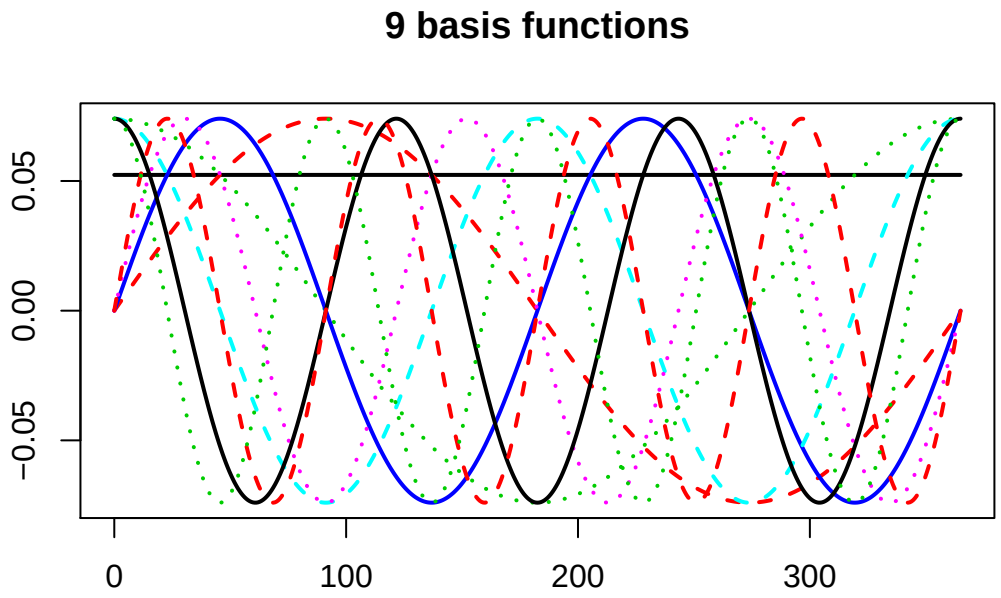

```
plot(create.fourier.basis(dayrng,3),lwd=2,main="3 basis functions",type="l")
```




```
plot(create.fourier.basis(dayrng,5),lwd=2,main="5 basis functions")#  sin, cos sin2w, cos2w
```




```
plot(create.fourier.basis(dayrng,9),lwd=2,main="9 basis functions")
```

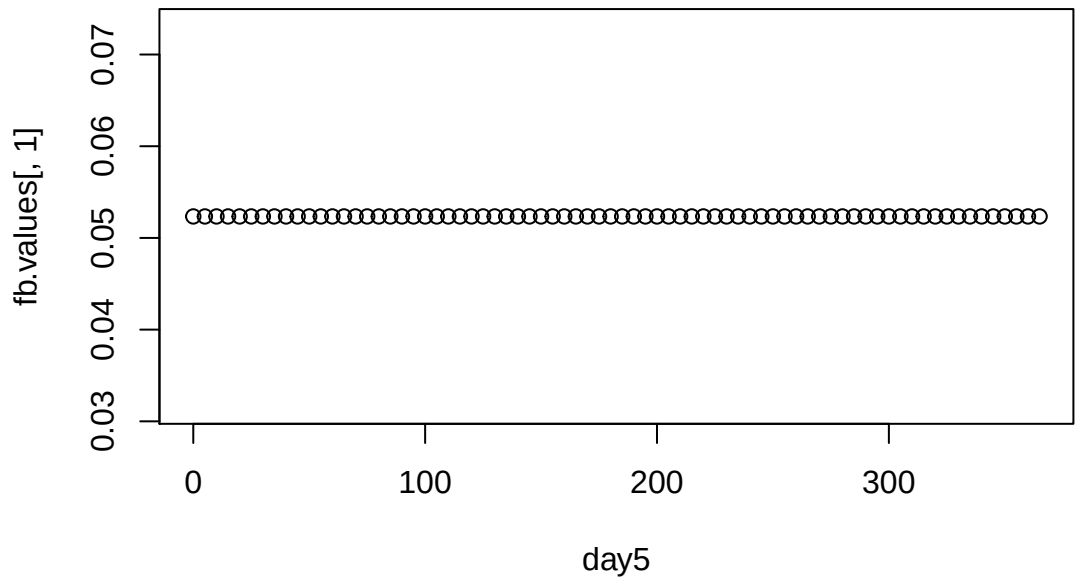



```
#

# Let's try a simple linear regression of the first temperature record on the
```

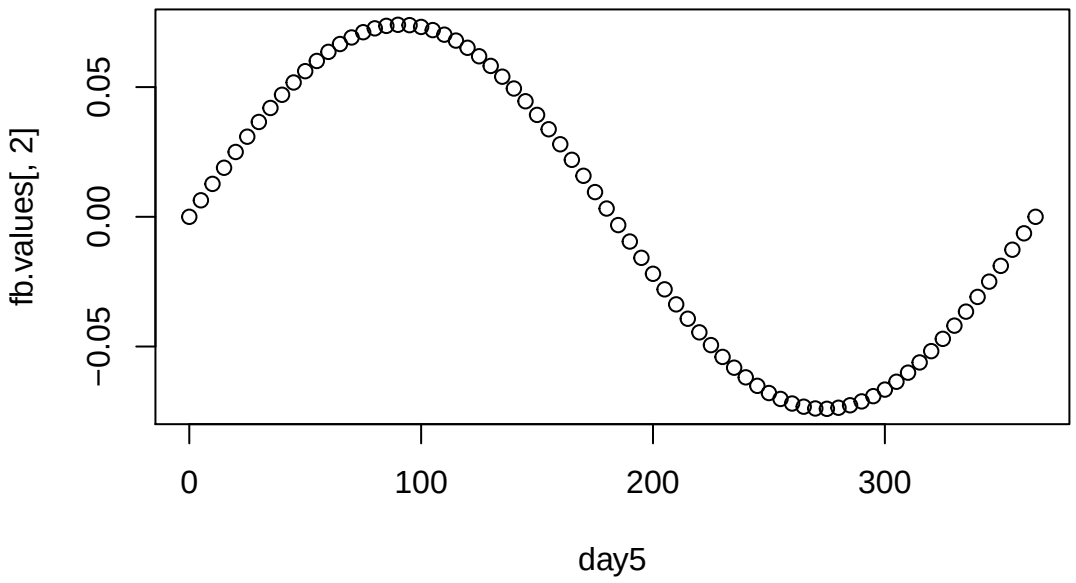
```r
# first few basis functions

fb.values = eval.basis(day5,fbasis)

# the 74 by 365 matrix that results has rows as days, columns as bases

dim(fb.values)
```

```
## [1]  74 365
```

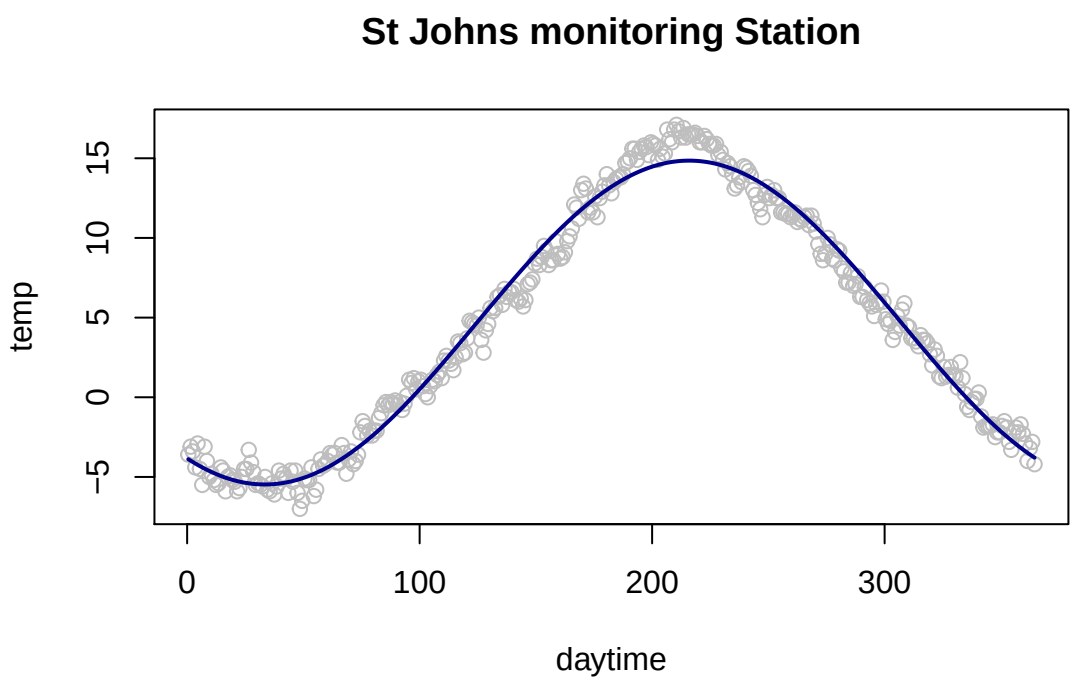
```r
plot(day5,fb.values[,1])
```



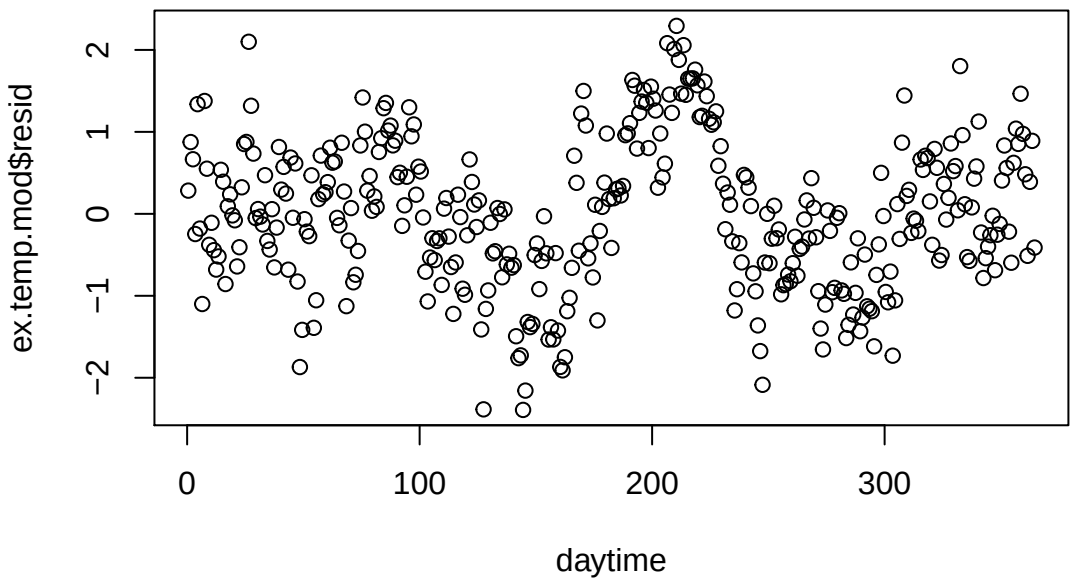

```r
plot(day5,fb.values[,2])
```




```
# Extract the first temperature record

ex.temp = temp[,1] #St Johns' station

# Run a linear regression on temperature with the first 5 basis functions. In
# this case, evaluate the basis at the observation times

Xmat = eval.basis(daytime,fbasis)
Xmat = Xmat[,1:5]    # First 5 basis functions
#Xmat=Xmat[,1:20]
Xmat=Xmat[,1:3] # First 3 basis functions
ex.temp.mod = lm(ex.temp~Xmat)

# Plot this; the fitted values returned by lm will represent the smooth
# well enough to plot.

plot(daytime,ex.temp,col="grey", main="St Johns monitoring Station",ylab="temp")
lines(daytime,ex.temp.mod$fitted,col="darkblue",lwd=2)
```




```
# We can also look at residuals

plot(daytime,ex.temp.mod$resid)
```



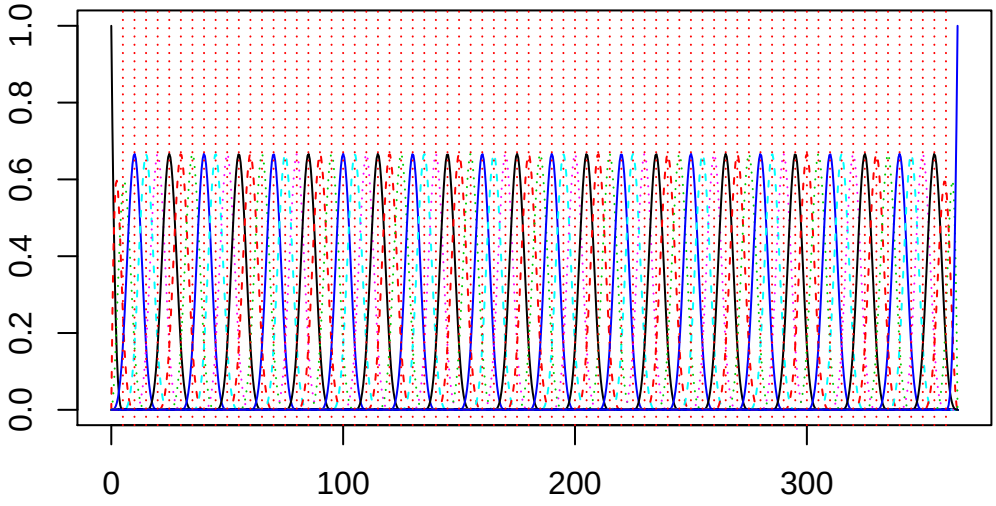
```
# There's some clear autocorrelation; more basis functions may be warranted.

###  EXERCISE: repeat the above with different numbers of basis functions (say
###  20, 50, 100, 200, 365. How many look like they give a reasonable smooth?

#### 2. B-spline bases with knots every 5 days

# First of all define a knot sequence; this will be the same as day5

knots = day5

# We'll use fourth-order B-splines (that is, a local polynomial of degree 3)

norder = 4

# this implies the number of basis functions

nbasis = length(knots) + norder - 2

# Now we can define the basis

bbasis = create.bspline.basis(dayrng,nbasis,norder,knots)

# If in doubt, we can obtain

bbasis$nbasis    # number of basis functions
```

```
## [1] 76
```

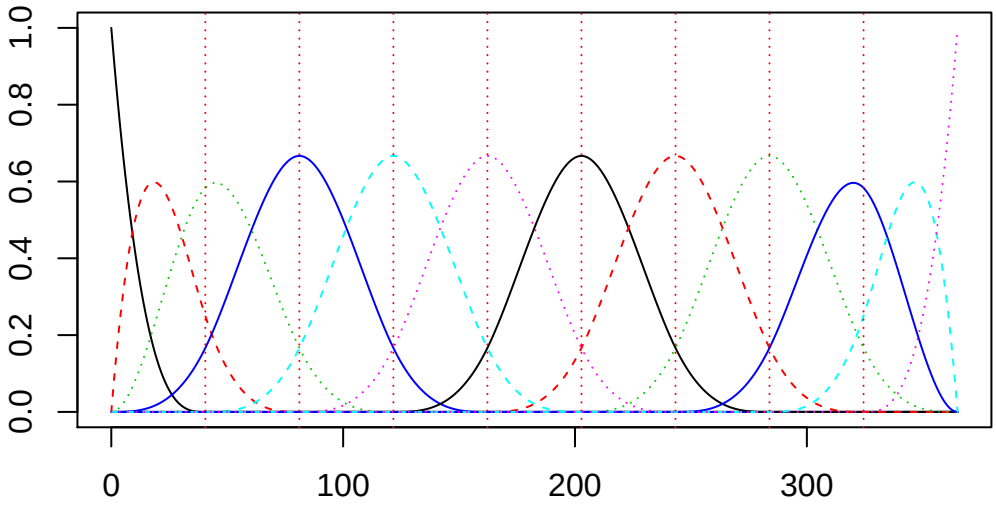
```
bbasis$rangeval   # basis range
```

```
## [1]   0 365
```

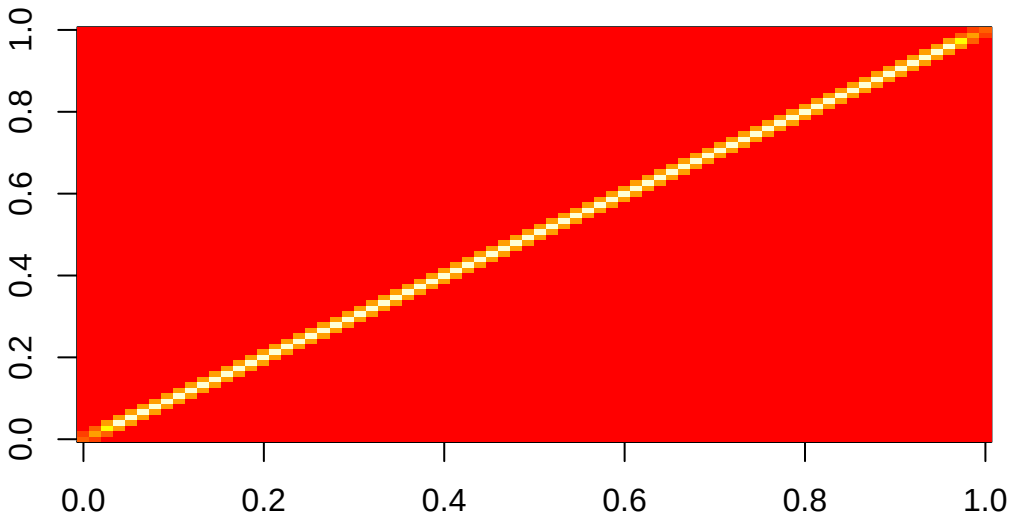
```
plot(bbasis)
```



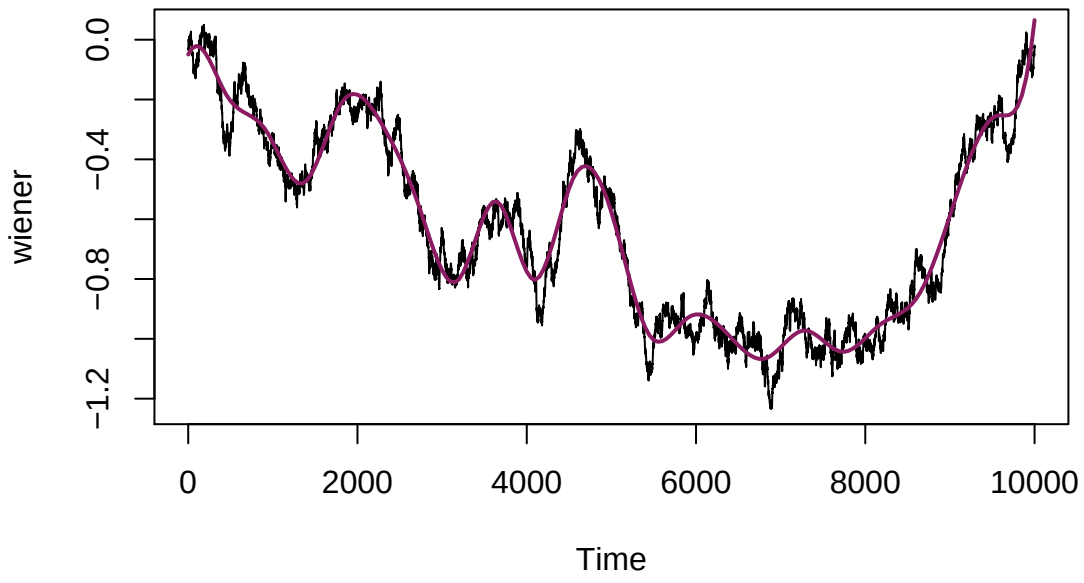

```
# but we can look at a smaller number

plot(create.bspline.basis(dayrng,nbasis=12,norder))
```



```r
# We can also look at the inner product of these (should be orthogonal)

in.mat = inprod(bbasis,bbasis)

par(mfrow=c(1,1))
image(in.mat)
```



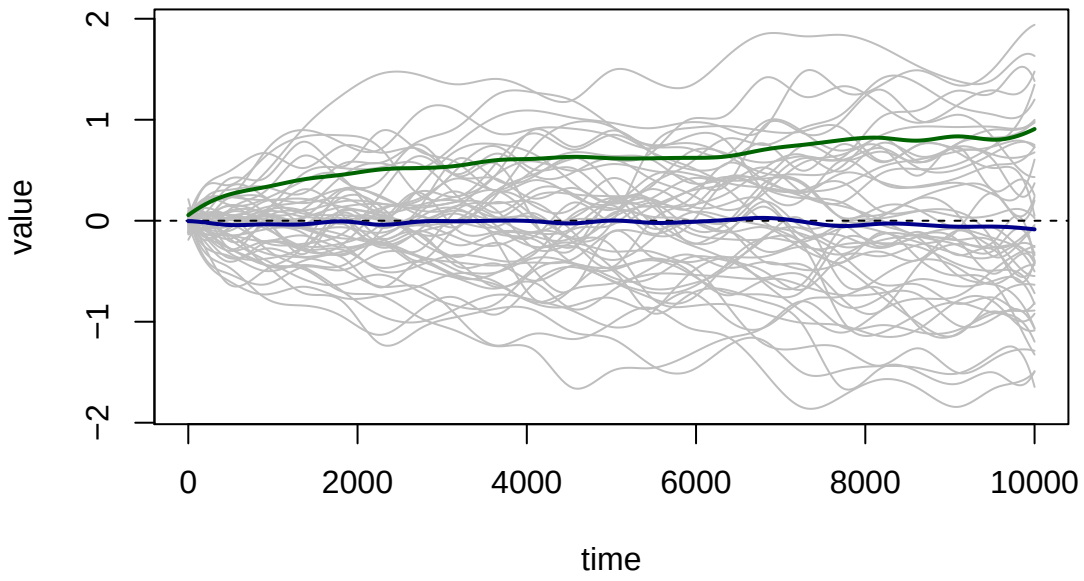

```r
# and see that it is zero outside of a diagonal band; this can help computation
# a great deal.

### EXERCISE: try changing the order of the basis and observe how the width
### of the support of the basis changes and how its smoothness properties change.

### EXERCISE: obtain a least squares smooth of these data with the Bspline basis
### how does this compare with a least squares smooth using a Fourier basis with
### the same number of basis functions?

##########  Wiener Process in 0,10000  ## from fda package
wiener=c(0,cumsum(rnorm(10^4-1))/100 )
plot.ts(wiener)
B25.basis=create.bspline.basis(rangeval=c(0,10^4),nbasis=25)
wiener.fd=smooth.basis(y=wiener,fdParobj=B25.basis)
lines(wiener.fd,col="maroon4",lwd=2)
```



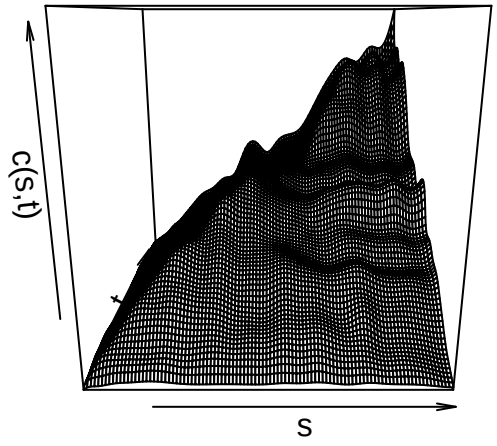

```
#several curves
N=50
w.vec=rnorm(10^4*N)/100
w.mat=matrix(rnorm((10^4-1)*N)/100,ncol=N,nrow=10^4-1)
w.mat=rbind(rep(0,N),w.mat)
w.mat=apply(w.mat,2,cumsum)
w.fd=smooth.basis(y=w.mat,fdParobj=B25.basis)
plot(w.fd,col="gray",lty=1)
```

```
## [1] "done"
```

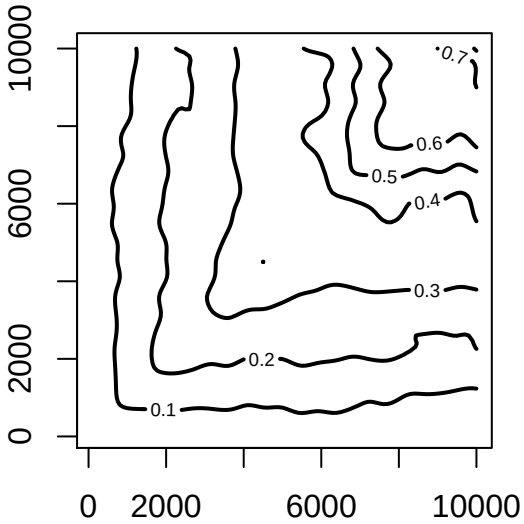
```
lines(mean(w.fd$fd),col="darkblue",lwd=2)
lines(std.fd(w.fd$fd),col="darkgreen",lwd=2)
```




```
## computing the covariance matrix
w.cov=var.fd(w.fd$fd)
## to plot it
grid=(1:100)*100
w.cov.mat=eval.bifd(grid,grid,w.cov)
persp(grid,grid,w.cov.mat, xlab="s",ylab="t",zlab="c(s,t)")
```




```
par(pty="s")
contour(grid,grid,w.cov.mat,lwd=2)
```

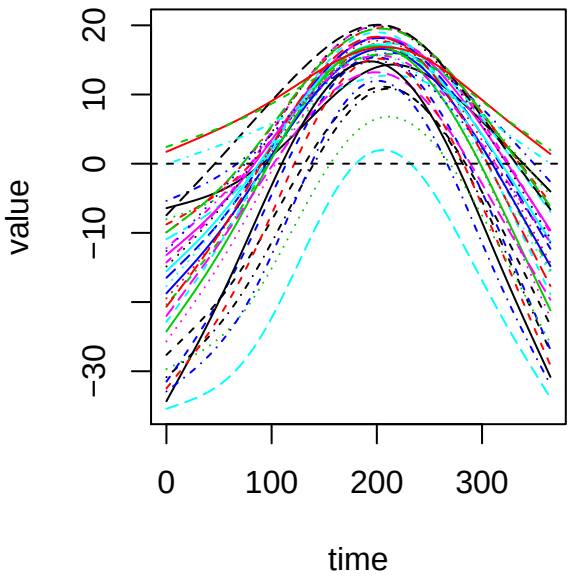



```
################################################################################
###                           SMOOTHING FUNCTIONS                            ####
```

```
##############################################################################################

#### 1. Lfd Objects

# Two common means of generating Lfd objects
# 1. int2Lfd -- just penalize some derivatives.

curv.Lfd = int2Lfd(2)

# 2. vec2Lfd -- a (constant) linear combination of derivatives; for technical
# reasons this also requires the range of the basis.

harmLfd = vec2Lfd(c(0,(2*pi/365)^2,0),rangeval=dayrng)

# looking inside these objects is not terribly enlightening.

#### 2. fdPar objects

# We'll concentrate on B-splines and second-derivative penalties.

# First, a value of lambda  (purposefully large so that we can distinguish a fit
# from data below).

lambda = 1e6

# Now we can define the fdPar object

curv.fdPar = fdPar(bbasis,curv.Lfd,lambda)

#### 3. Smoothing functions

# We're now in a position to smooth

tempSmooth1 = smooth.basis(daytime,temp,curv.fdPar)

# Let's look at the result

names(tempSmooth1)
```

```
## [1] "fd"      "df"      "gcv"     "beta"    "SSE"     "penmat"  "y2cMap"
## [8] "argvals" "y"
```

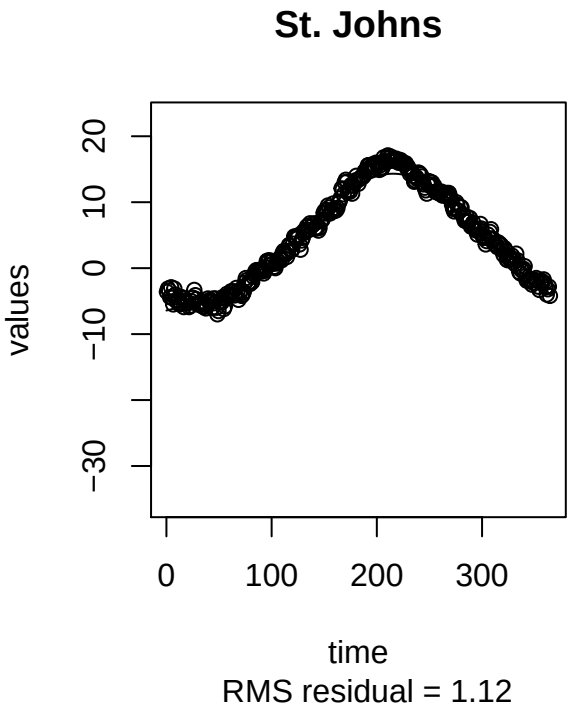
```
# First of all, let's plot it

plot(tempSmooth1$fd)
```

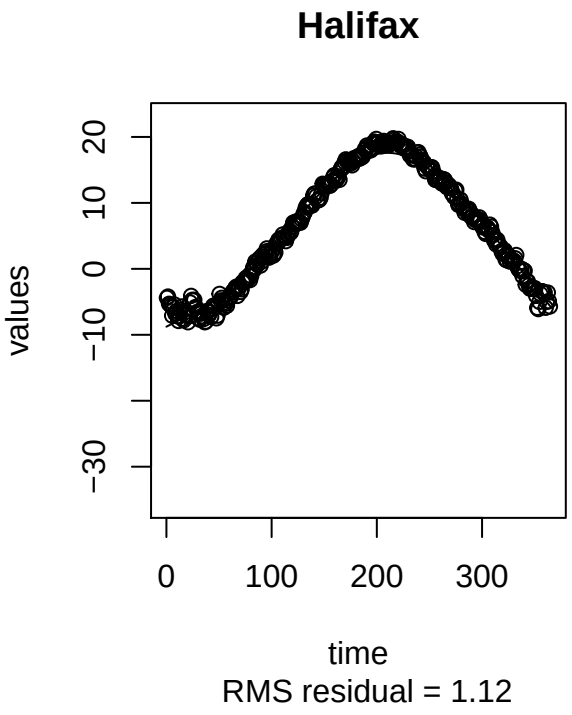


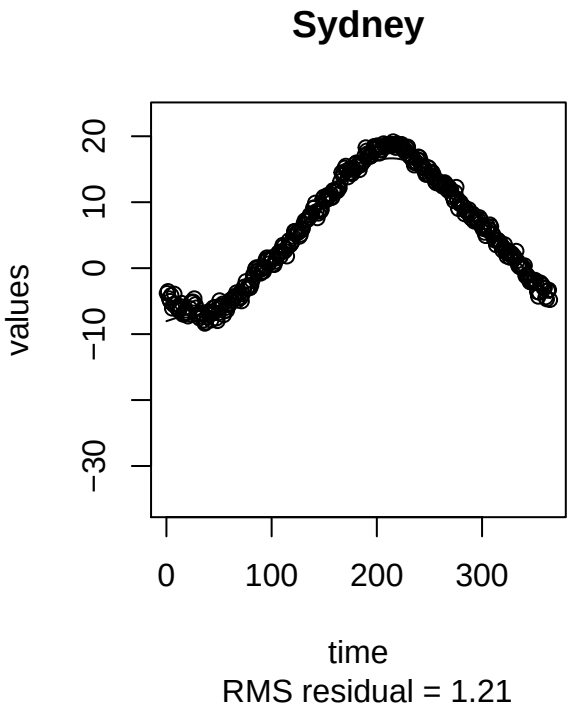

```
## [1] "done"
```

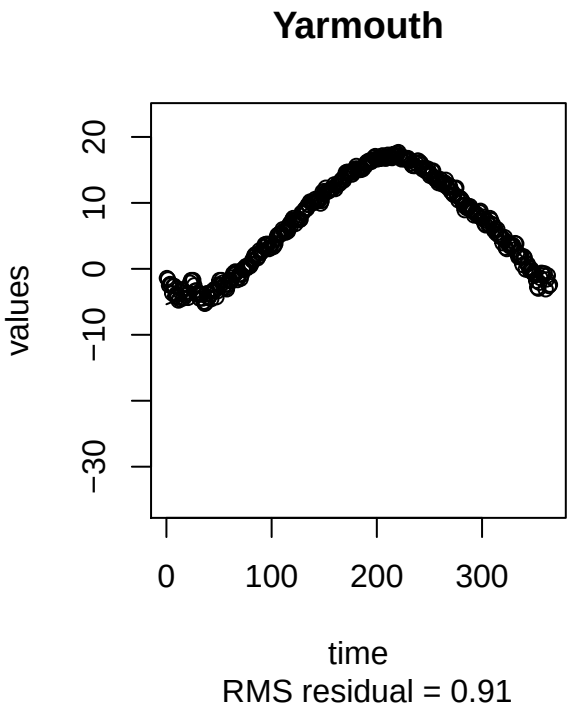
```
# There is also a neat utility to go through each curve in turn and look at its
# fit to the data:

plotfit.fd(temp,daytime,tempSmooth1$fd)
```

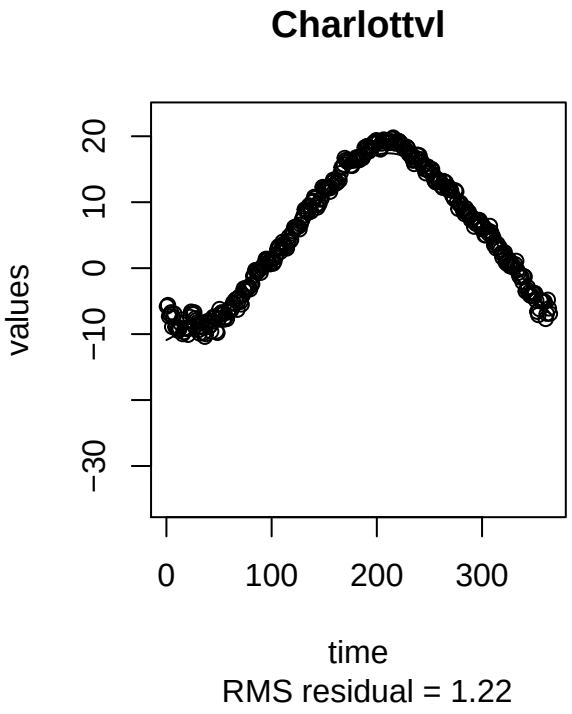
```
## Multiple plots:  Click in the plot to advance to the next plot
```

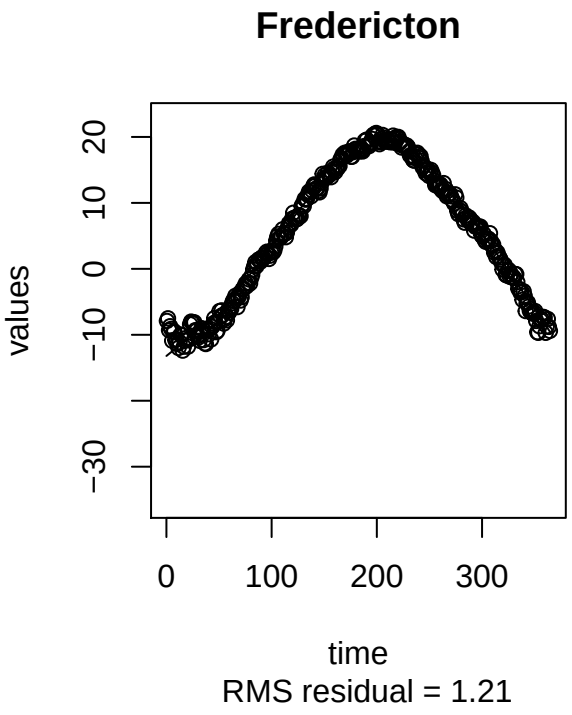

St. Johns
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.12

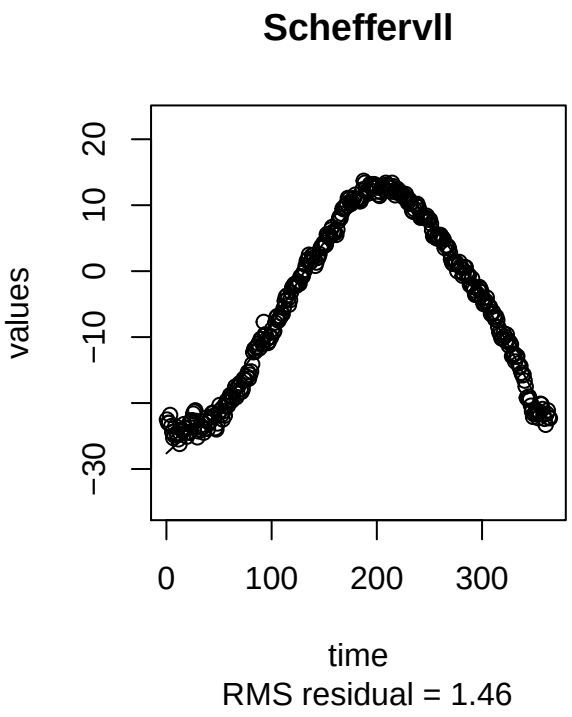

Halifax
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.12

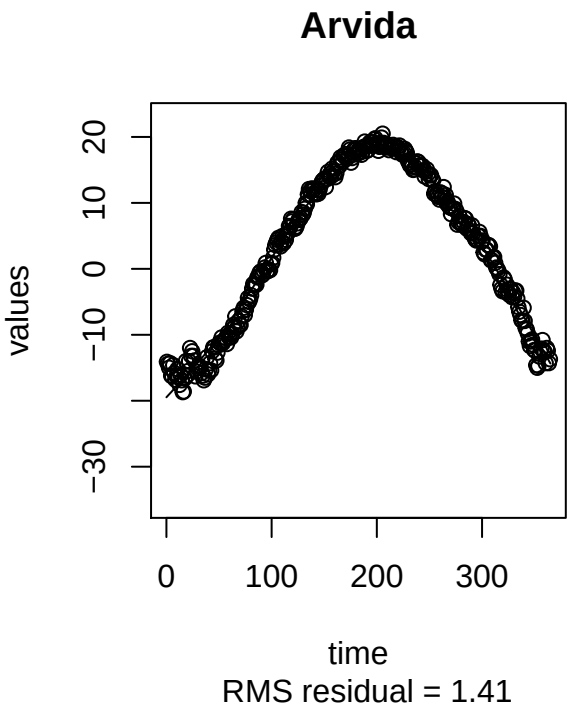

Sydney
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.21

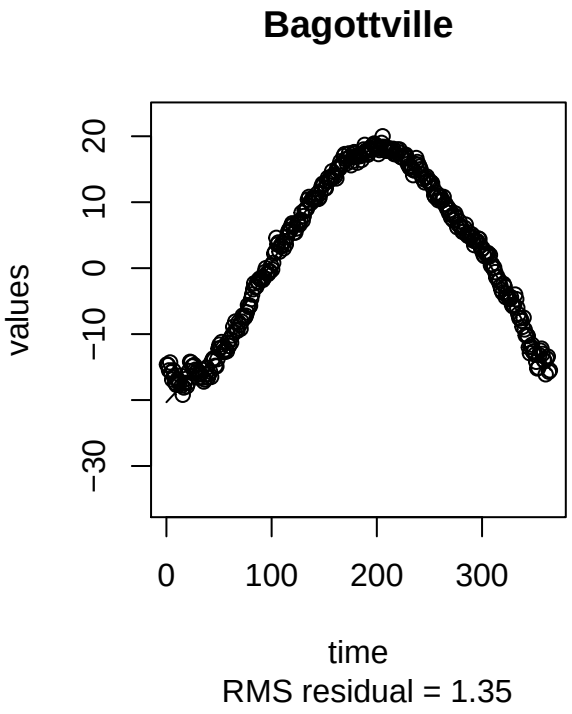

Yarmouth
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.91

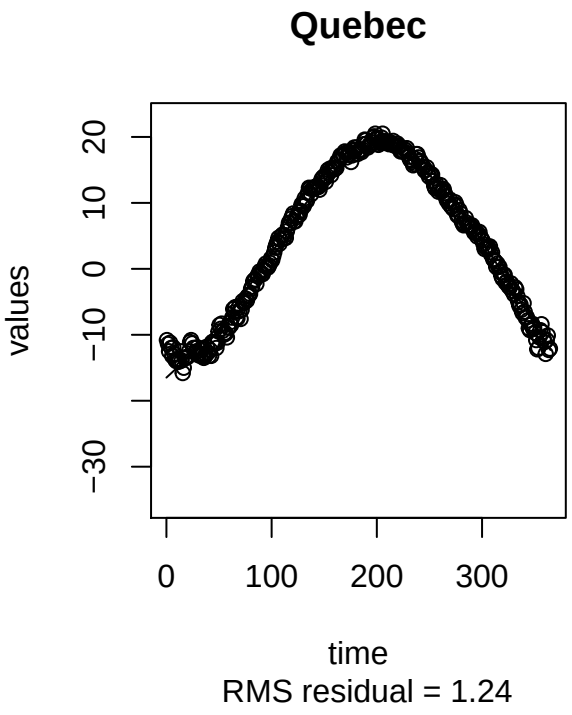

Charlottvl
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.22

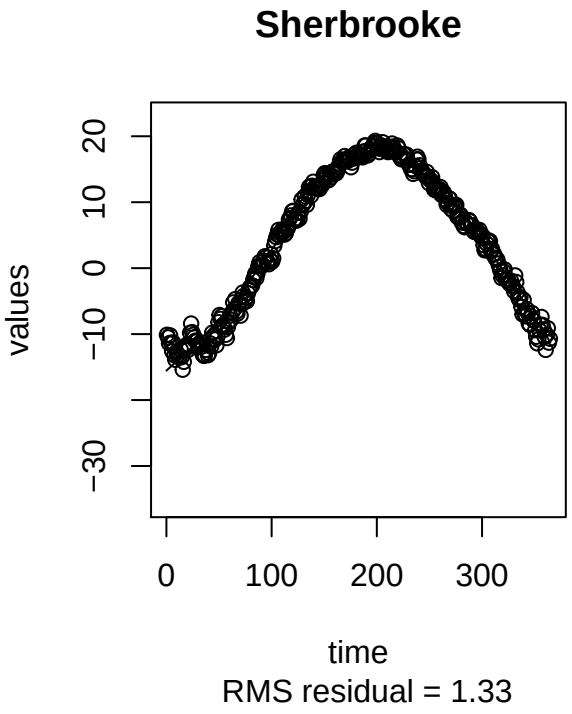

Fredericton
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.21

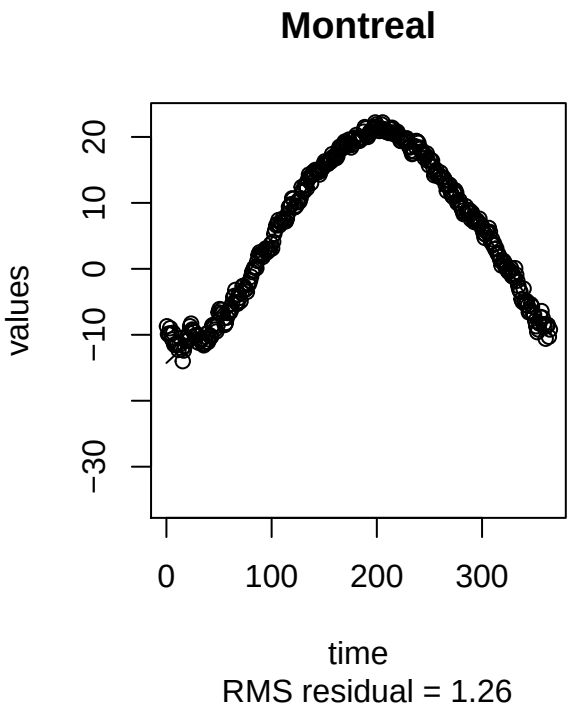

Scheffervll
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.46

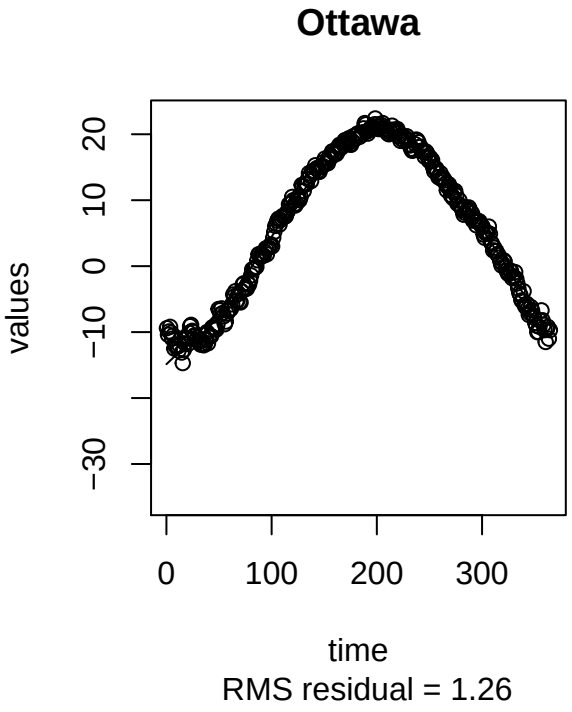

Arvida
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.41

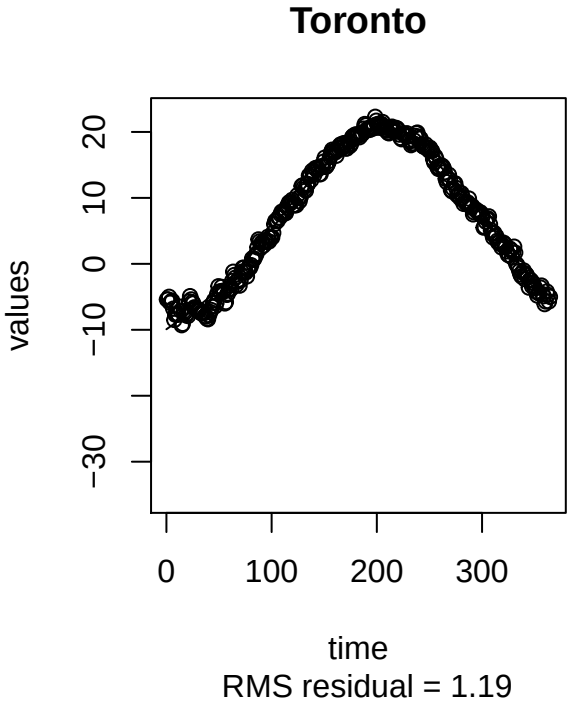

Bagottville
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.35

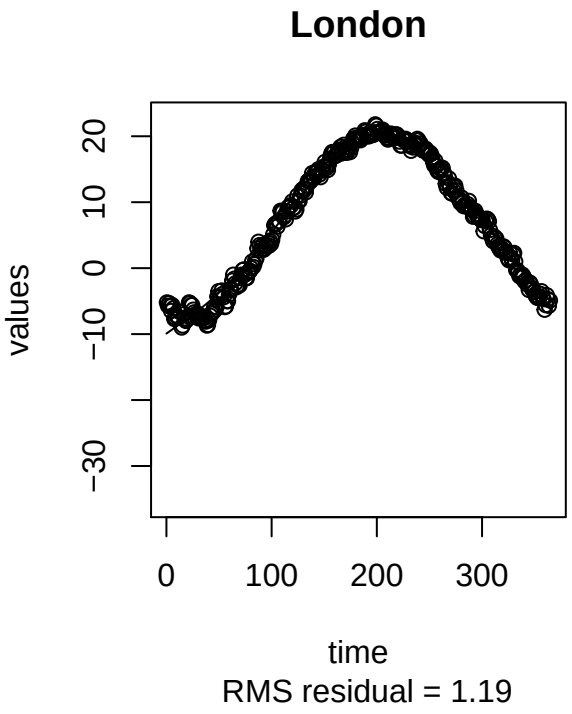

Quebec
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.24

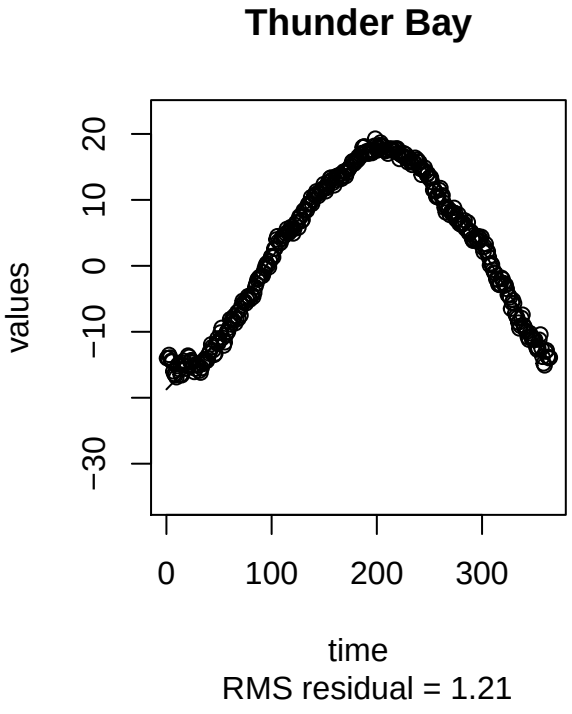

Sherbrooke
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.33

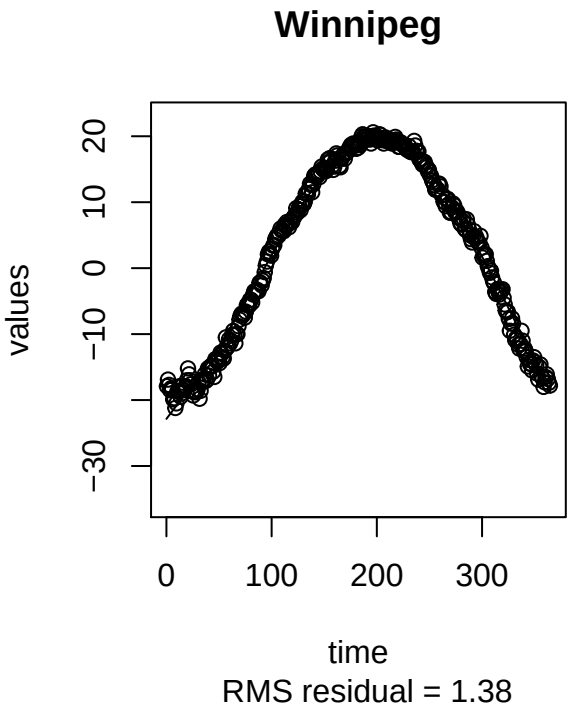

Montreal
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.26

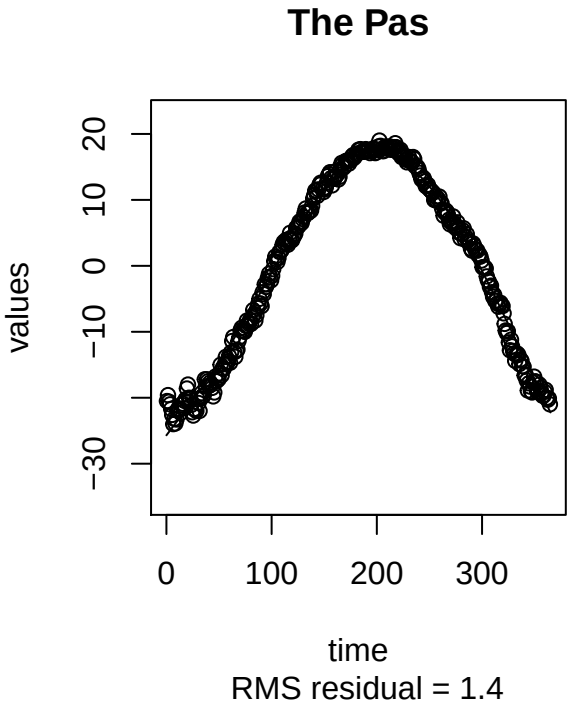

Ottawa
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.26

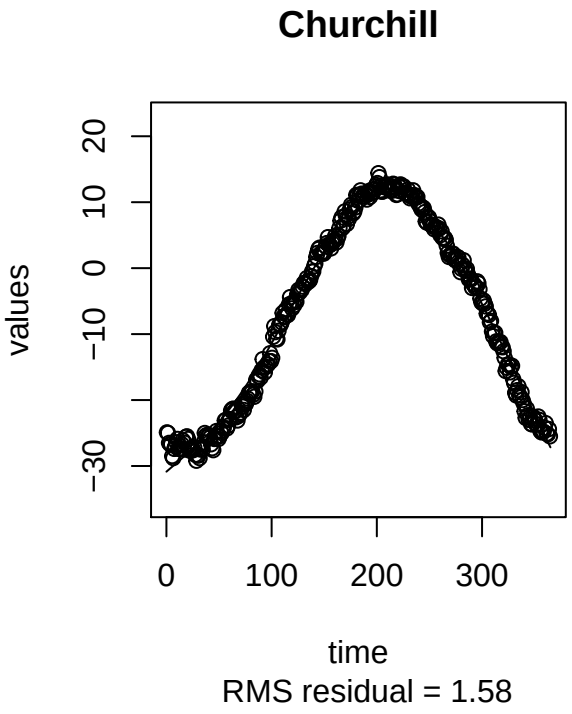

Toronto
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.19

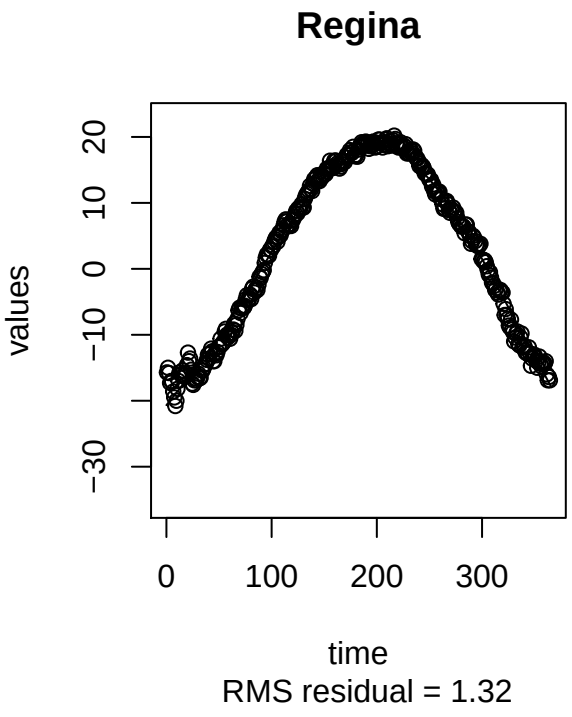

London
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.19

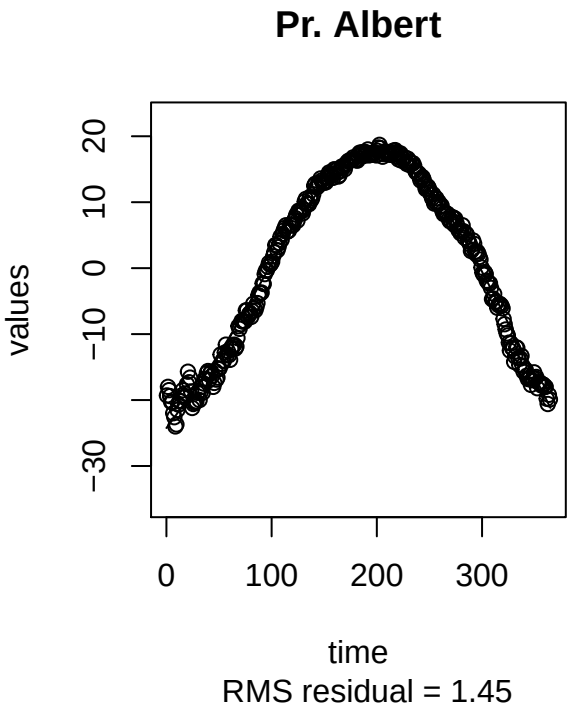

Thunder Bay
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.21

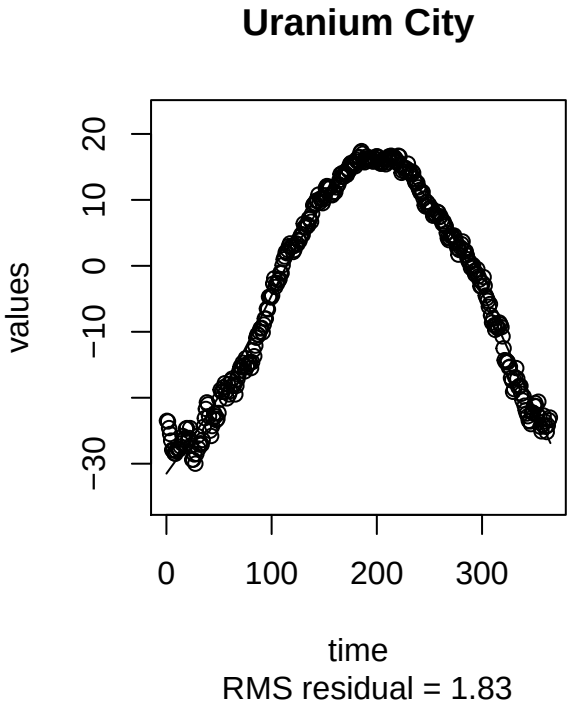

Winnipeg
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.38

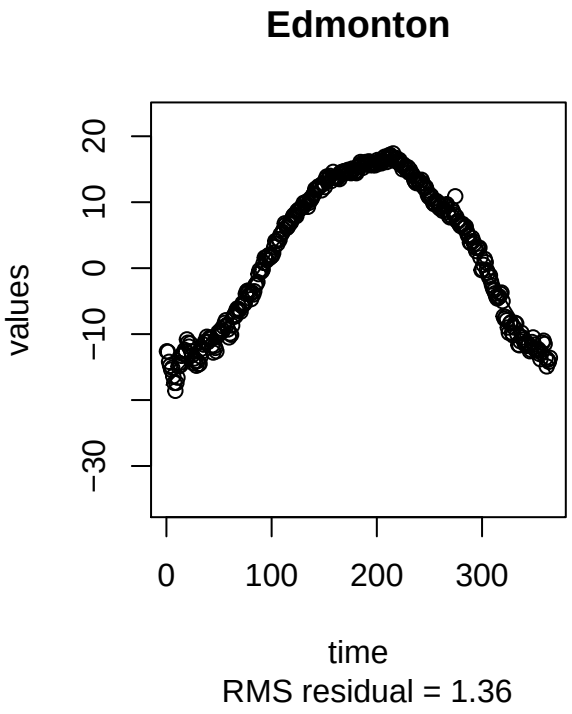

The Pas
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.4

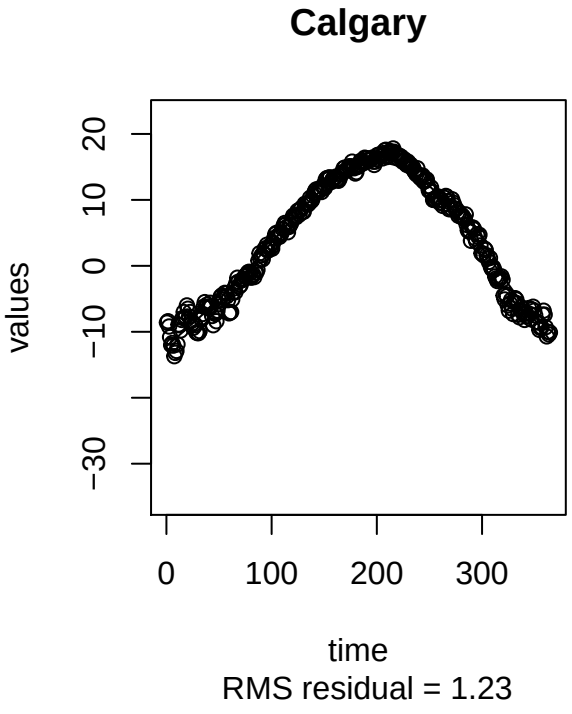

Churchill
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.58

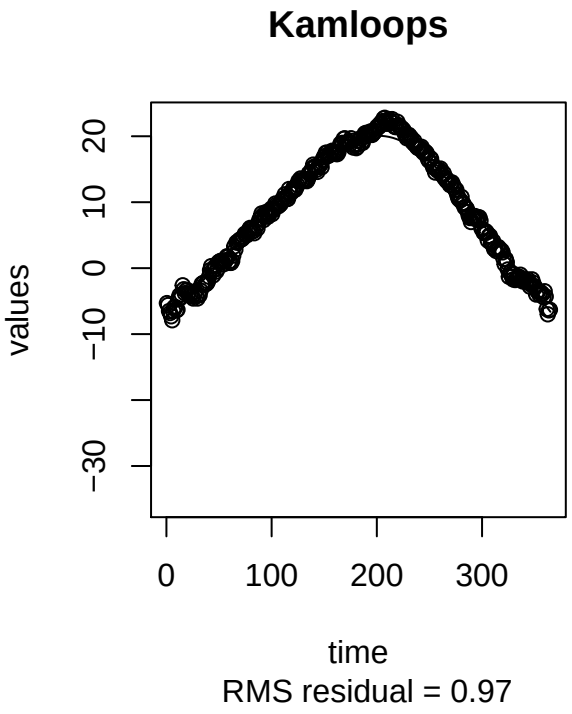

Regina
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.32

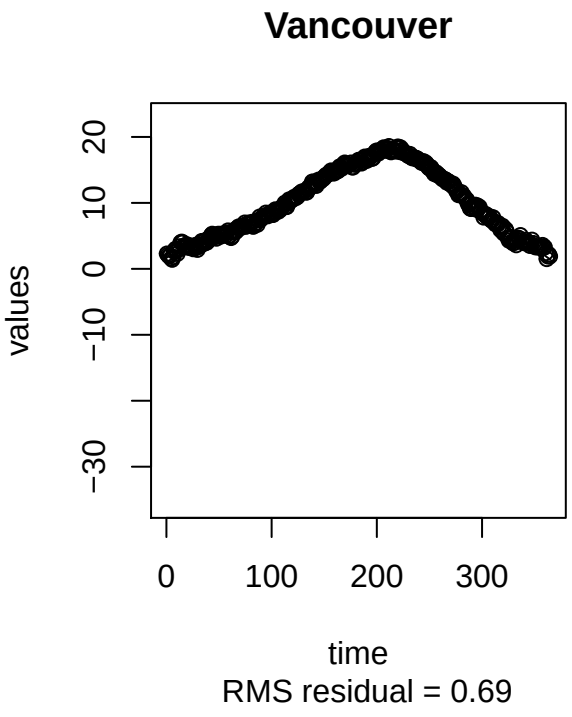

Pr. Albert
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.45

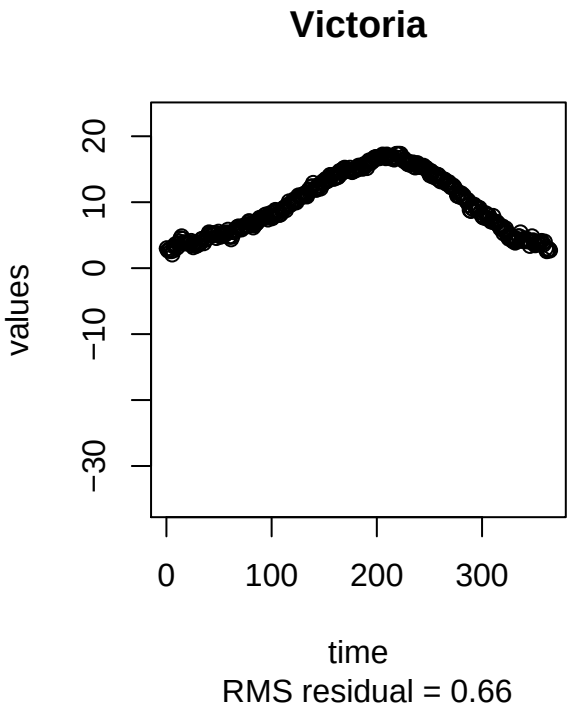

Uranium City
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.83

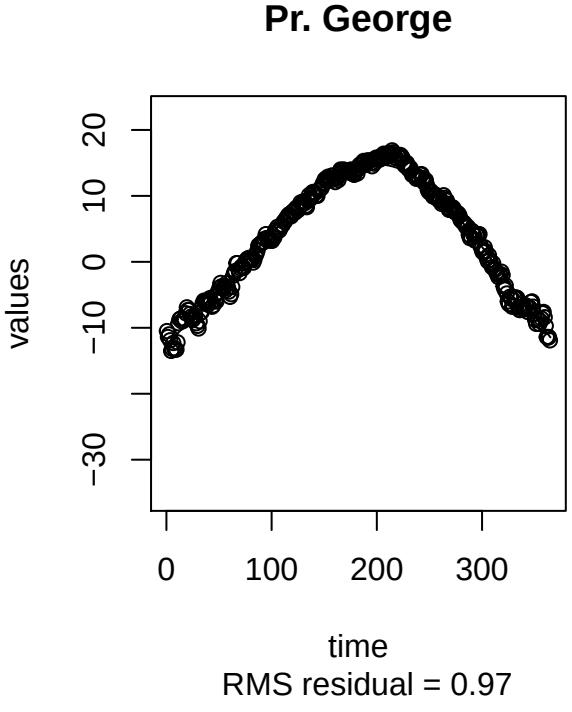

Edmonton
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.36

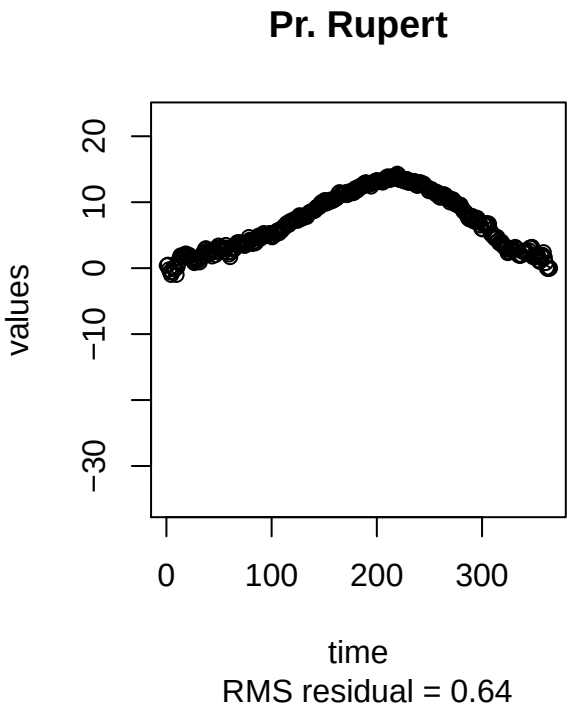

Calgary
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.23

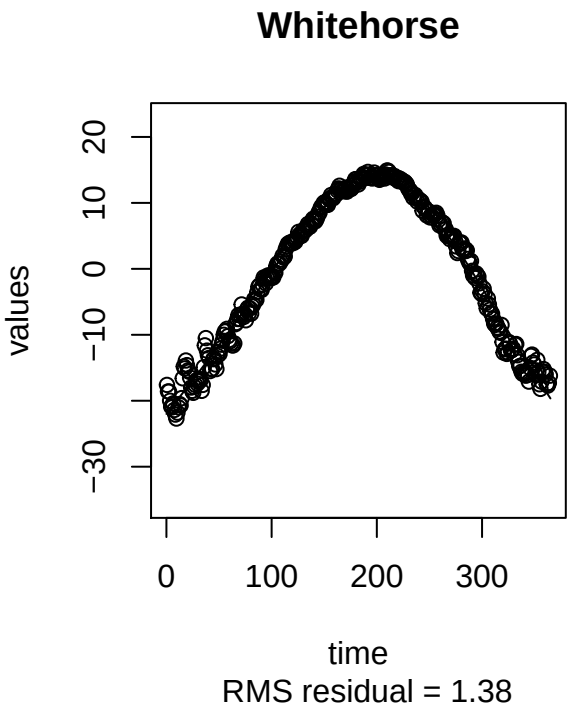

Kamloops
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.97

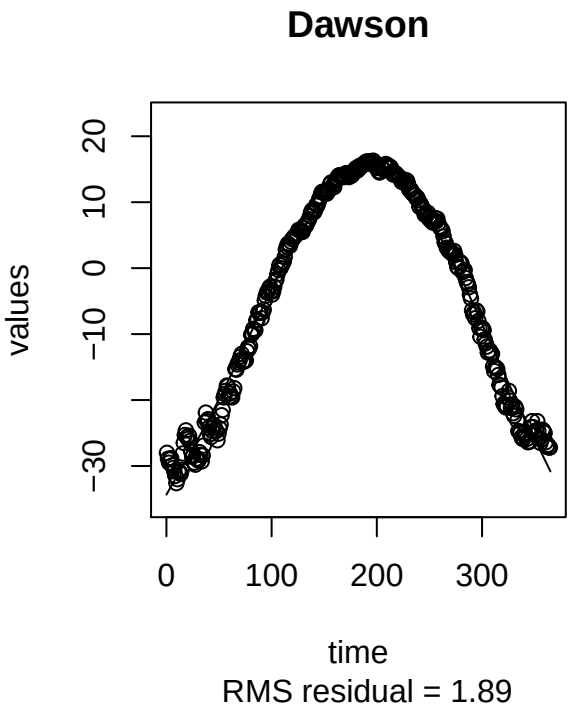

Vancouver
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.69

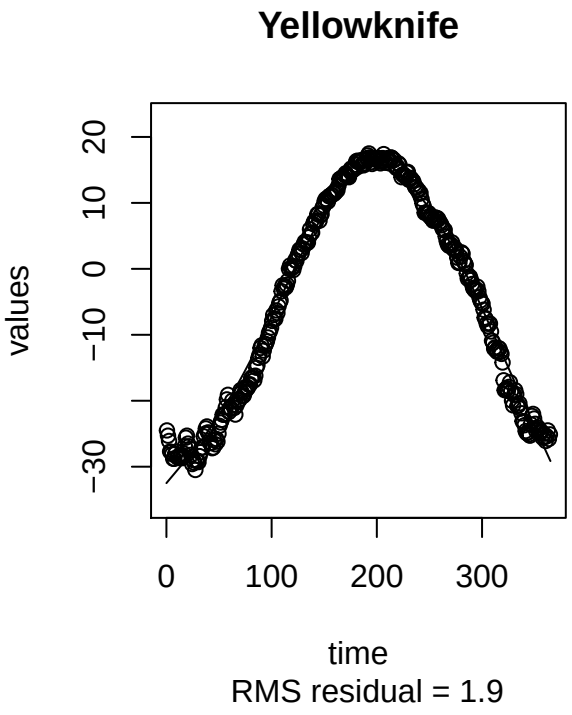

Victoria
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.66

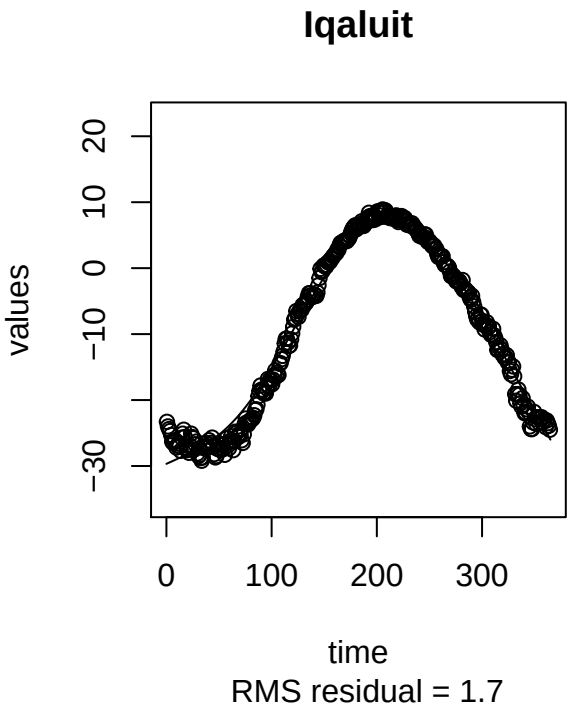

Pr. George
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.97

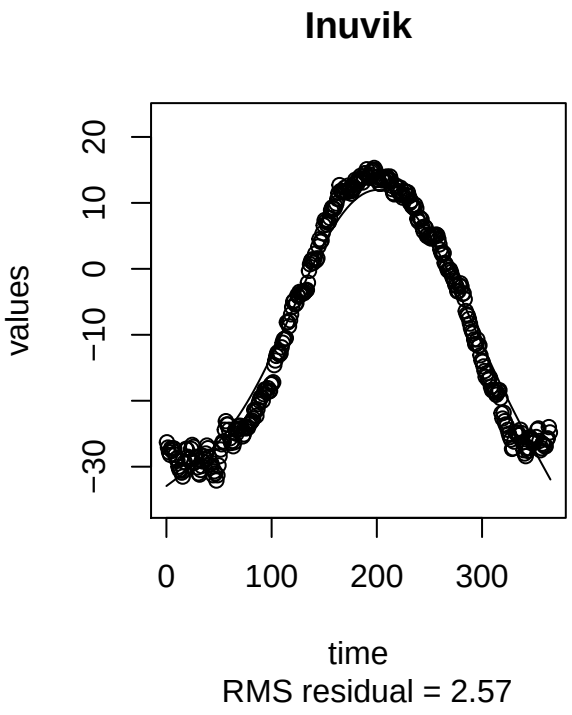

Pr. Rupert
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.64

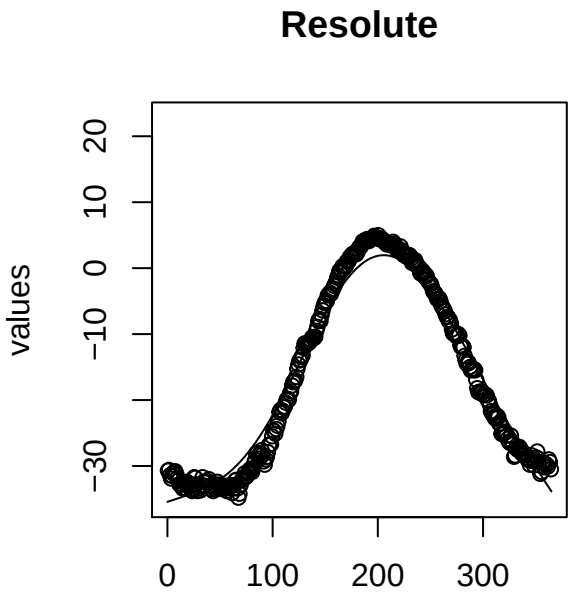

Whitehorse
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.38


Dawson
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.89


Yellowknife
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.9


Iqaluit
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 1.7


Inuvik
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 2.57




```
# Let's examine some fit statistics

# degrees of freedom

tempSmooth1$df
```

```
## [1] 5.080492
```

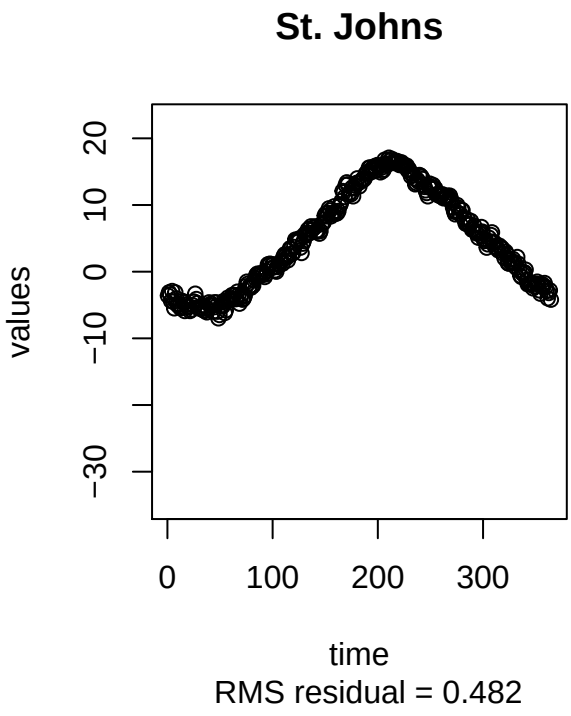
```
# Just about equivalent to fitting 5 parameters

# We'll also look at GCV, this is given for each observation

tempSmooth1$gcv
```

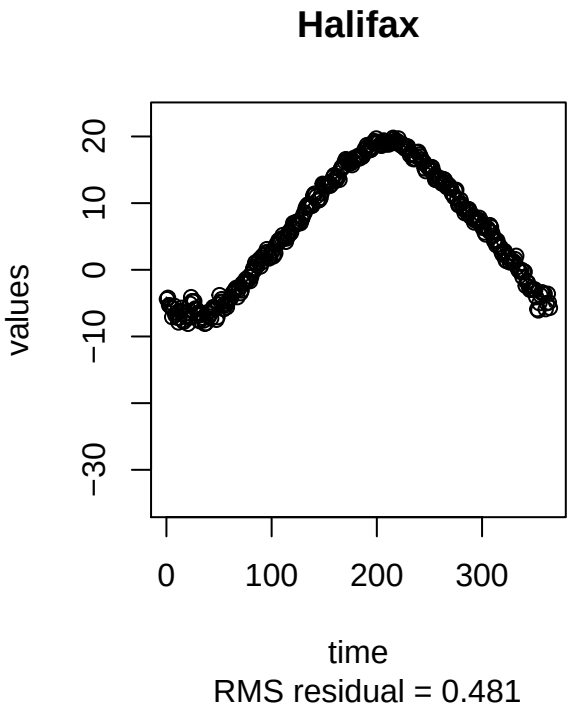
```
##    St. Johns      Halifax       Sydney     Yarmouth   Charlottvl
##    1.2793025    1.2915703    1.5055770    0.8453448    1.5280518
##  Fredericton  Scheffervll       Arvida  Bagottville       Quebec
##    1.5144666    2.1862025    2.0336730    1.8812819    1.5729447
##   Sherbrooke     Montreal       Ottawa      Toronto       London
##    1.8063005    1.6287013    1.6413104    1.4649208    1.4647411
##  Thunder Bay     Winnipeg      The Pas    Churchill       Regina
##    1.4935841    1.9585076    2.0169531    2.5643633    1.7813282
##   Pr. Albert Uranium City     Edmonton      Calgary     Kamloops
##    2.1509234    3.4357030    1.9078682    1.5604659    0.9611010
##    Vancouver     Victoria   Pr. George   Pr. Rupert   Whitehorse
##    0.4836271    0.4491240    0.9751638    0.4194347    1.9688753
##       Dawson  Yellowknife      Iqaluit       Inuvik     Resolute
##    3.6698310    3.6946757    2.9823377    6.8100775    4.7478155
```

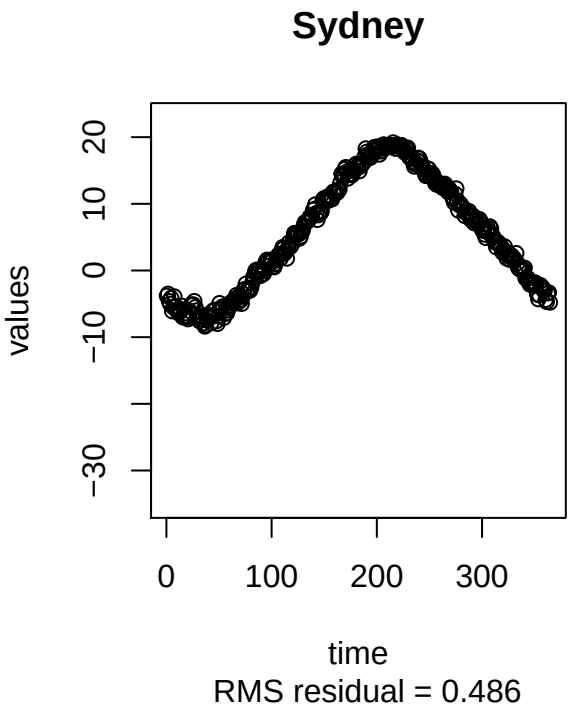
```
# Let's change to a more realistic value of lambda
```

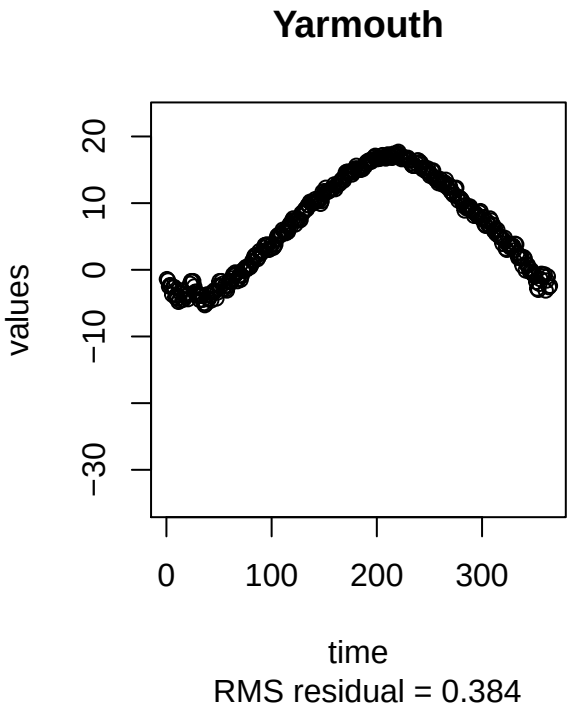
```
lambda = 1e1
curv.fdPar$lambda = lambda

tempSmooth = smooth.basis(daytime,temp,curv.fdPar)

# and repeat the previous steps

plotfit.fd(temp,daytime,tempSmooth$fd)
```

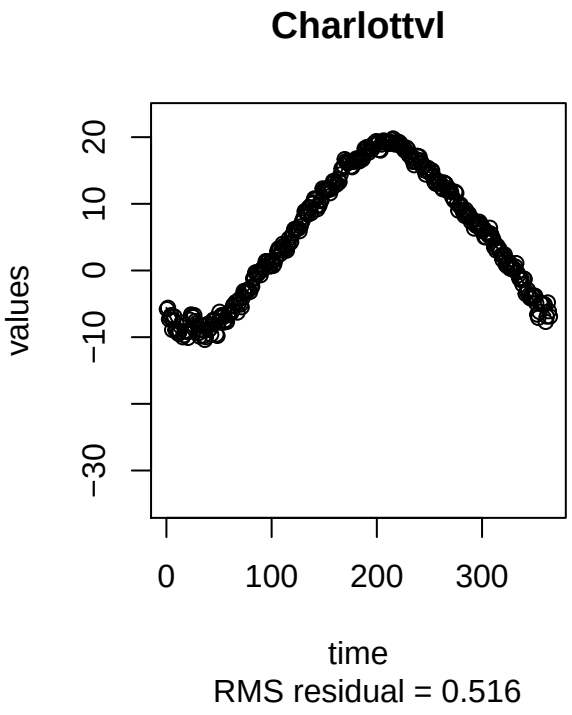
```
## Multiple plots:  Click in the plot to advance to the next plot
```

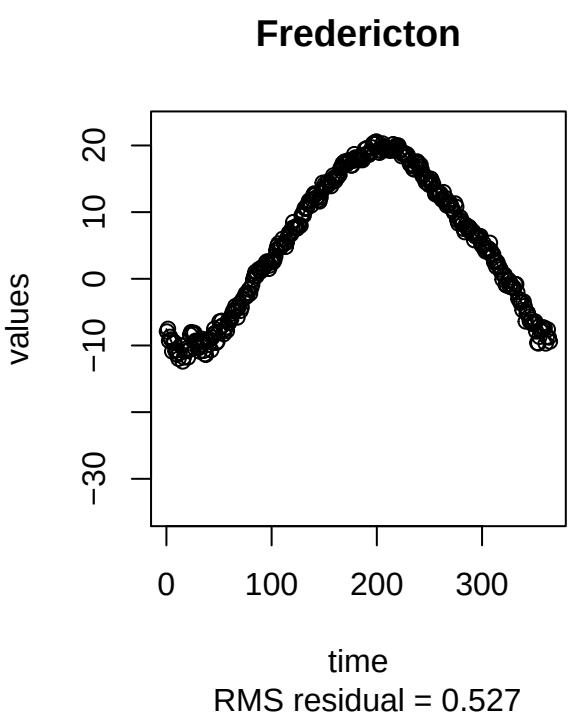


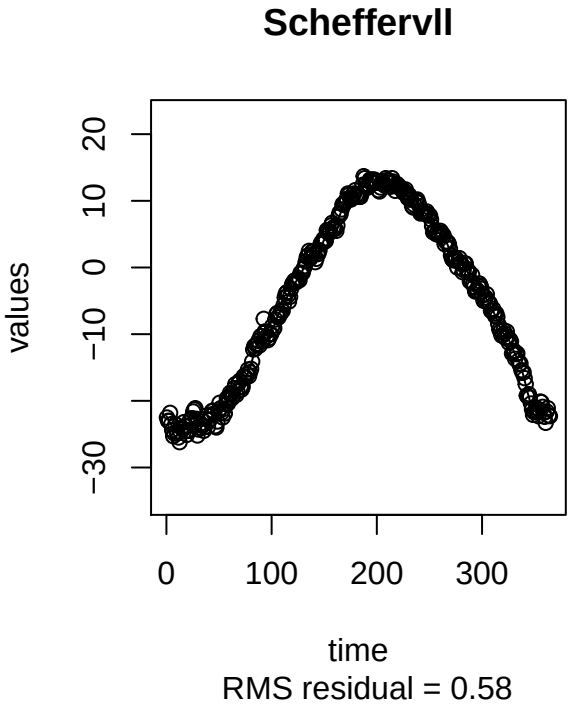

Halifax
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.481

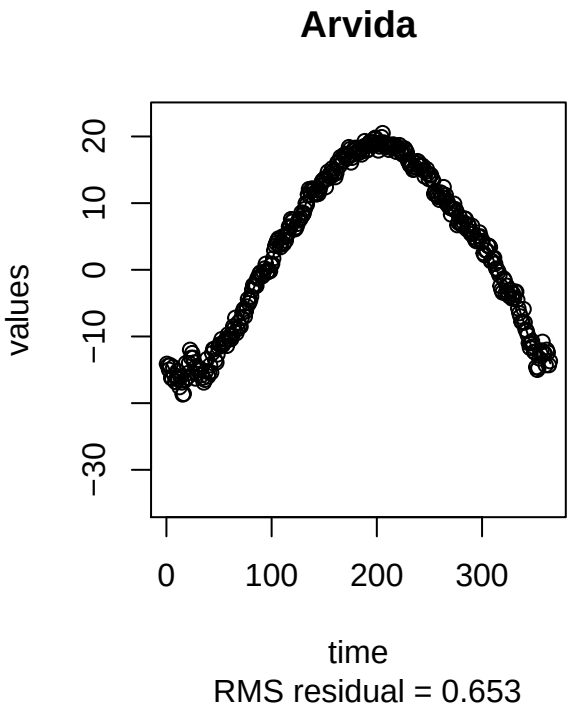

Sydney
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.486

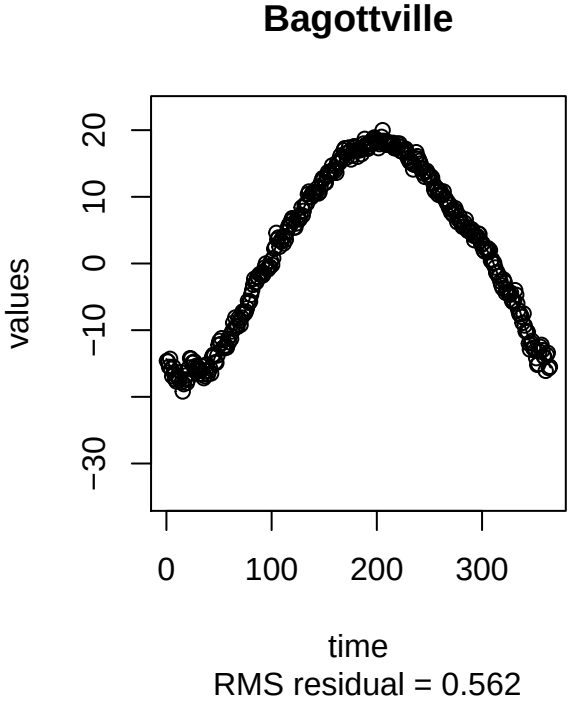

Yarmouth
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.384

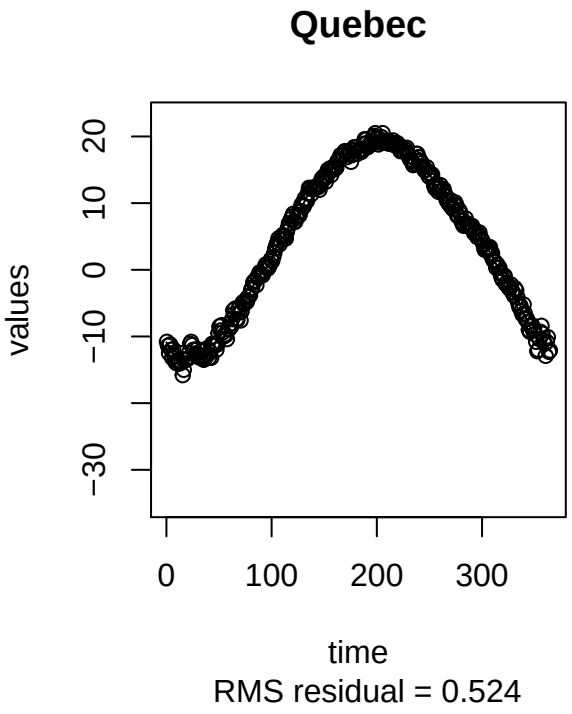

Charlottvl
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.516

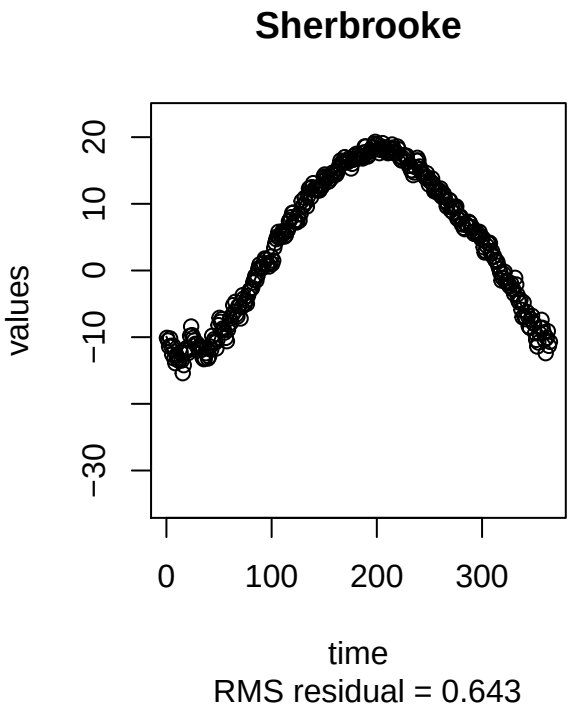

Fredericton
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.527

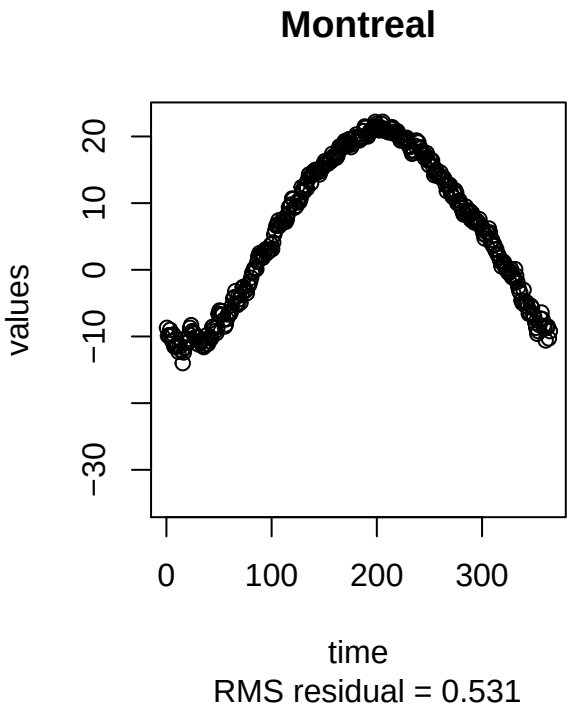

Scheffervll
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.58

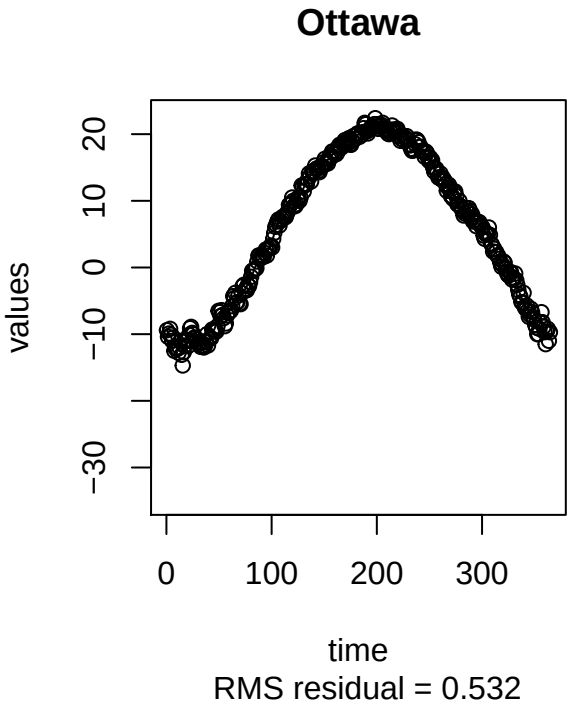

Arvida
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.653

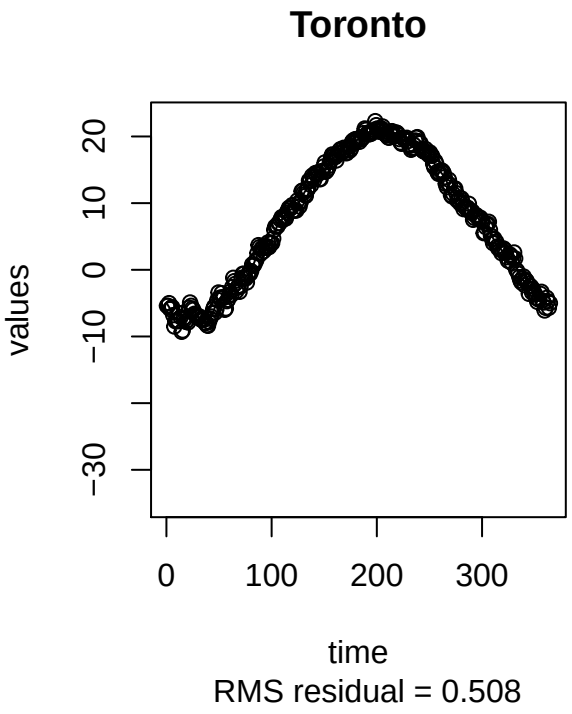

Bagottville
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.562

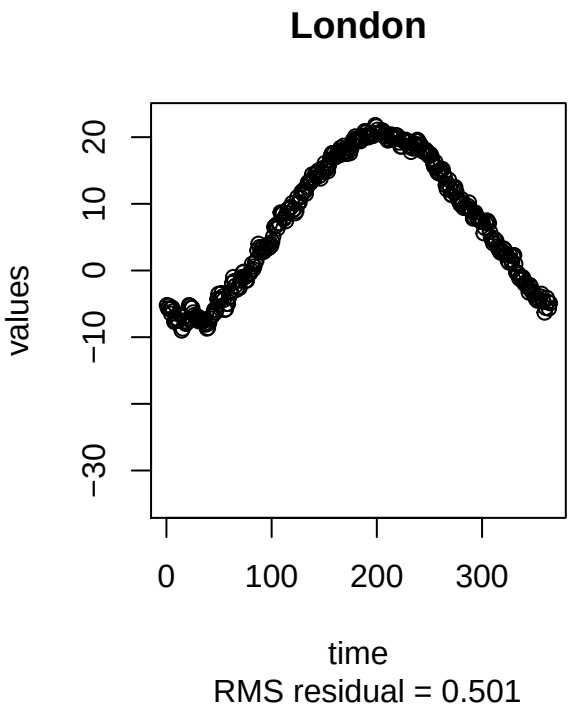

Quebec
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.524

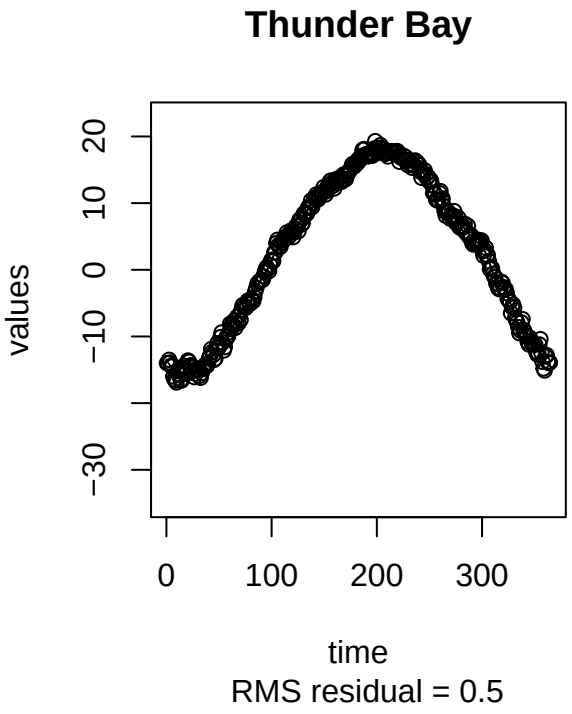

Sherbrooke
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.643

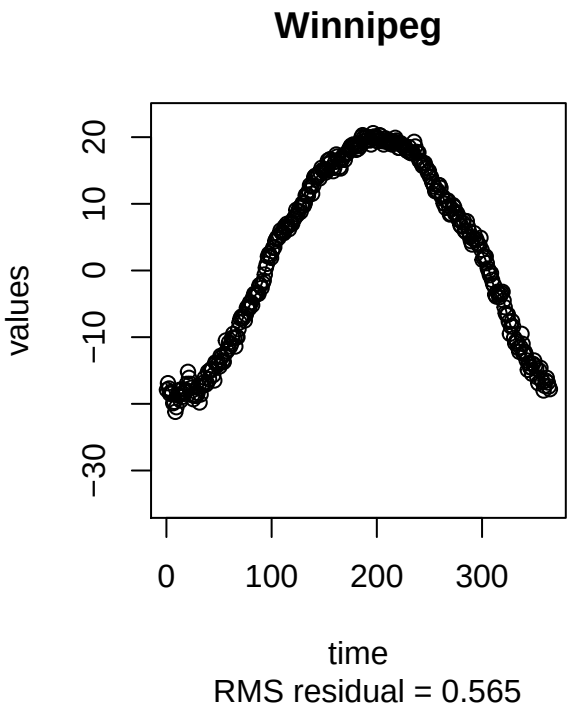

Montreal
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.531

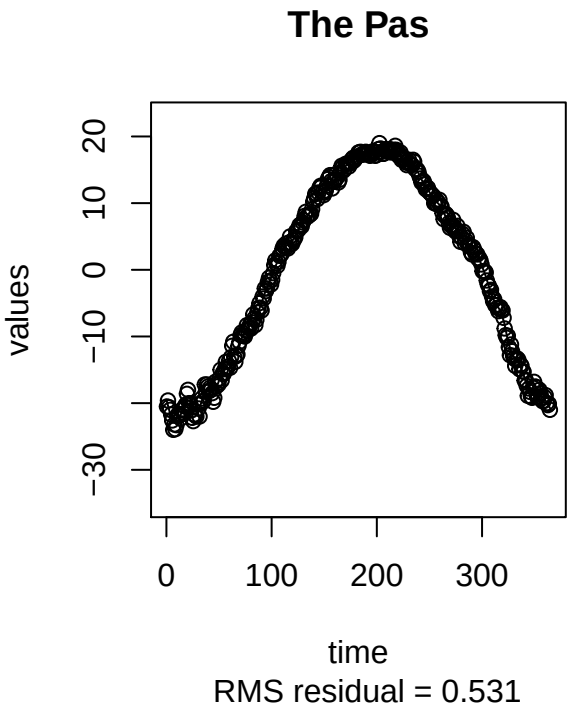

Ottawa
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.532

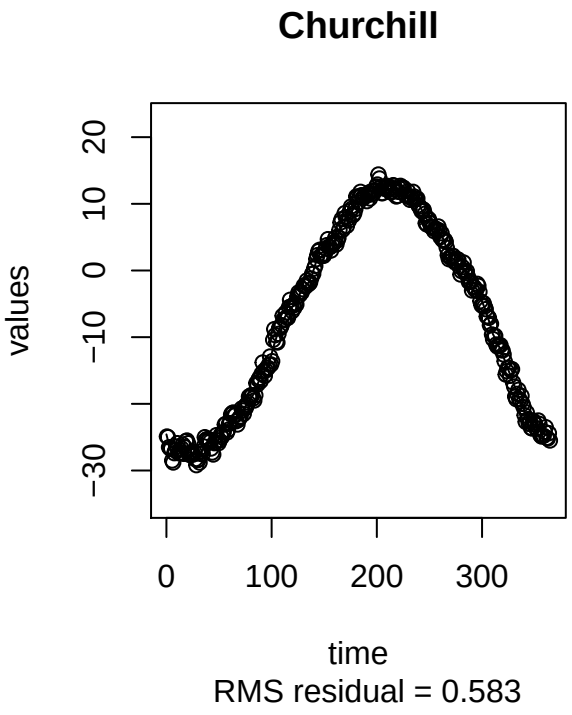

Toronto
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.508

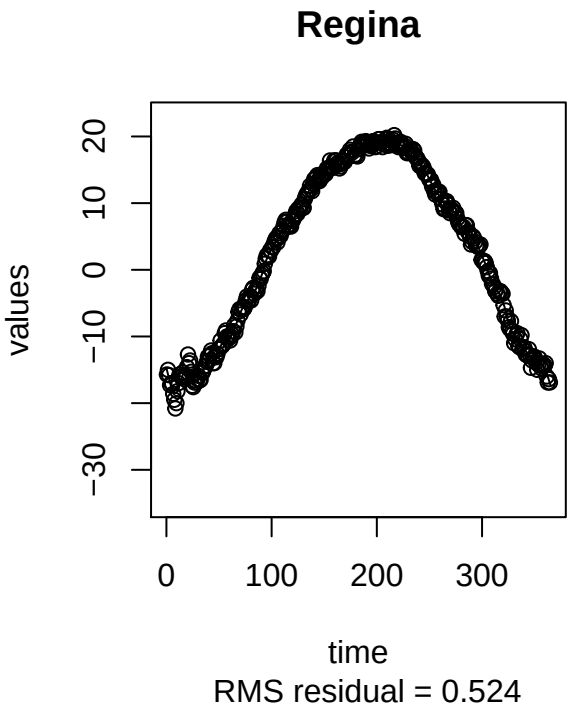

London
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.501

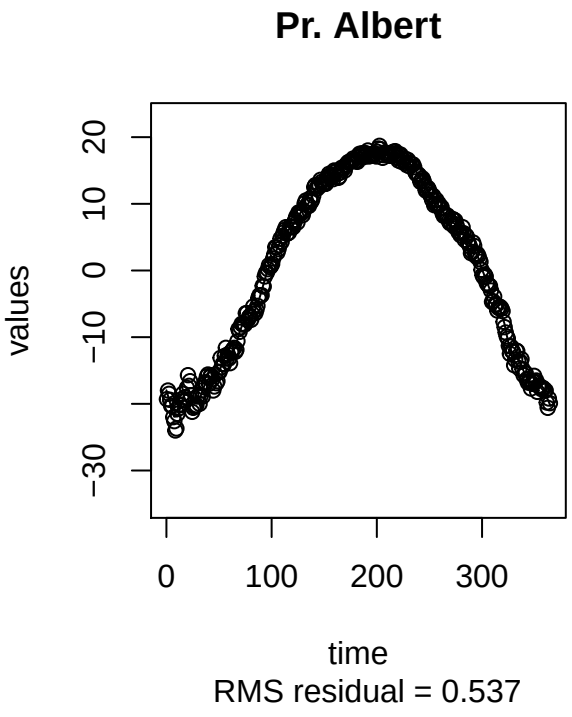

Thunder Bay
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.5

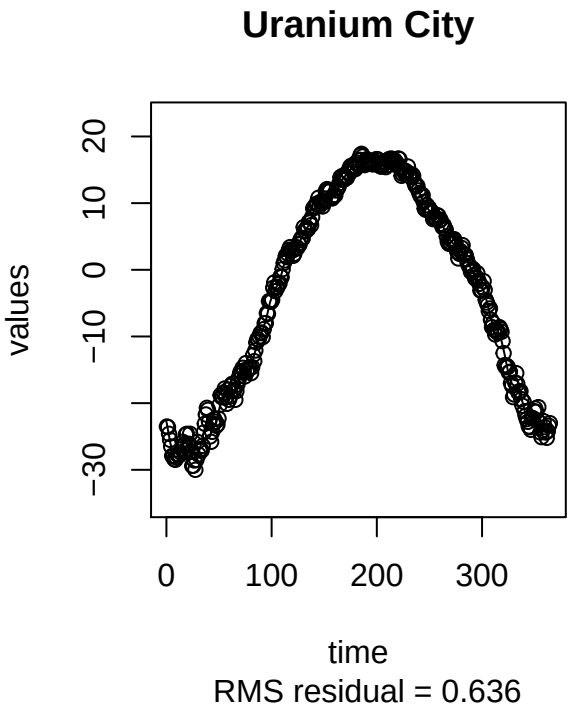

Winnipeg
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.565

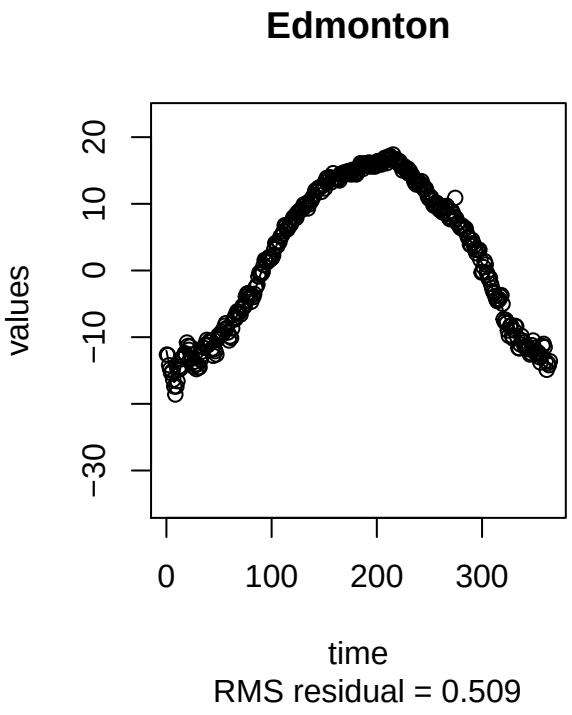

The Pas
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.531

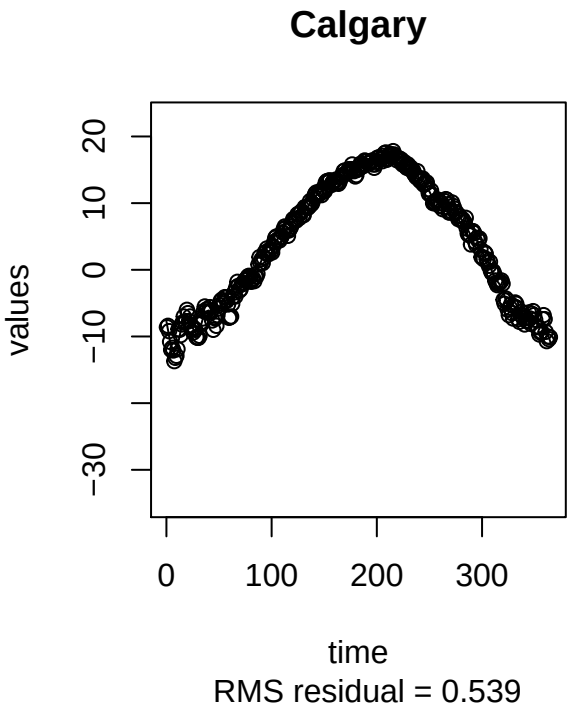

Churchill
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.583

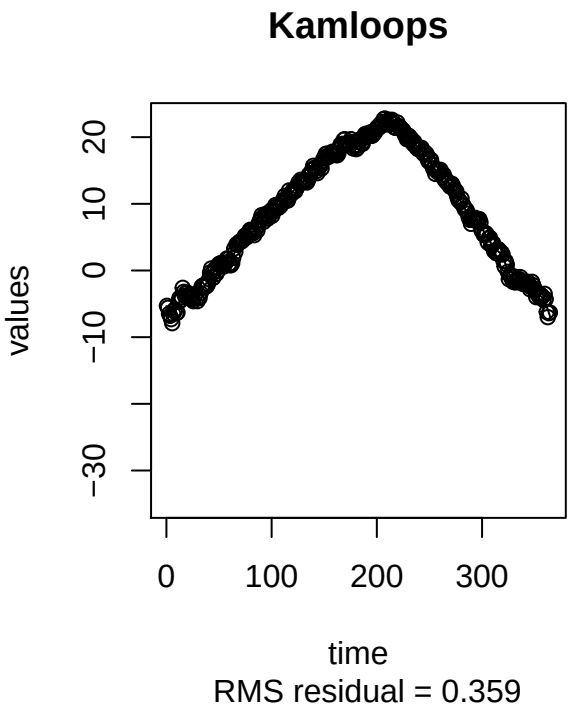

Regina
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.524

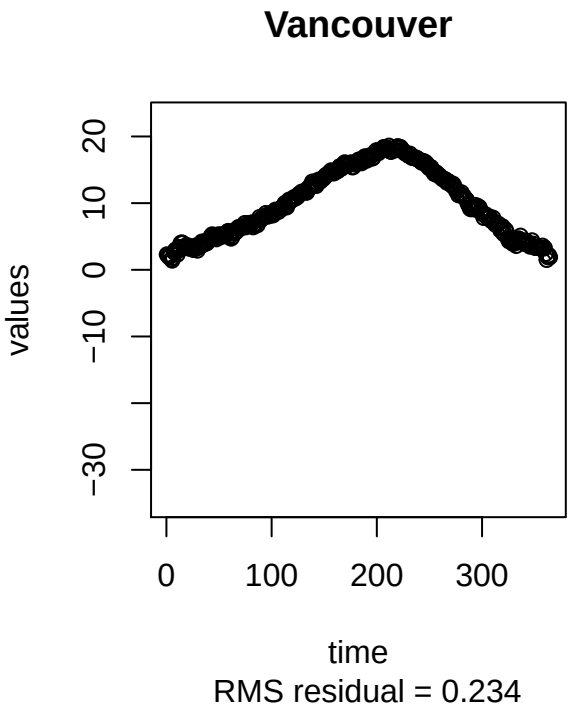

Pr. Albert
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.537

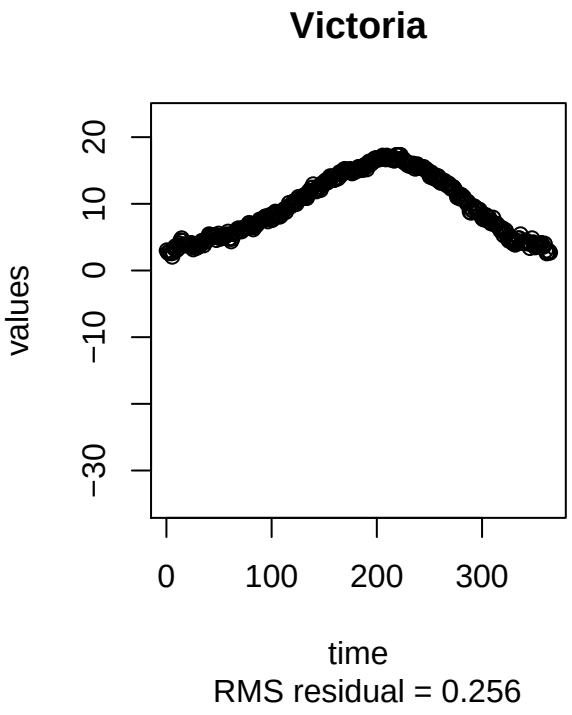

Uranium City
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.636

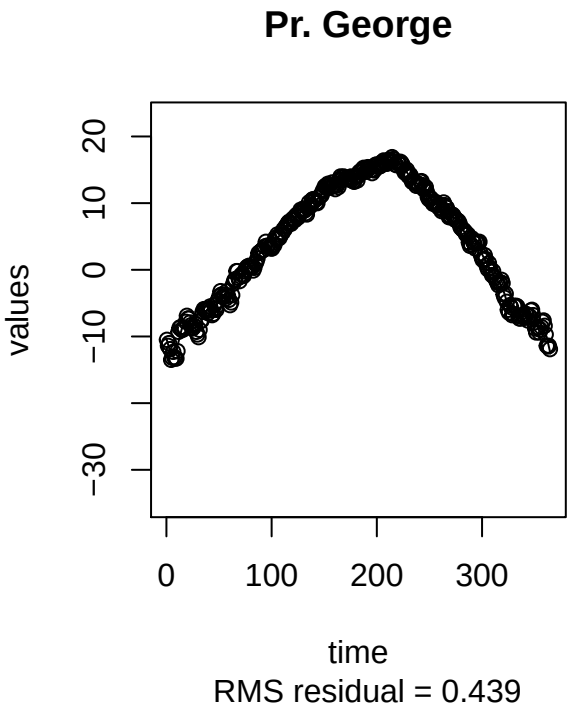

Edmonton
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.509

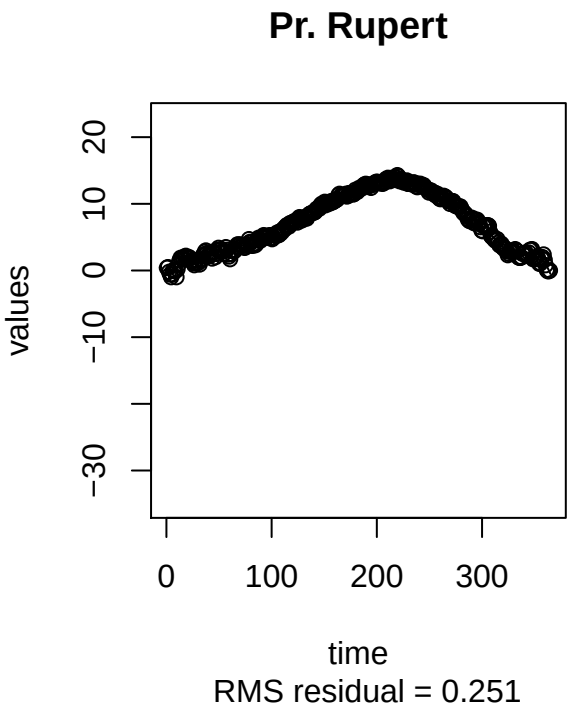

Calgary
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.539

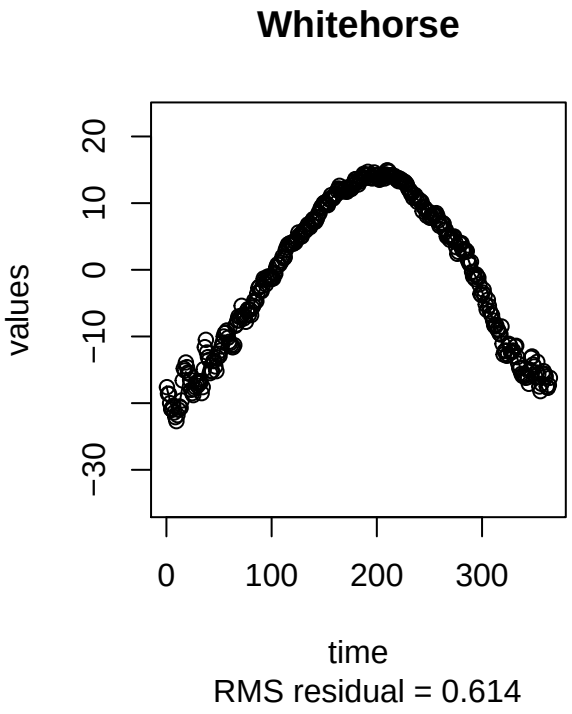

Kamloops
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.359

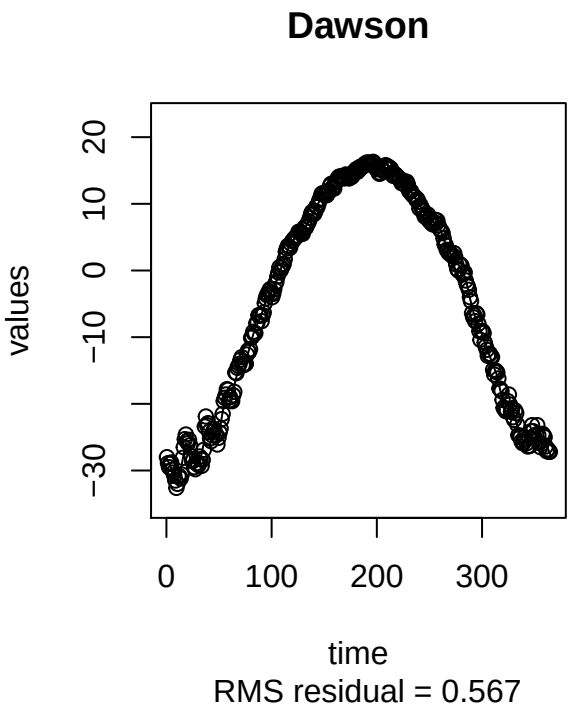

Vancouver
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.234

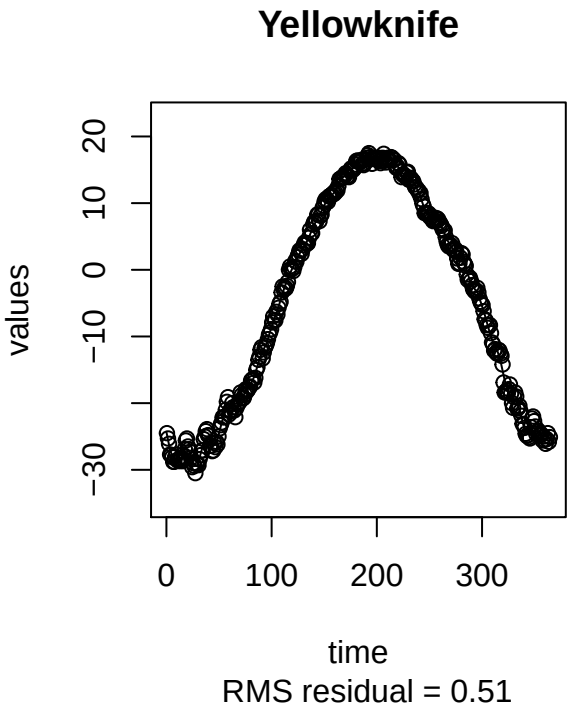

Victoria
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.256

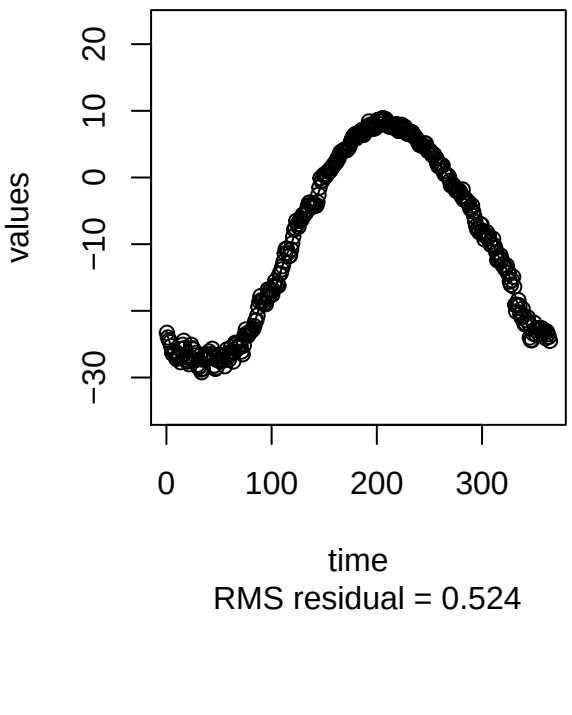

Pr. George
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.439

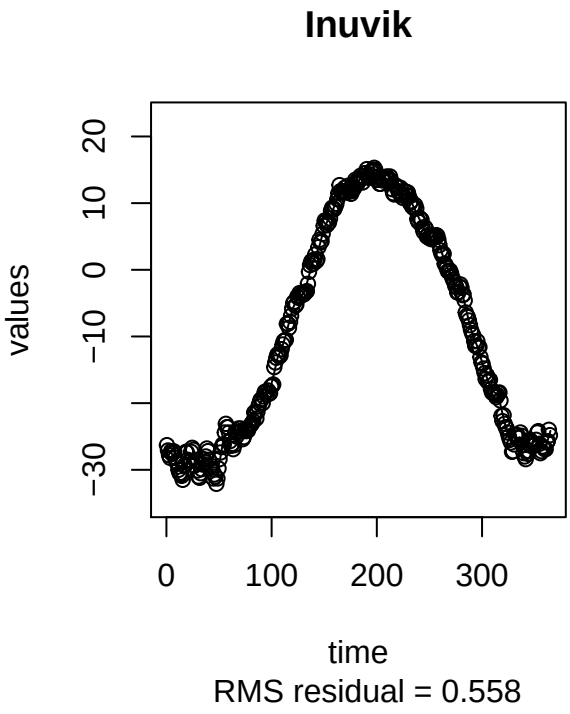

Pr. Rupert
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.251

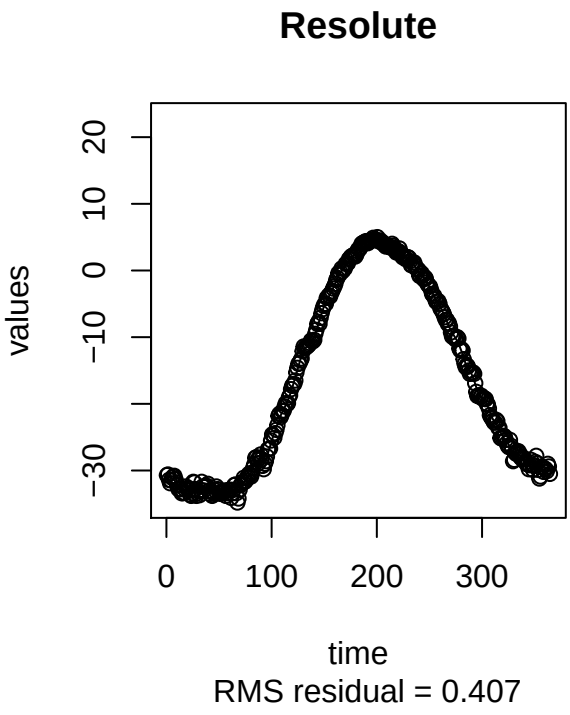

Whitehorse
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.614


Dawson
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.567


Yellowknife
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.51


Iqaluit
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.524





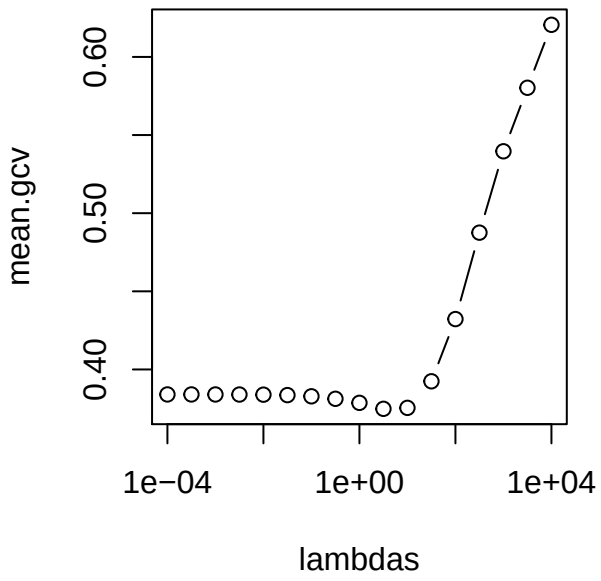

```
tempSmooth$df
```

```
## [1] 60.34489
```

```
tempSmooth$gcv
```

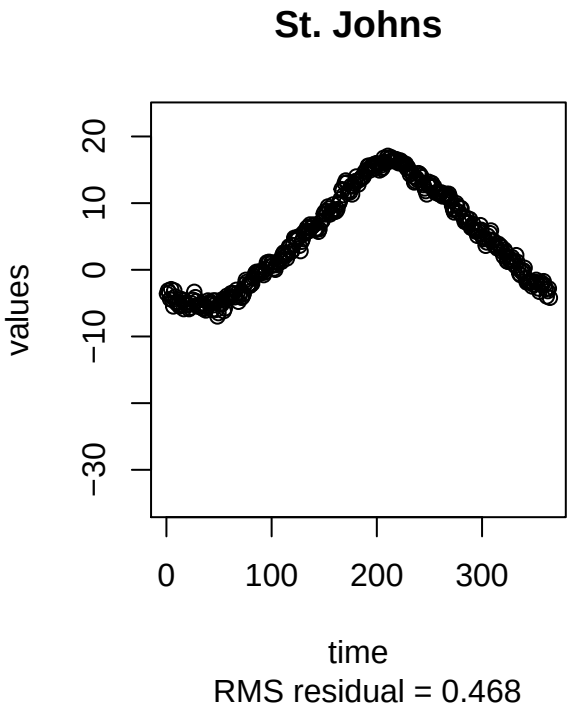
```
##    St. Johns      Halifax       Sydney     Yarmouth   Charlottvl
##   0.33366601   0.33181904   0.33863543   0.21215624   0.38179744
##  Fredericton  Scheffervll       Arvida  Bagottville       Quebec
##   0.39930264   0.48218881   0.61208772   0.45333179   0.39400313
##   Sherbrooke     Montreal       Ottawa      Toronto       London
##   0.59402231   0.40478730   0.40667244   0.36984013   0.36094282
##  Thunder Bay     Winnipeg      The Pas    Churchill       Regina
##   0.35912948   0.45752058   0.40410827   0.48797712   0.39422415
##   Pr. Albert Uranium City     Edmonton      Calgary     Kamloops
##   0.41420209   0.58133341   0.37116719   0.41693577   0.18535385
##    Vancouver     Victoria   Pr. George   Pr. Rupert   Whitehorse
##   0.07850927   0.09422677   0.27619925   0.09031481   0.54073730
##       Dawson  Yellowknife      Iqaluit       Inuvik     Resolute
##   0.46079938   0.37306056   0.39363018   0.44747960   0.23753704
```

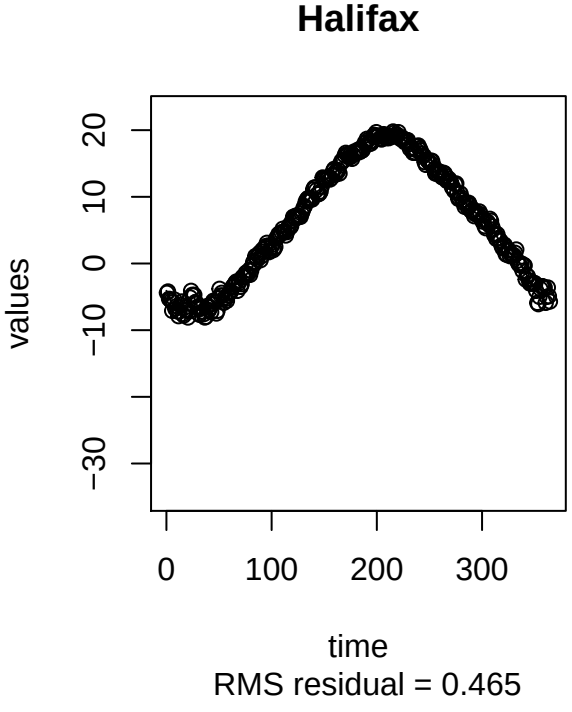
```
# Here the fit looks a lot better and the gcv values are much smaller.

#### 4. Choosing smoothing parameters

# We can search through a collection of smoothing parameters to try and find
# an optimal parameter.

# We will record the average gcv and choose lambda to be the minimum of these.

lambdas = 10^seq(-4,4,by=0.5)    # lambdas to look over

mean.gcv = rep(0,length(lambdas)) # store mean gcv


for(ilam in 1:length(lambdas)){
  # Set lambda
  curv.fdPari = curv.fdPar
  curv.fdPari$lambda = lambdas[ilam]

  # Smooth
  tempSmoothi = smooth.basis(daytime,temp,curv.fdPari)

  # Record average gcv
  mean.gcv[ilam] = mean(tempSmoothi$gcv)
}

# We can plot what we have

plot(lambdas,mean.gcv,type='b',log='x')
```

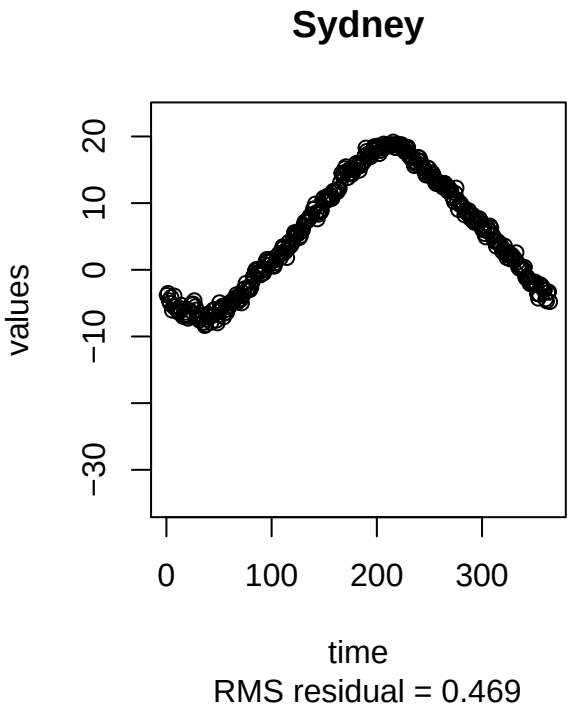


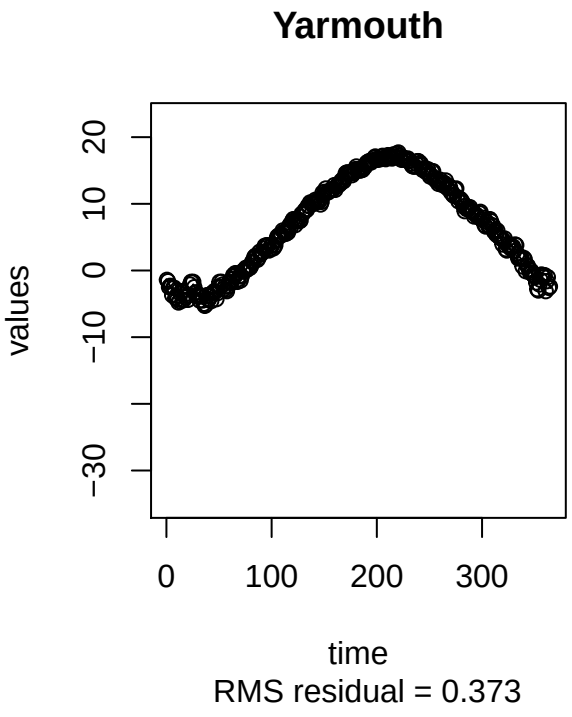

```
# Lets select the lowest of these and smooth

best = which.min(mean.gcv)
lambdabest = lambdas[best]

curv.fdPar$lambda = lambdabest
tempSmooth = smooth.basis(daytime,temp,curv.fdPar)

# And look at the same statistics

plotfit.fd(temp,daytime,tempSmooth$fd)
```

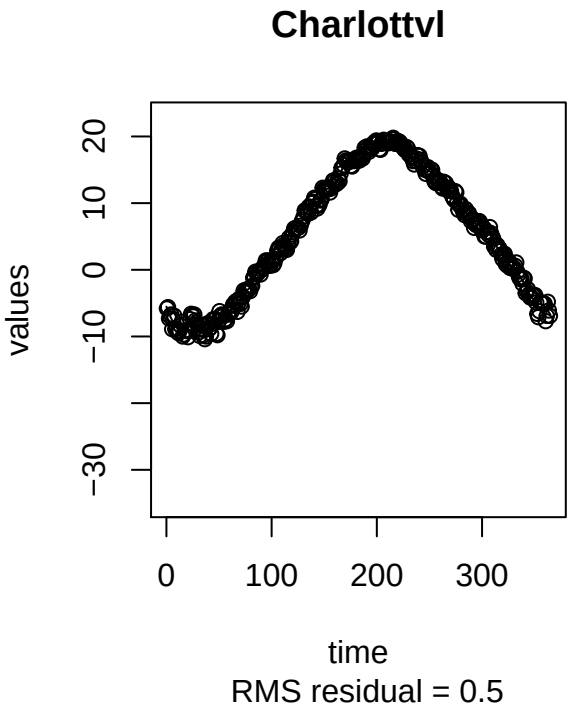
```
## Multiple plots:  Click in the plot to advance to the next plot
```

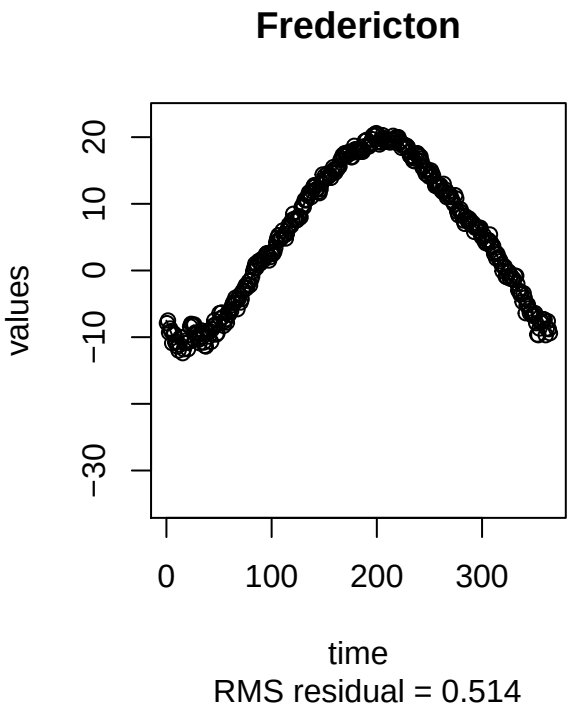

St. Johns
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.468

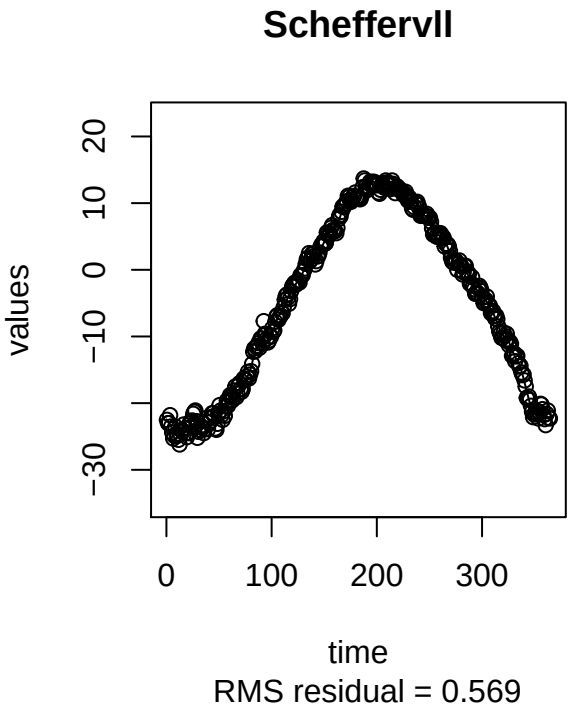

Halifax
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.465

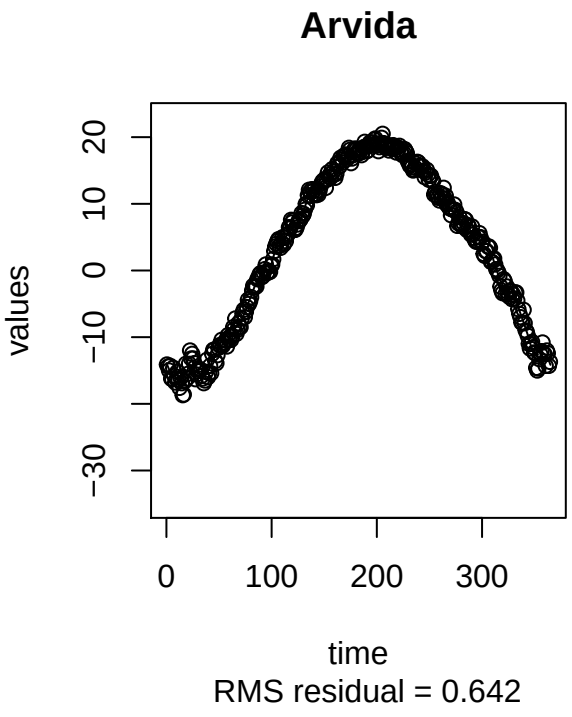

Sydney
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.469

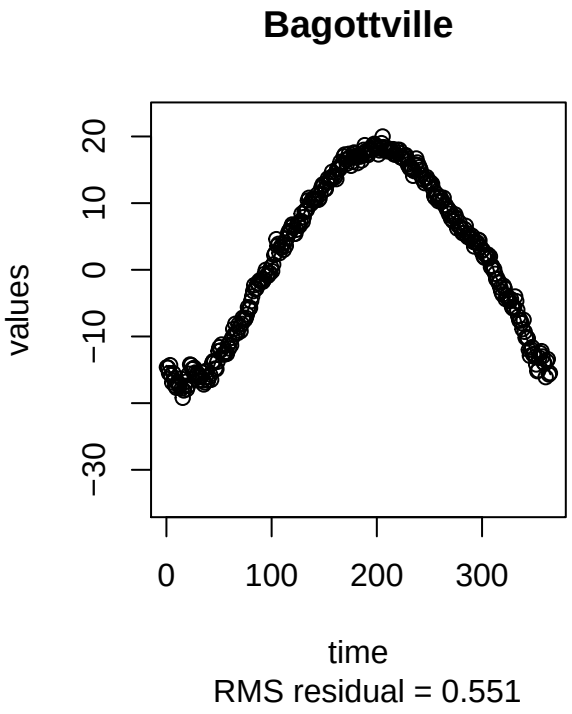

Yarmouth
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.373

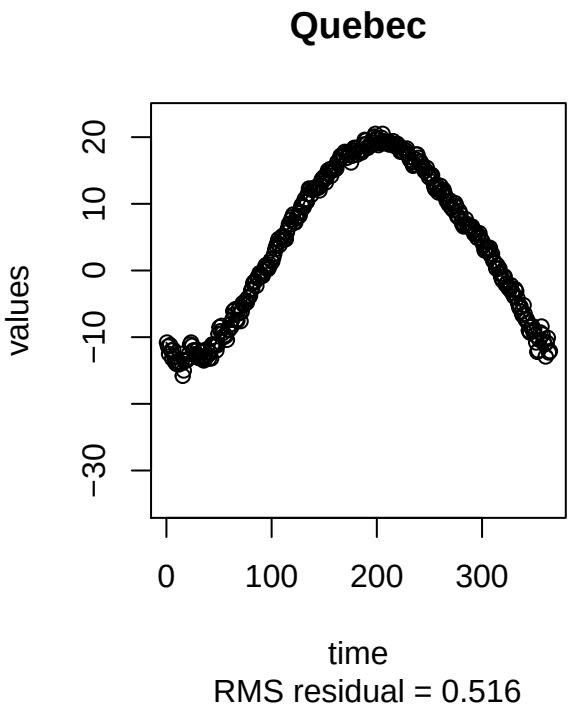

Charlottvl
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.5

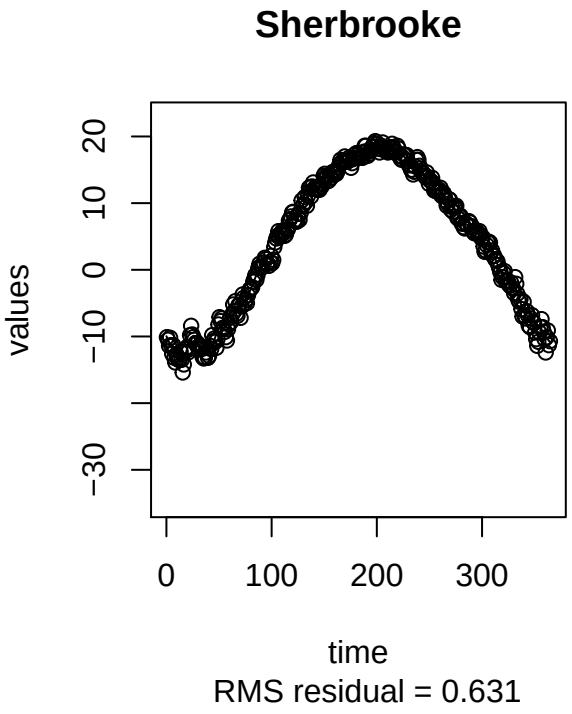

Fredericton
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.514

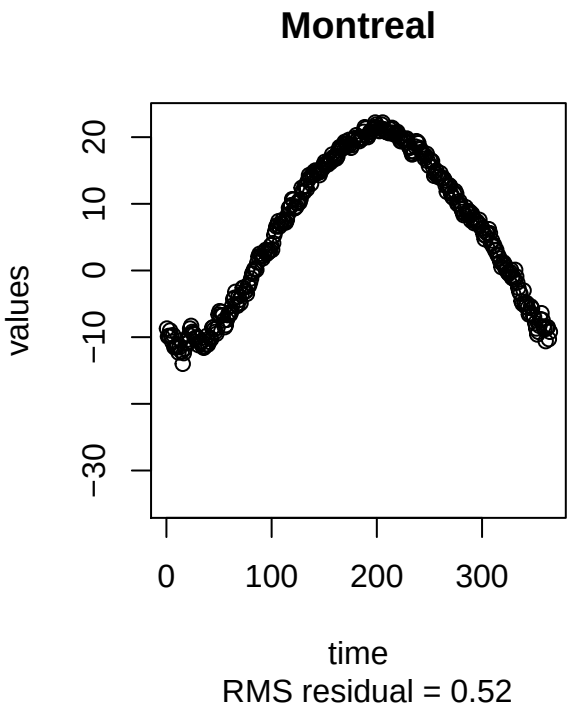

Scheffervll
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.569

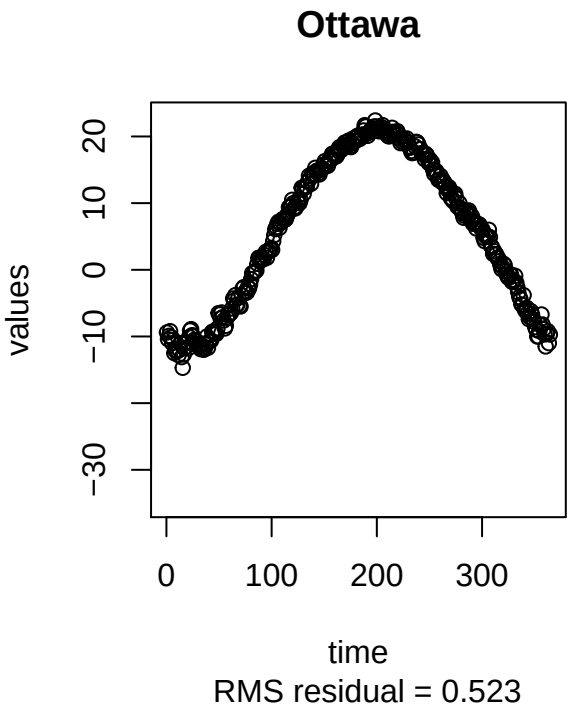

Arvida
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.642

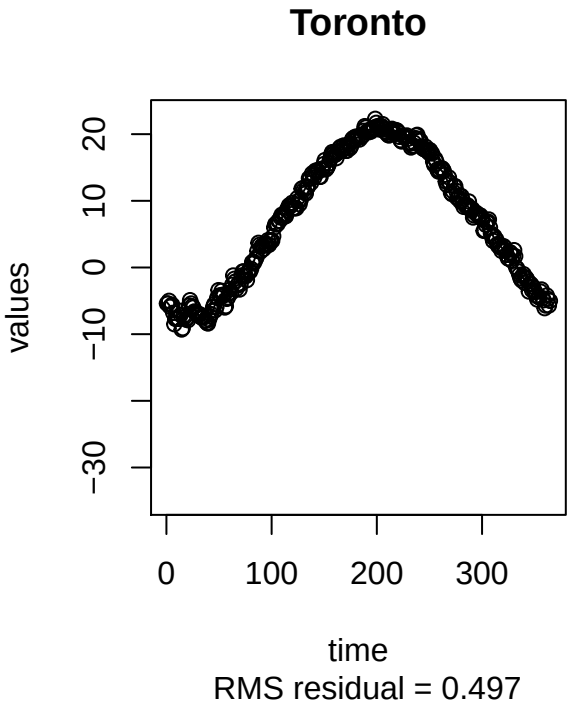

Bagottville
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.551

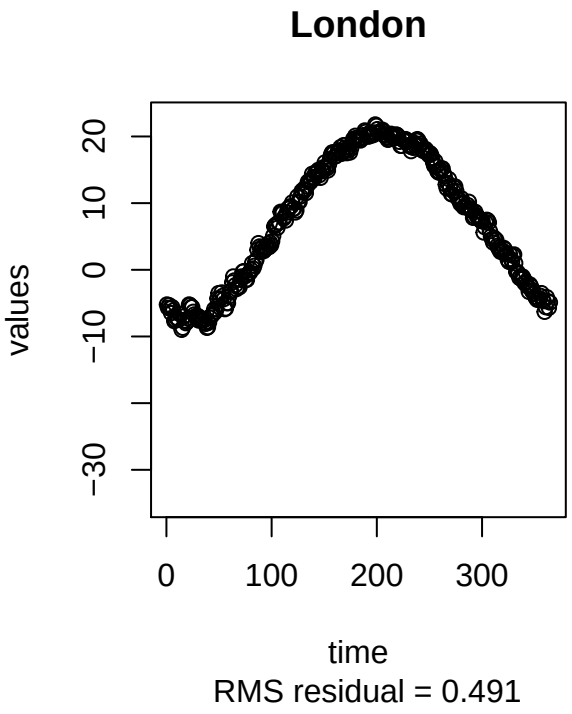

Quebec
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.516

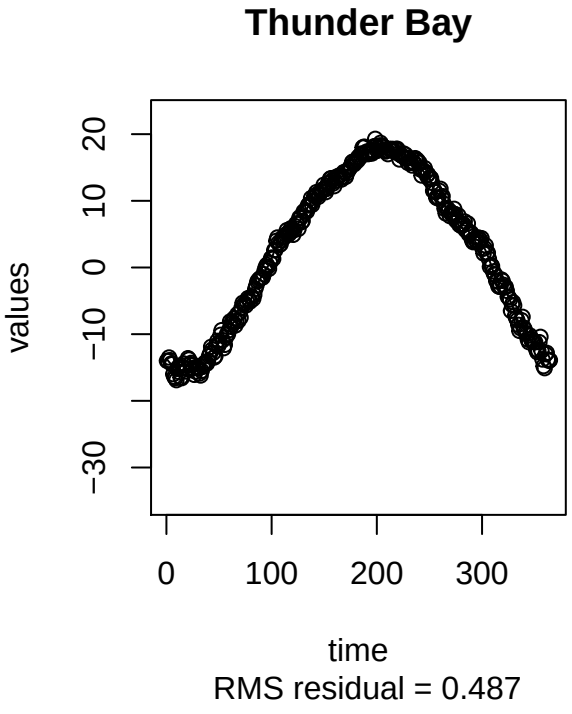

Sherbrooke
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.631

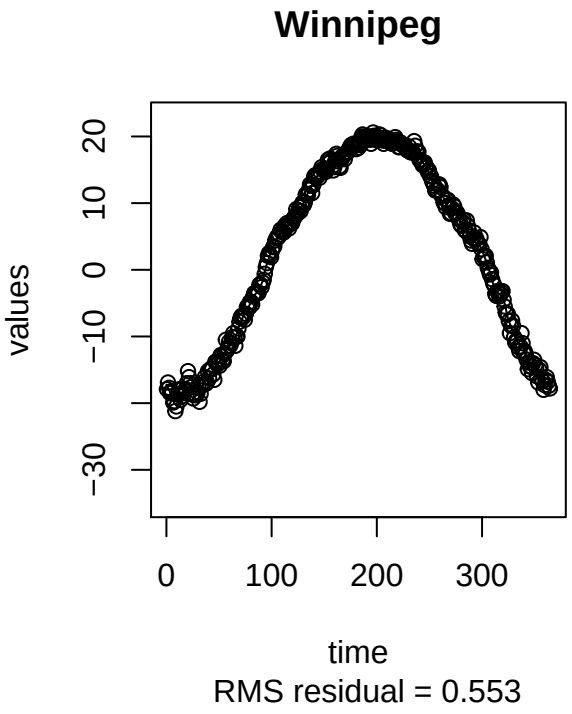

Montreal
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.52

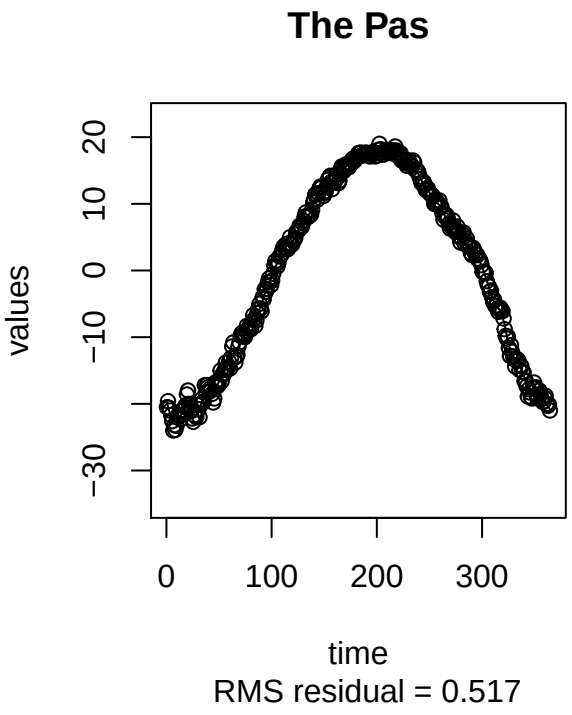

Ottawa
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.523

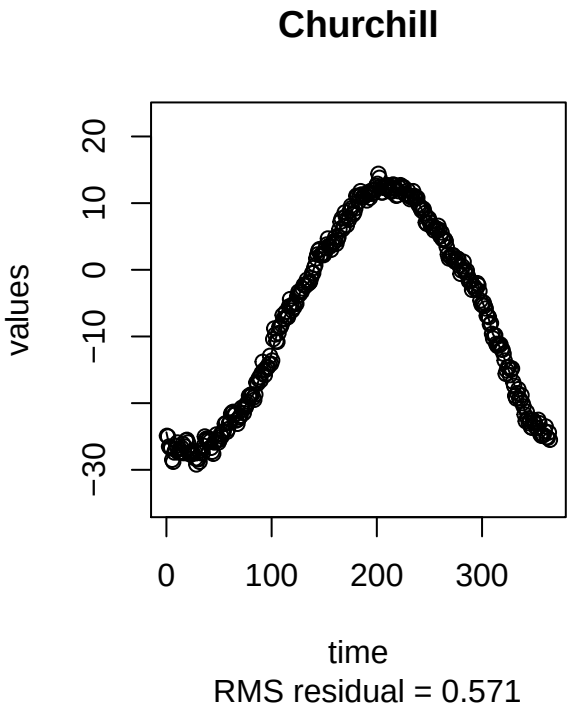

Toronto
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.497

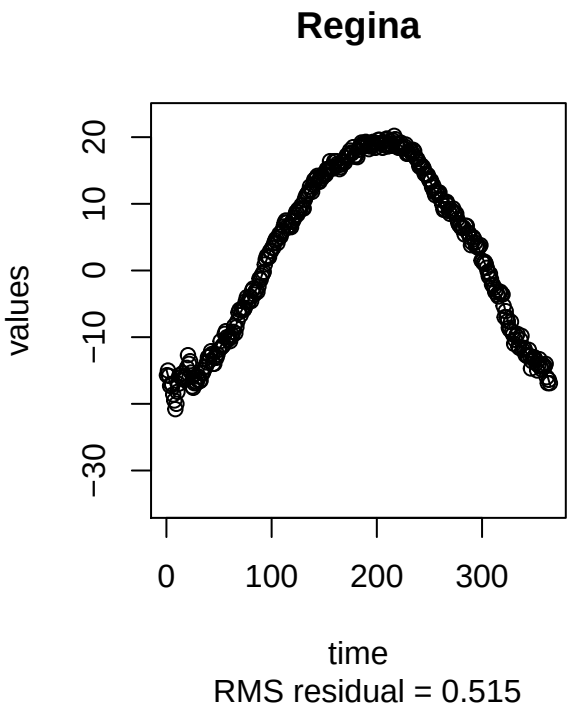

London
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.491

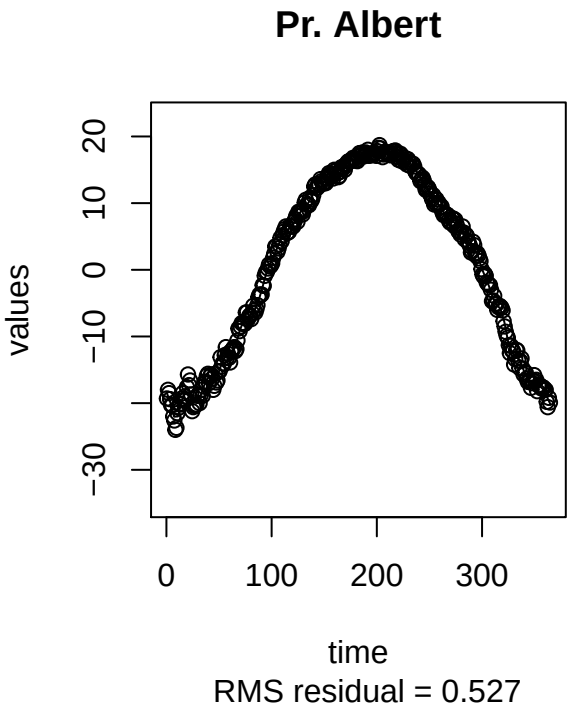

Thunder Bay
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.487

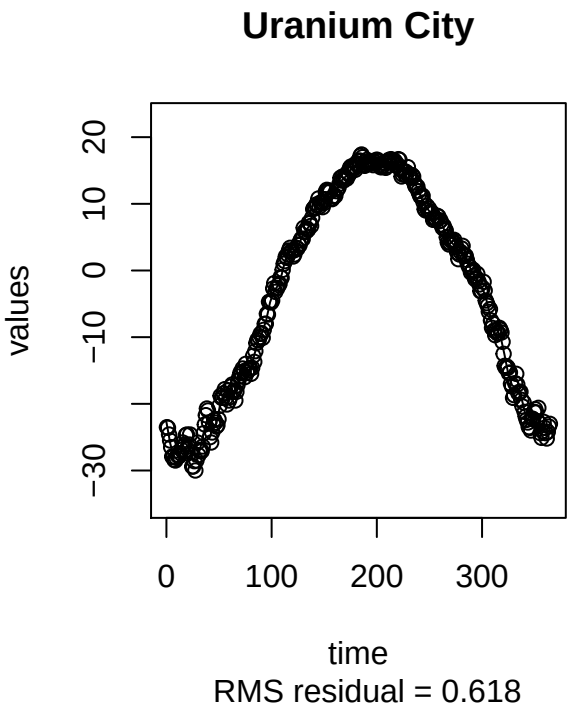

Winnipeg
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.553

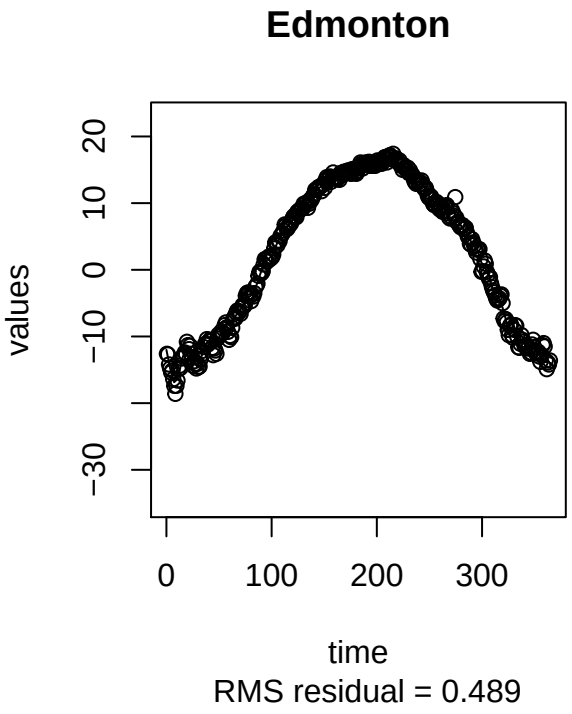

The Pas
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.517

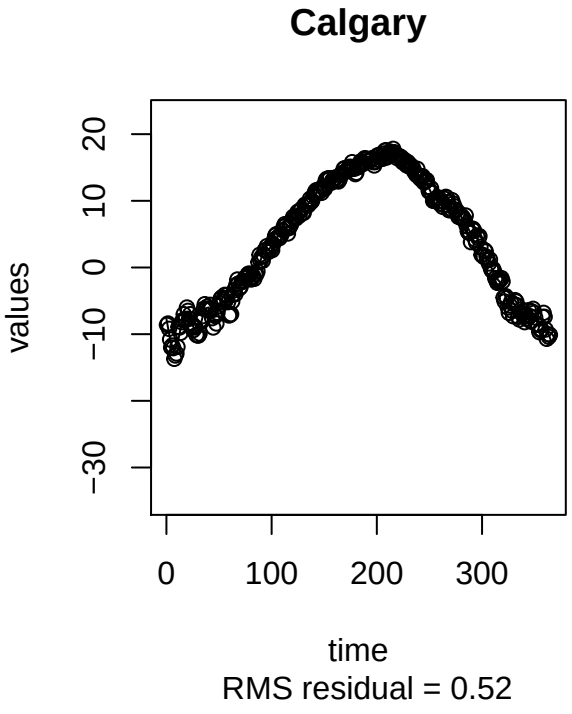

Churchill
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.571

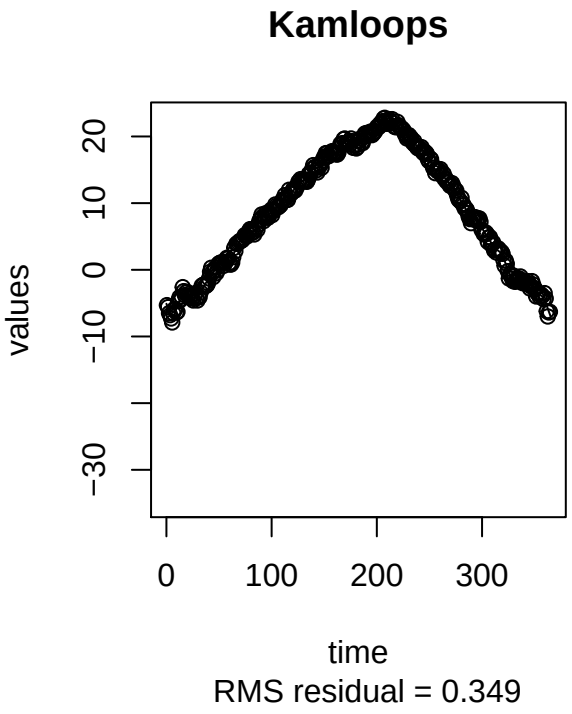

Regina
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.515

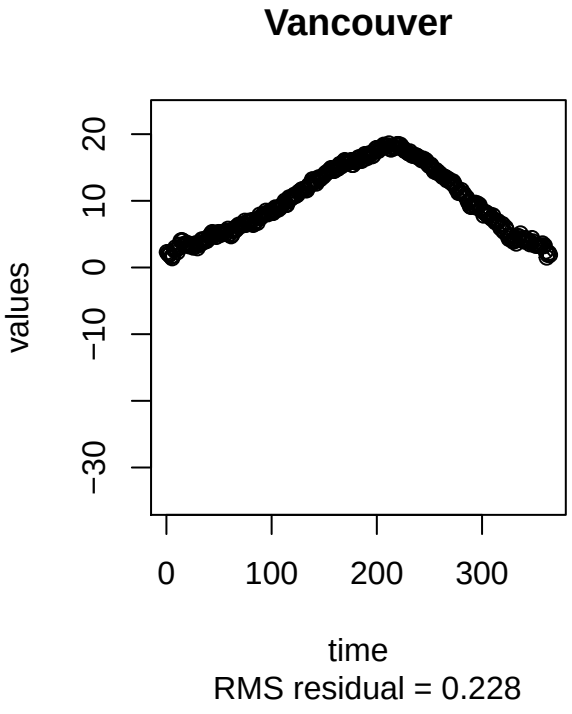

Pr. Albert
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.527

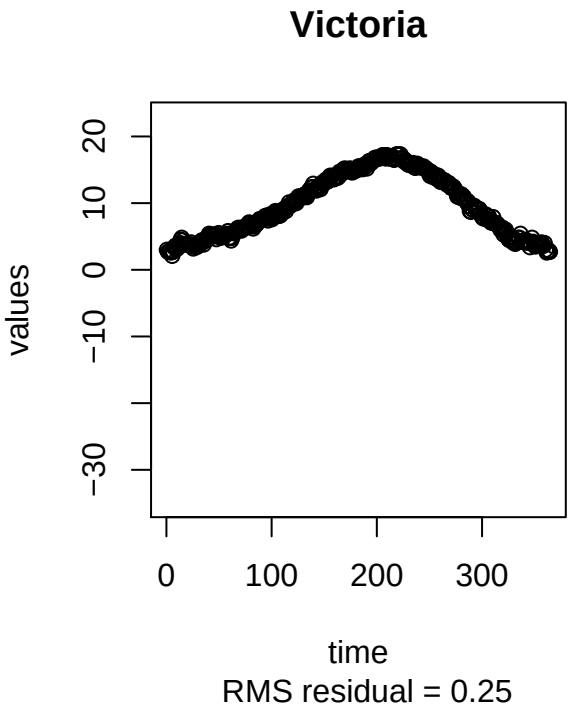

Uranium City
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.618

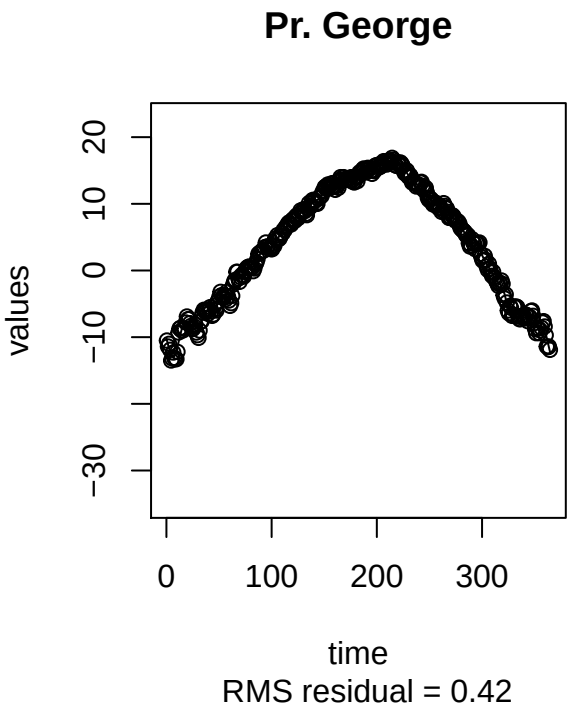

Edmonton
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.489

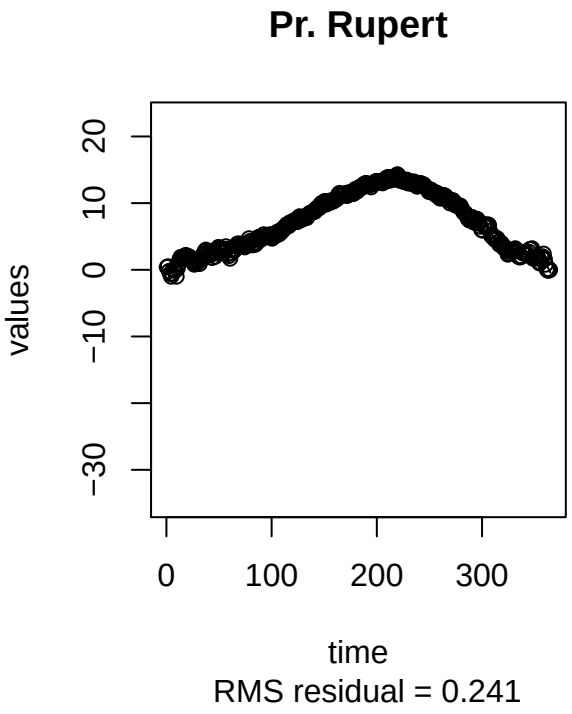

Calgary
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.52

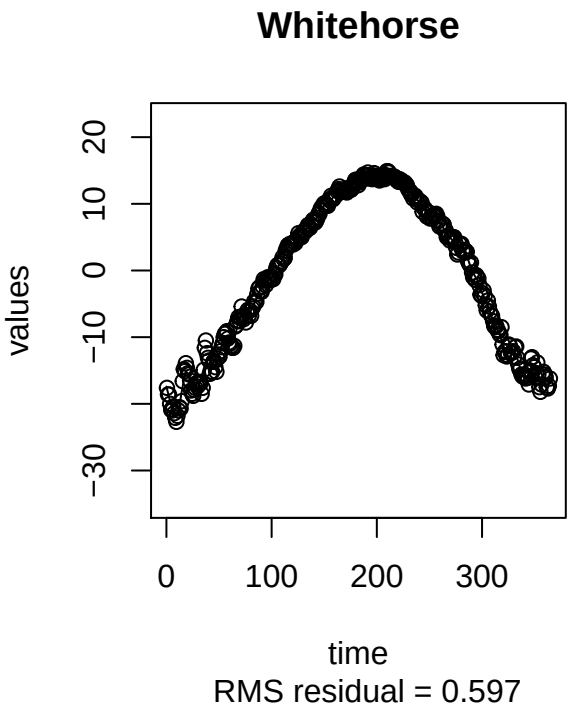

Kamloops
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.349

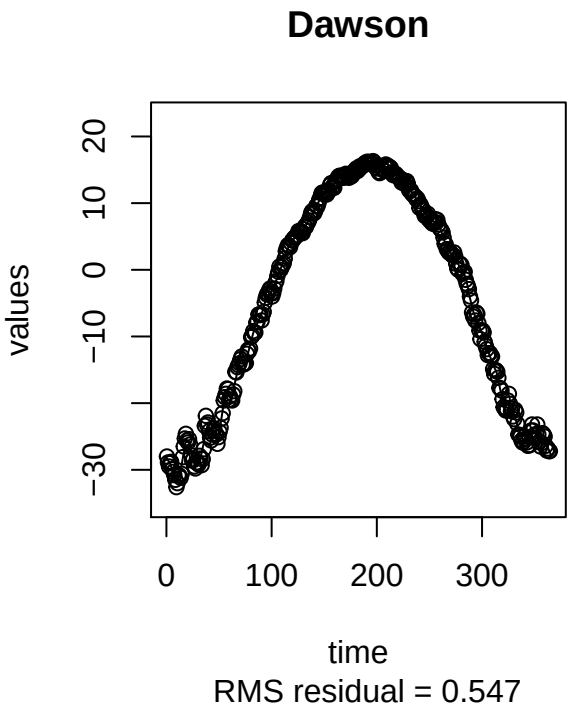

Vancouver
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.228

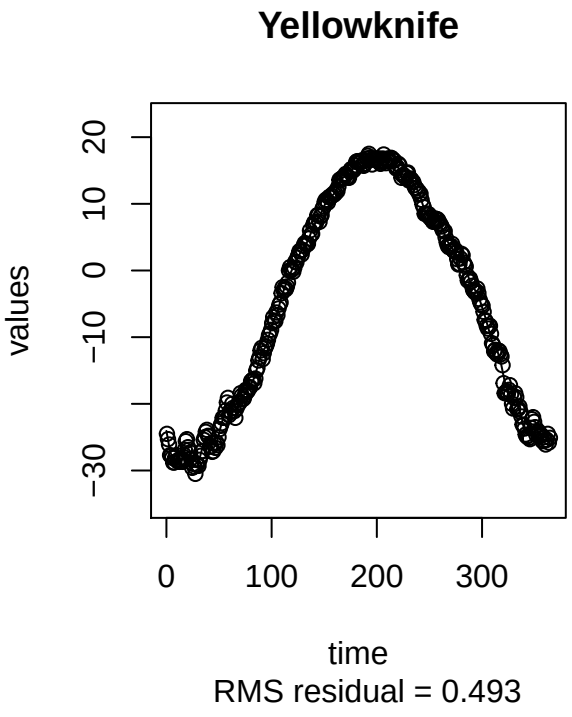

Victoria
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.25

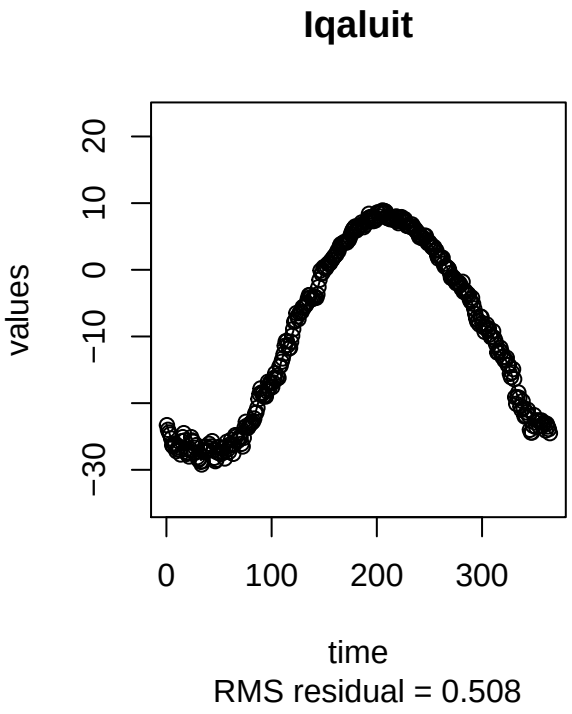

Pr. George
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.42

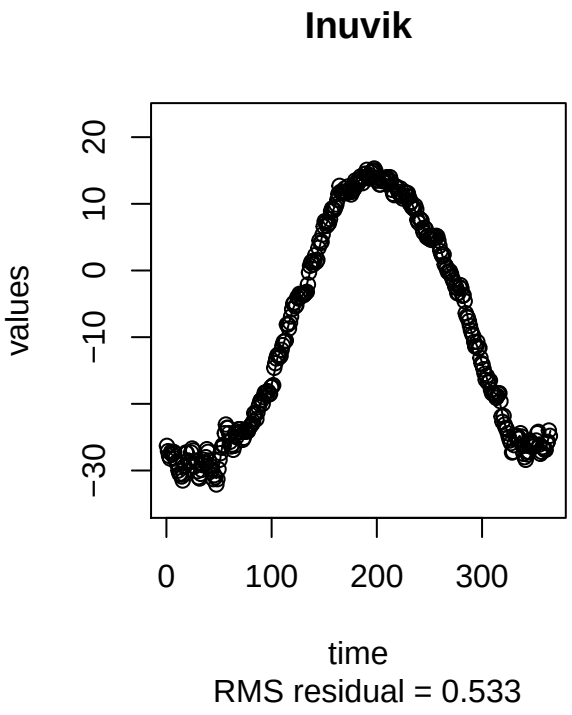

Pr. Rupert
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.241

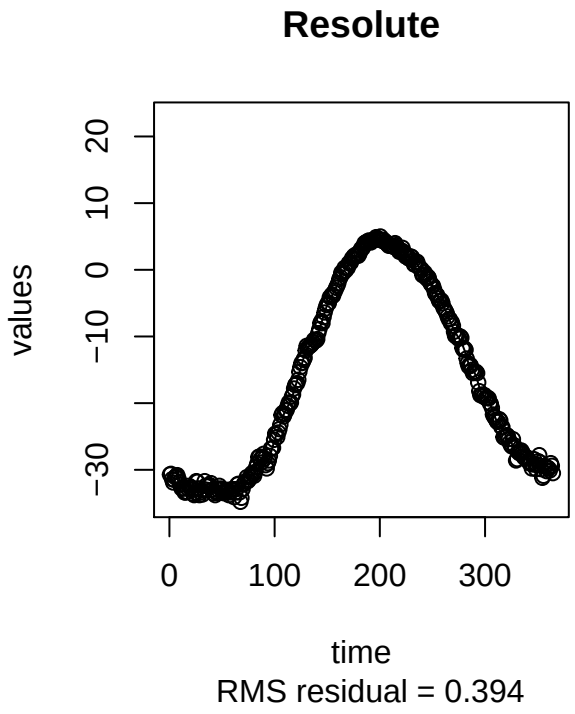

Whitehorse
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.597


Dawson
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.547


Yellowknife
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.493


Iqaluit
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.508

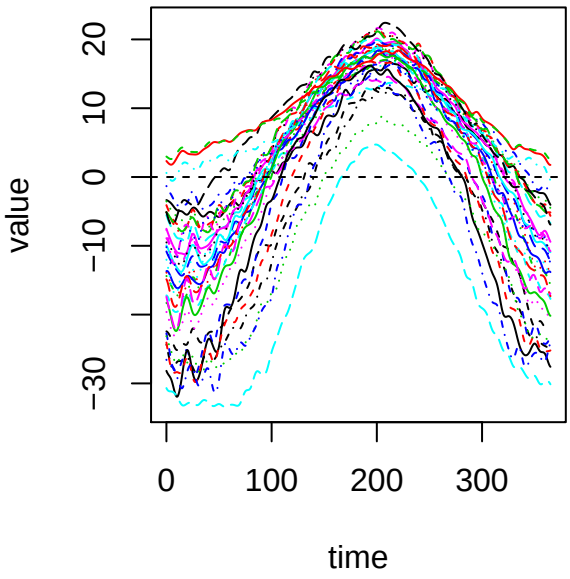

Inuvik
values
20
10
0
−10
−30
0
100
200
300
time
RMS residual = 0.533




```
tempSmooth$df
```

```
## [1] 68.086
```

```
# We'll also plot these

plot(tempSmooth)
```




```
## [1] "done"
```

```
### EXERCISE: try obtaining a smooth of the precipitation data

### EXERCISE: how much does the result change if the basis has a knot every day
### instead of every 5 days?


################################################################################
###       FUNCTIONAL DATA OBJECTS: MANIPULATION AND STATISTICS              ####
################################################################################

## Now that we have a functional data object we can manipulate them in various
# ways.  First let's extract the fd object

tempfd <- tempSmooth$fd

# if we look at what's in this we see

names(tempfd)
```

```
## [1] "coefs"   "basis"   "fdnames"
```

```
# We see a basis, plus coefficient matrix

dim(tempfd$coefs)
```

```
## [1] 76 35
```

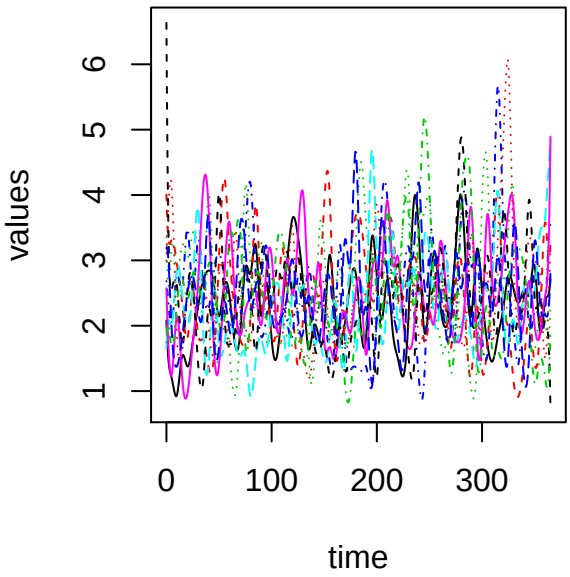
```
# and an array giving names
```

```
tempfd$fdnames
```

```
## $time
##   [1] "jan01" "jan02" "jan03" "jan04" "jan05" "jan06" "jan07" "jan08"
##   [9] "jan09" "jan10" "jan11" "jan12" "jan13" "jan14" "jan15" "jan16"
##  [17] "jan17" "jan18" "jan19" "jan20" "jan21" "jan22" "jan23" "jan24"
##  [25] "jan25" "jan26" "jan27" "jan28" "jan29" "jan30" "jan31" "feb01"
##  [33] "feb02" "feb03" "feb04" "feb05" "feb06" "feb07" "feb08" "feb09"
##  [41] "feb10" "feb11" "feb12" "feb13" "feb14" "feb15" "feb16" "feb17"
##  [49] "feb18" "feb19" "feb20" "feb21" "feb22" "feb23" "feb24" "feb25"
##  [57] "feb26" "feb27" "feb28" "mar01" "mar02" "mar03" "mar04" "mar05"
##  [65] "mar06" "mar07" "mar08" "mar09" "mar10" "mar11" "mar12" "mar13"
##  [73] "mar14" "mar15" "mar16" "mar17" "mar18" "mar19" "mar20" "mar21"
##  [81] "mar22" "mar23" "mar24" "mar25" "mar26" "mar27" "mar28" "mar29"
##  [89] "mar30" "mar31" "apr01" "apr02" "apr03" "apr04" "apr05" "apr06"
##  [97] "apr07" "apr08" "apr09" "apr10" "apr11" "apr12" "apr13" "apr14"
## [105] "apr15" "apr16" "apr17" "apr18" "apr19" "apr20" "apr21" "apr22"
## [113] "apr23" "apr24" "apr25" "apr26" "apr27" "apr28" "apr29" "apr30"
## [121] "may01" "may02" "may03" "may04" "may05" "may06" "may07" "may08"
## [129] "may09" "may10" "may11" "may12" "may13" "may14" "may15" "may16"
## [137] "may17" "may18" "may19" "may20" "may21" "may22" "may23" "may24"
## [145] "may25" "may26" "may27" "may28" "may29" "may30" "may31" "jun01"
## [153] "jun02" "jun03" "jun04" "jun05" "jun06" "jun07" "jun08" "jun09"
## [161] "jun10" "jun11" "jun12" "jun13" "jun14" "jun15" "jun16" "jun17"
## [169] "jun18" "jun19" "jun20" "jun21" "jun22" "jun23" "jun24" "jun25"
## [177] "jun26" "jun27" "jun28" "jun29" "jun30" "jul01" "jul02" "jul03"
## [185] "jul04" "jul05" "jul06" "jul07" "jul08" "jul09" "jul10" "jul11"
## [193] "jul12" "jul13" "jul14" "jul15" "jul16" "jul17" "jul18" "jul19"
## [201] "jul20" "jul21" "jul22" "jul23" "jul24" "jul25" "jul26" "jul27"
## [209] "jul28" "jul29" "jul30" "jul31" "aug01" "aug02" "aug03" "aug04"
## [217] "aug05" "aug06" "aug07" "aug08" "aug09" "aug10" "aug11" "aug12"
## [225] "aug13" "aug14" "aug15" "aug16" "aug17" "aug18" "aug19" "aug20"
## [233] "aug21" "aug22" "aug23" "aug24" "aug25" "aug26" "aug27" "aug28"
## [241] "aug29" "aug30" "aug31" "sep01" "sep02" "sep03" "sep04" "sep05"
## [249] "sep06" "sep07" "sep08" "sep09" "sep10" "sep11" "sep12" "sep13"
## [257] "sep14" "sep15" "sep16" "sep17" "sep18" "sep19" "sep20" "sep21"
## [265] "sep22" "sep23" "sep24" "sep25" "sep26" "sep27" "sep28" "sep29"
## [273] "sep30" "oct01" "oct02" "oct03" "oct04" "oct05" "oct06" "oct07"
## [281] "oct08" "oct09" "oct10" "oct11" "oct12" "oct13" "oct14" "oct15"
## [289] "oct16" "oct17" "oct18" "oct19" "oct20" "oct21" "oct22" "oct23"
## [297] "oct24" "oct25" "oct26" "oct27" "oct28" "oct29" "oct30" "oct31"
## [305] "nov01" "nov02" "nov03" "nov04" "nov05" "nov06" "nov07" "nov08"
## [313] "nov09" "nov10" "nov11" "nov12" "nov13" "nov14" "nov15" "nov16"
## [321] "nov17" "nov18" "nov19" "nov20" "nov21" "nov22" "nov23" "nov24"
## [329] "nov25" "nov26" "nov27" "nov28" "nov29" "nov30" "dec01" "dec02"
## [337] "dec03" "dec04" "dec05" "dec06" "dec07" "dec08" "dec09" "dec10"
## [345] "dec11" "dec12" "dec13" "dec14" "dec15" "dec16" "dec17" "dec18"
## [353] "dec19" "dec20" "dec21" "dec22" "dec23" "dec24" "dec25" "dec26"
## [361] "dec27" "dec28" "dec29" "dec30" "dec31"
##
## $reps
##  [1] "St. Johns"    "Halifax"      "Sydney"       "Yarmouth"
##  [5] "Charlottvl"   "Fredericton"  "Scheffervll"  "Arvida"
##  [9] "Bagottville"  "Quebec"       "Sherbrooke"   "Montreal"
```

```
## [13] "Ottawa"       "Toronto"      "London"       "Thunder Bay"
## [17] "Winnipeg"     "The Pas"      "Churchill"    "Regina"
## [21] "Pr. Albert"   "Uranium City" "Edmonton"     "Calgary"
## [25] "Kamloops"     "Vancouver"    "Victoria"     "Pr. George"
## [29] "Pr. Rupert"   "Whitehorse"   "Dawson"       "Yellowknife"
## [33] "Iqaluit"      "Inuvik"       "Resolute"
##
## $values
## [1] "value"
```

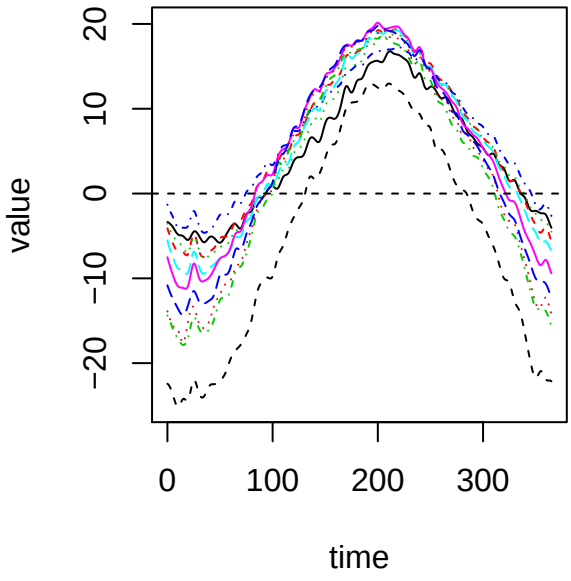
```
# With lists giving names for time points, replicates and dimensions. Each list
# also has a name that can be used in plotting.  Apart from plotting functions,
# fdnames isn't used and you can generally ignore it.

# We can also create fd objects by adding a basis and a coefficient array. Let's
# make a random one, say

newcoefs = matrix(rgamma(nbasis*10,5,2),nbasis,10)
newfd = fd(newcoefs,bbasis)

# Notice that we haven't specified fdnames.

# The plotting command nicely draws these.

plot(newfd)
```




```
## [1] "done"
```

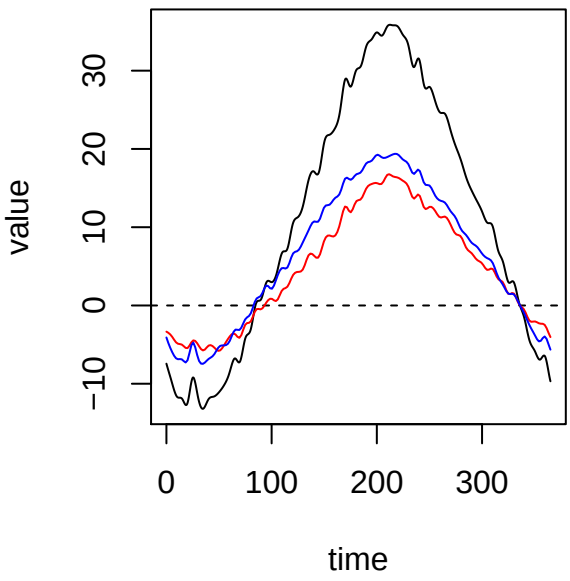
```
# Not that this looks very nice; we'll stick with the Canadian weather data.
```

```
#### 1. Manipulation

# We can do a number of things with these functions, treating them as data.
# These operations all result in new functional data objects, but we will plot
# them directly as an illustration.

# Subset

plot(tempfd[1:10])
```

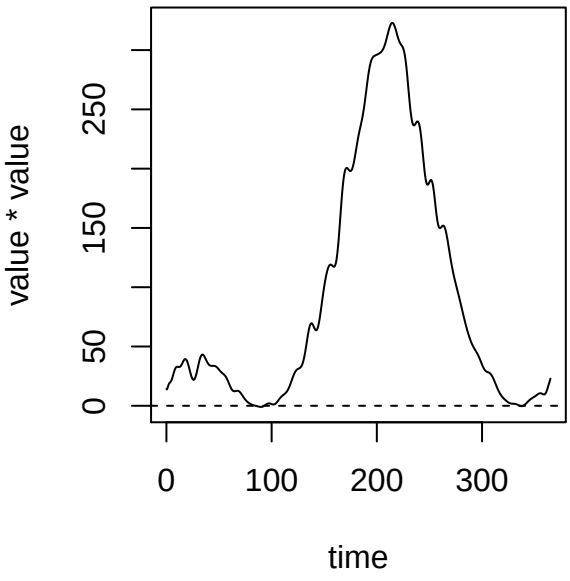



```
## [1] "done"
# We can add them together; the 'lines' function also works with them

newfd = tempfd[1] + tempfd[2]
plot(newfd)
```

```
## [1] "done"
lines(tempfd[1],col=2)
lines(tempfd[2],col=4)
```

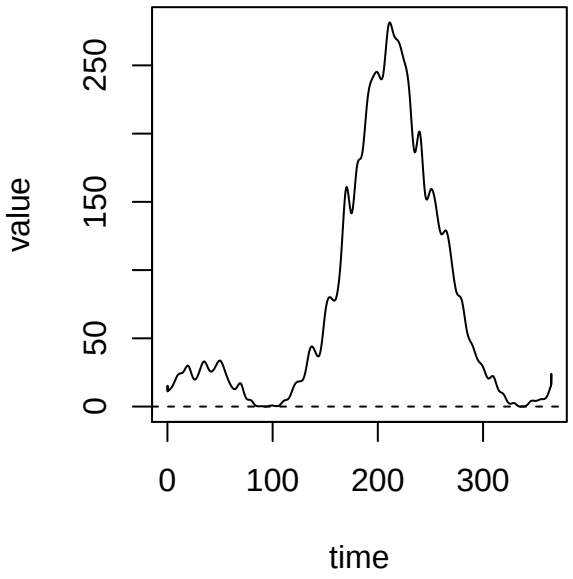



```
# We can also multiply

plot(tempfd[1]*tempfd[2])
```



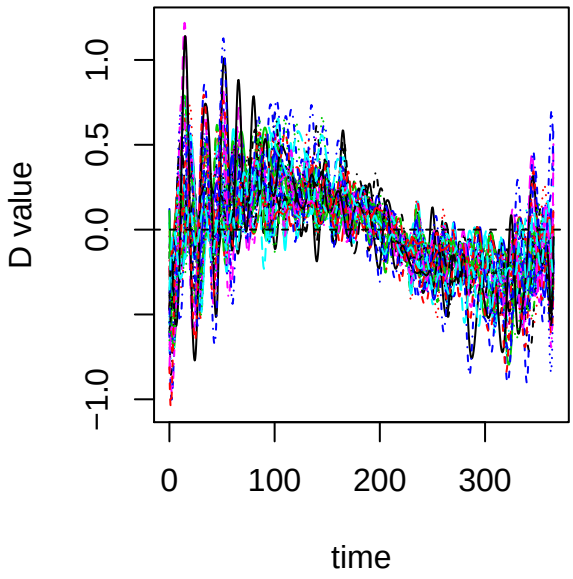

```
## [1] "done"
```

```r
# And obtain powers

plot(tempfd[1]^2)
```



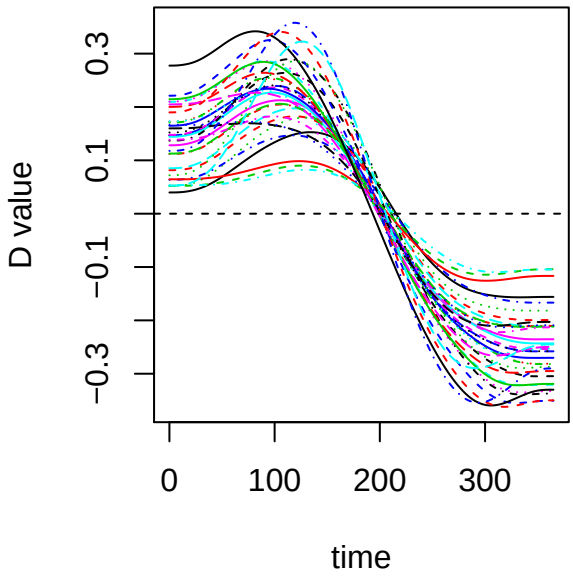

```
## [1] "done"
```

```r
# We can also obtain derivatives

plot(deriv.fd(tempfd))
```



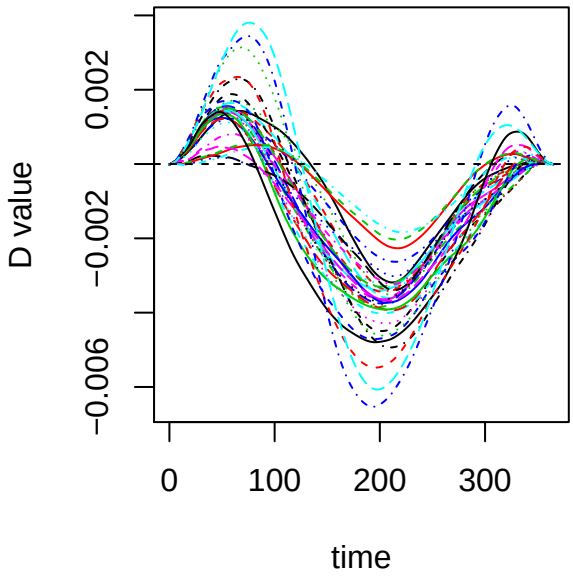

```
## [1] "done"
```

```
# These are pretty wild because of the roughness of the resulting curves
# instead let's have a look at the over-smoothed data:

plot(deriv.fd(tempSmooth1$fd))
```




```
## [1] "done"
```

```
# We can also look at second derivatives

plot(deriv.fd(tempSmooth1$fd,2))
```

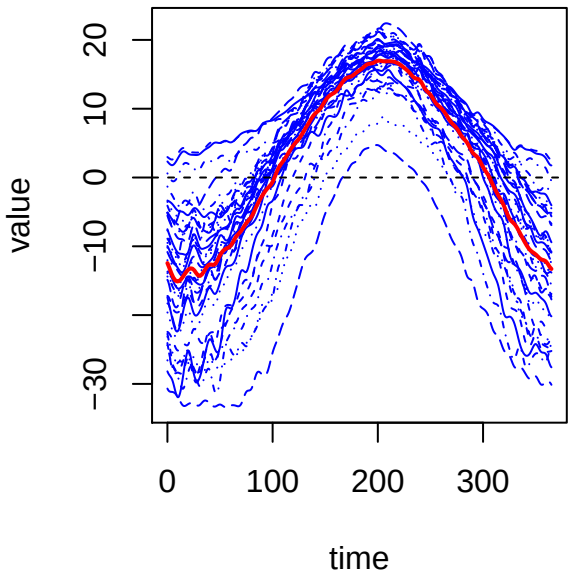


```
## [1] "done"
```

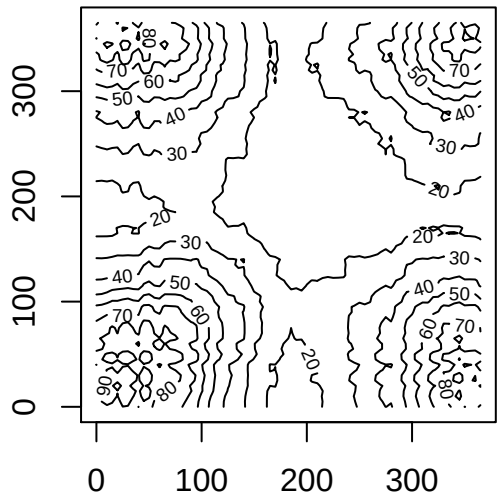
```
# Note that it is a property of B-splines of order m that the (m-2)th derivative
# is zero at the end of the interval.

#### 2. Summary statistics

# The obvious thing to look at is the mean

mtempfd = mean(tempfd)
plot(tempfd,col=4)
```

```
## [1] "done"
```

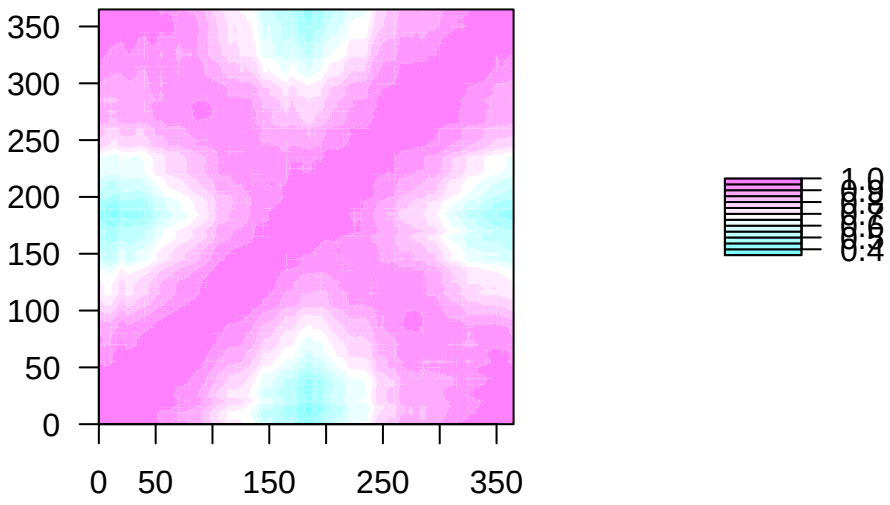
```
lines(mtempfd,lwd=2,col=2)
```



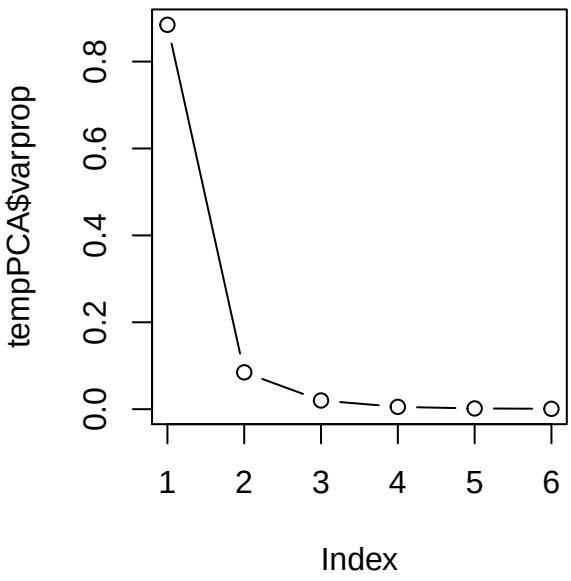

```
# We can also examine a variance

temp.varbifd = var.fd(tempfd)

# temp.varbifd is a bivariate functional data object -- meaning it takes values
# on a rectangle.

# To plot this, we need to evaluate it; here we'll use day5 -- 365 points is
# a bit overkill.

temp.var = eval.bifd(day5,day5,temp.varbifd)
contour(day5,day5,temp.var)
```



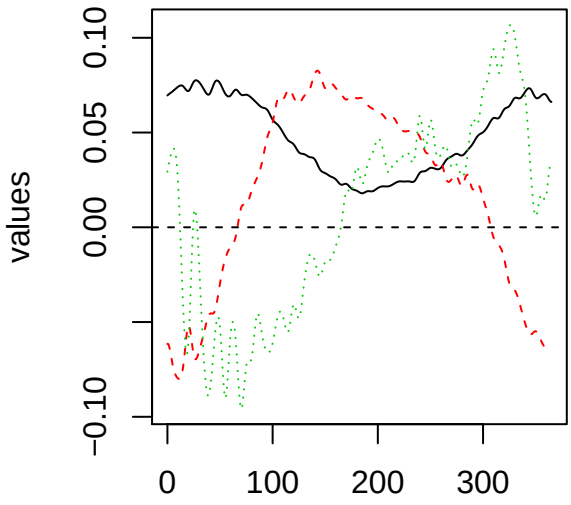

```
# Mostly high variance in the winter, low in summer. Let's have a look at
# correlation. In this case, evaluation arguments go in with the function call

temp.cor = cor.fd(day5,tempfd)
filled.contour(day5,day5,temp.cor)
```




```
# Here we see high correlation between Summer and Winter temperatures, but
# much less in the spring and fall (although spring to fall correlation is still
# high).

### EXERCISE: obtain these for the precipitation data and look at the covariance
### and correlation between temperature and precipitation.

### EXERCISE: try repeating the above with a Fourier basis and the harmonic
### acceleration penalty. Does this make much difference?


################################################################################
###                    FUNCTIONAL PRINCIPAL COMPONENTS                       ####
################################################################################

#### 1. pca.fd

# We can conduct a fPCA through

tempPCA = pca.fd(tempfd,nharm=6)

# Here we can look at proportion of variance explained:

plot(tempPCA$varprop,type='b')
```



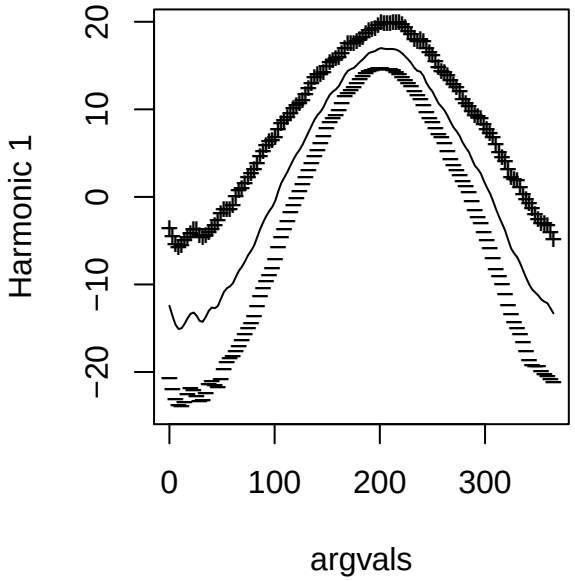

```
# Looks like we could have stopped at 3.

## Looking at the principal components:

plot(tempPCA$harmonics[1:3])
```



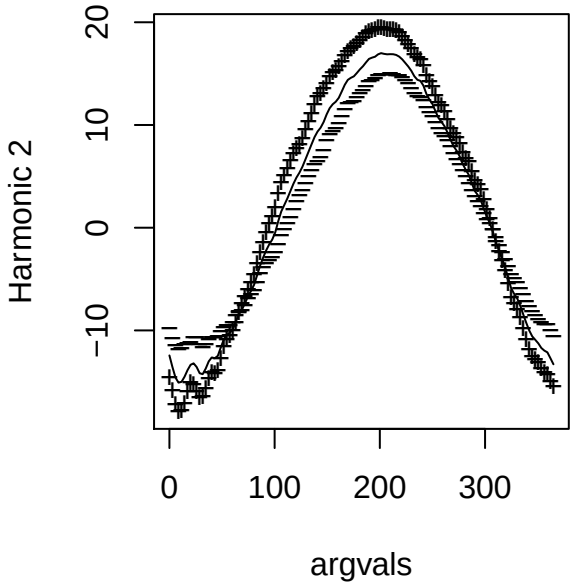

```
## [1] "done"
```

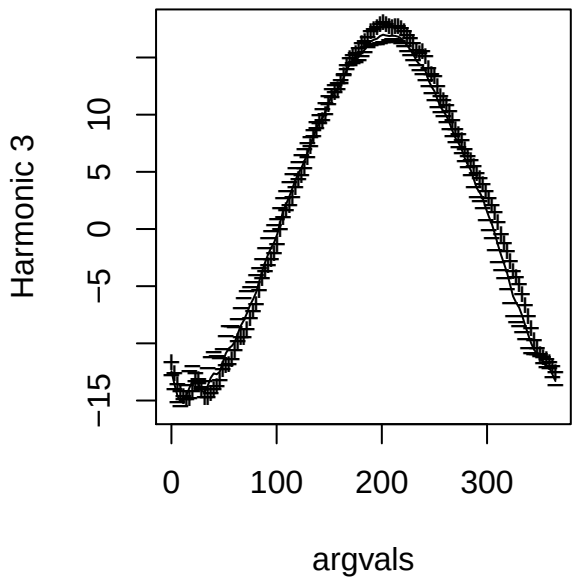
```
# 1 Looks like over-all temperature.
# 2 Looks like Summer vs Winter
# 3 Is Spring vs Fall.

## But let's plot these

plot(tempPCA,harm=1:3)
```

PCA function 1 (Percentage of variability 88.5)

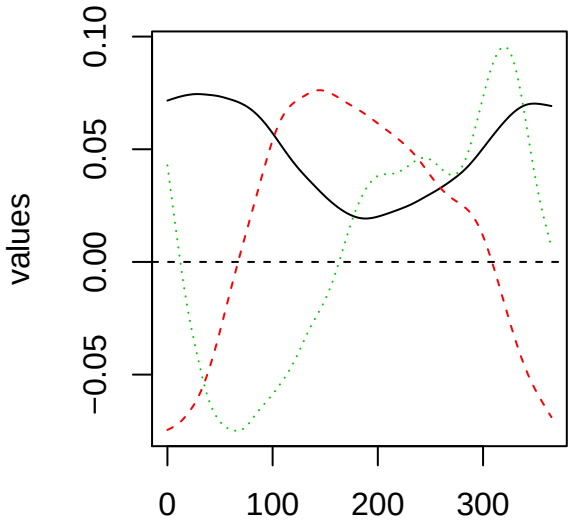


PCA function 2 (Percentage of variability 8.5)





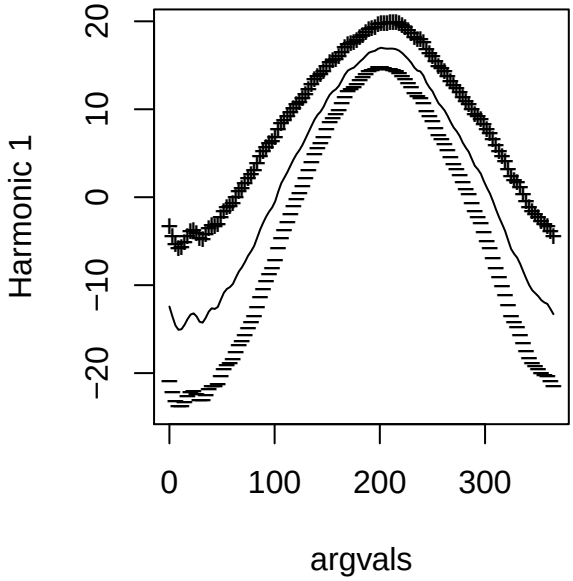

```
# Which gives a much better picture of what's going on.

##### 2. PCA and Smoothing

# The PCs above are fairly rough, we can also add a smoothing penalty (see
# the special topics slides).

pca.fdPar = fdPar(bbasis,curv.Lfd,1e4)

tempPCAsmooth = pca.fd(tempfd,nharm=6,harmfdPar=pca.fdPar)

# Let's plot the PCs

plot(tempPCAsmooth$harmonics[1:3])
```



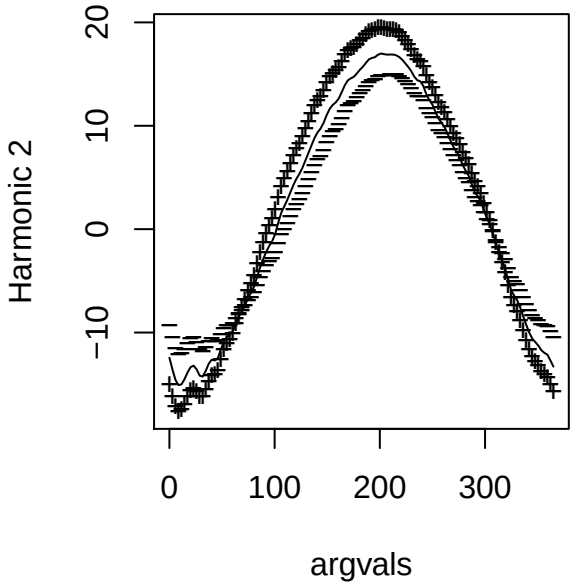

```
## [1] "done"
```

```
# We can see that these are considerably smoother but still have pretty much
# the same interpretation.

plot(tempPCAsmooth)
```

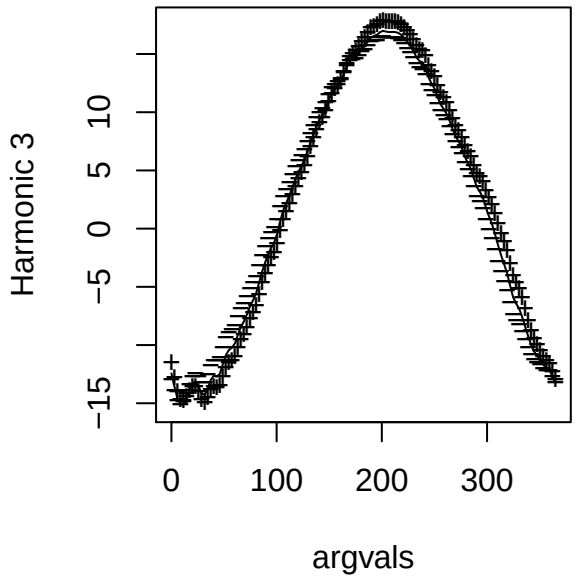
PCA function 1 (Percentage of variability 88.9)



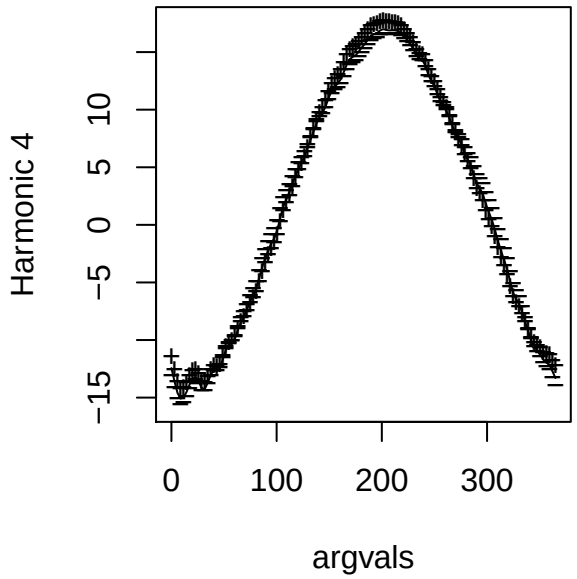
PCA function 2 (Percentage of variability 8.5)



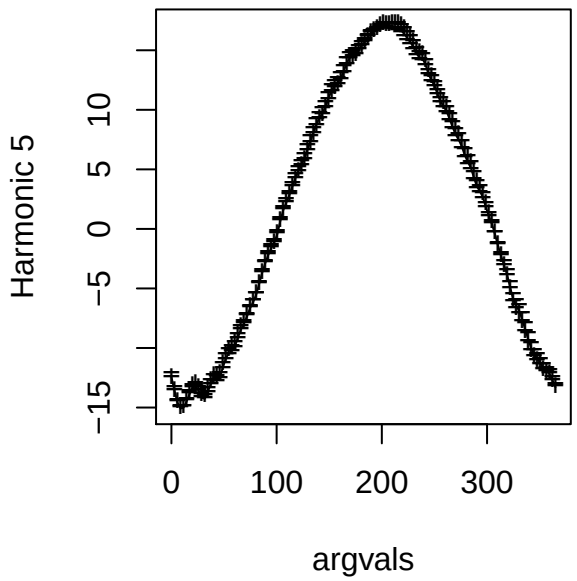
PCA function 3 (Percentage of variability 1.9)



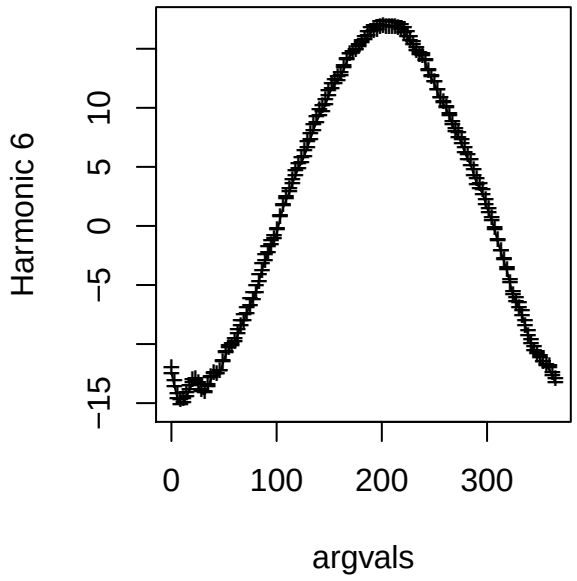
PCA function 4 (Percentage of variability 0.5)



PCA function 5 (Percentage of variability 0.1)



PCA function 6 (Percentage of variability 0.1)

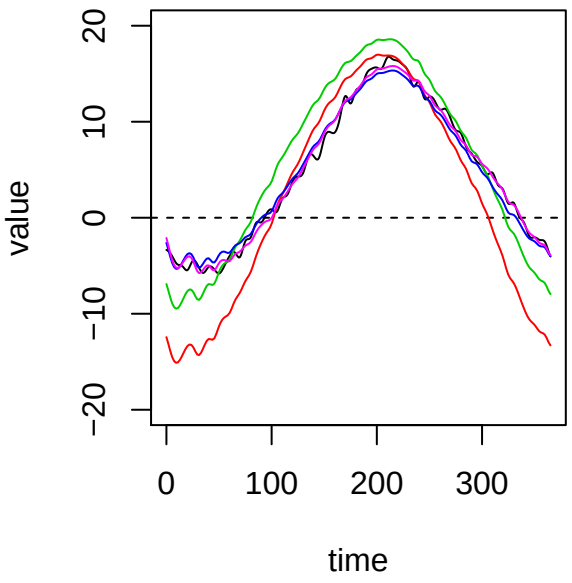



```
##### 3. PCA and Reconstructing Data

# We can ask how well the PCs reconstruct the observed functions. We'll focus
# on the first observation and reconstructing using PC score.
```

```r
# Let's extract the scores and PCs just to make the code easier to read

scores = tempPCAsmooth$scores
PCs = tempPCAsmooth$harmonics

# Firstly, just the mean + PC1

ex.temp.r1 = mtempfd + scores[1,1]*PCs[1]

# and plot these

plot(tempfd[1],ylim=c(-20,20))
```

```
## [1] "done"
```

```r
lines(mtempfd,col=2)
lines(ex.temp.r1,col=3)

# Try adding the second PC

ex.temp.r2 = mtempfd + scores[1,1]*PCs[1] + scores[1,2]*PCs[2]
lines(ex.temp.r2,col=4)

# And the third

ex.temp.r3 = ex.temp.r2  + scores[1,3]*PCs[3]
lines(ex.temp.r3,col=6)
```

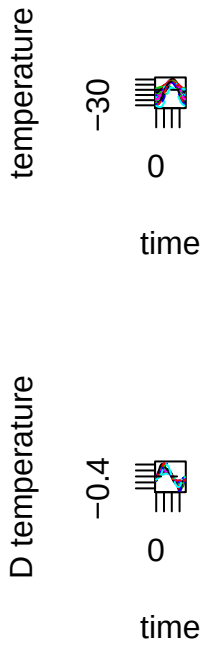



```r
### THOUGHT EXERCISE: how would you use this to choose the level of smoothing
### in pca.fd by leave-one-curve-out cross validation?
```

```
##### 3. PCA of Multivariate Functions

# To look at two dimensions, we'll re-smooth the temperature data and look at
# it along with its derivative. To do that, let's consider a fourier basis
# and harmonic acceleration.

# First an fdPar object; we'll use 65 Fourier basis functions; more than enough
# but this will cut down the computational time. We'll also over-smooth so our
# derivatives don't look so wild.

fbasis =  create.fourier.basis(dayrng,65)

harm.fdPar = fdPar(fbasis,harmLfd,1e6)

# Do the smooth
tempSmooth2 = smooth.basis(daytime,temp,harm.fdPar)

# and take a derivative

temp.deriv = deriv.fd(tempSmooth2$fd)

# Now we need to duck under the hood to create a joine fd object for both
# temperature and precipitation at once.

# This basically means I need an fd object that stacks the coefficients for
# temperature and the coefficients for D temperature along a third dimension
# of a coefficient array.

Dtempcoefs = array(0,c(65,35,2))

Dtempcoefs[,,1] = tempSmooth2$fd$coefs
Dtempcoefs[,,2] = temp.deriv$coefs

# Let's also deal with the dimension names

Dtemp.fdnames = tempfd$fdnames
Dtemp.fdnames[[3]] = c('temperature','D temperature')

# Put it all together

Dtempfd = fd(Dtempcoefs,fbasis,Dtemp.fdnames)

# Now we can plot

par(mfrow=c(2,1))
plot(Dtempfd[,1])
```

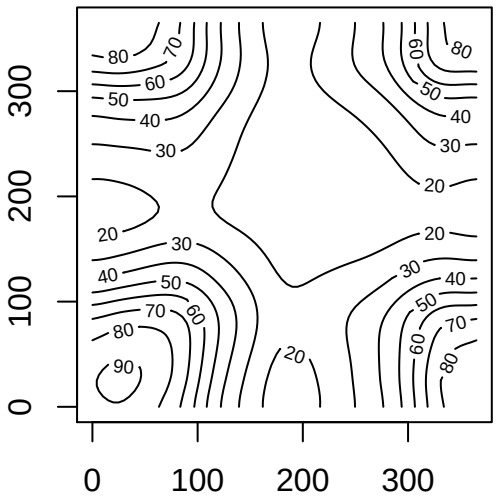
```
## [1] "done"
```

```
plot(Dtempfd[,2])
```

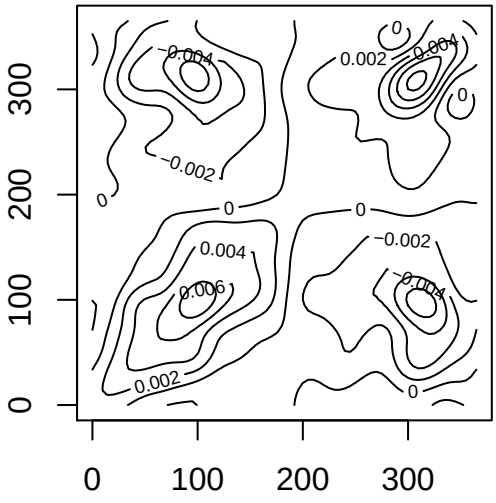



```
## [1] "done"
```

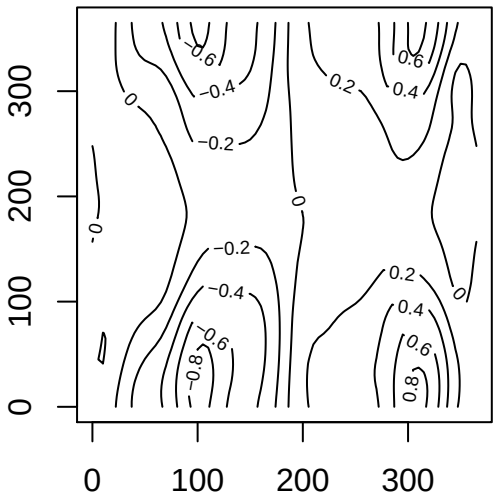
```
# We can also look at covariance

Dtemp.varbifd = var.fd(Dtempfd)

# If we look at

Dtemp.var = eval.bifd(day5,day5,Dtemp.varbifd)

# We have a 74 x 74 x 1 x 3 array.

par(mfrow=c(1,1))

# First temperature
contour(day5,day5,Dtemp.var[,,1,1])
```




```
# Then d temperature
contour(day5,day5,Dtemp.var[,,1,3])
```



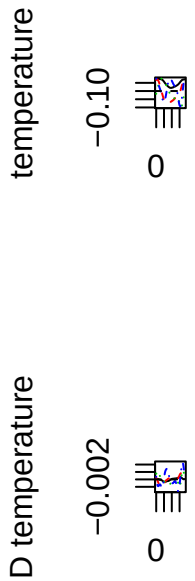

```
# Then their cross-product
contour(day5,day5,Dtemp.var[,,1,2])
```




```
### EXERCISE: experiment with cor.fd on this multidimensional fd object.

## Now let's look at the PCA

Dtemp.pca = pca.fd(Dtempfd,nharm=4)

# The PCs are now two dimensional

par(mfrow=c(2,1))
plot(Dtemp.pca$harmonics[,1])
```

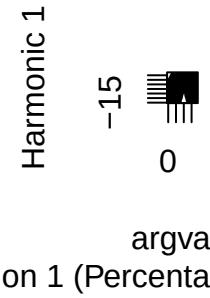
```
## [1] "done"
```

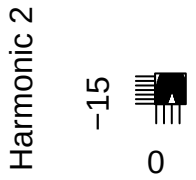
```
plot(Dtemp.pca$harmonics[,2])
```



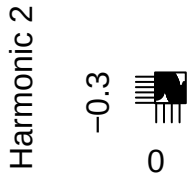

```
## [1] "done"
```

```
# We can plot these in the usual manner

plot(Dtemp.pca,harm=1:3)
```

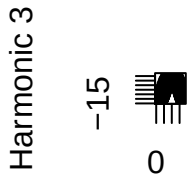




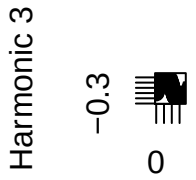
PCA function 2 (Percentage of variability 8.5)



PCA function 2 (Percentage of variability 8.5)



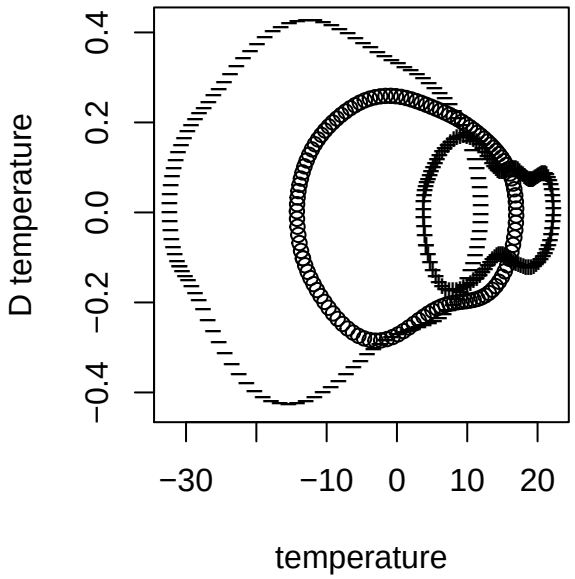
PCA function 3 (Percentage of variability 1.9)



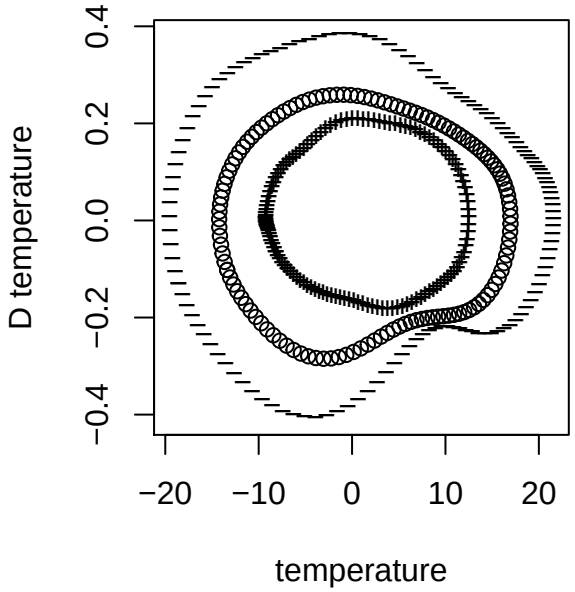
PCA function 3 (Percentage of variability 1.9)

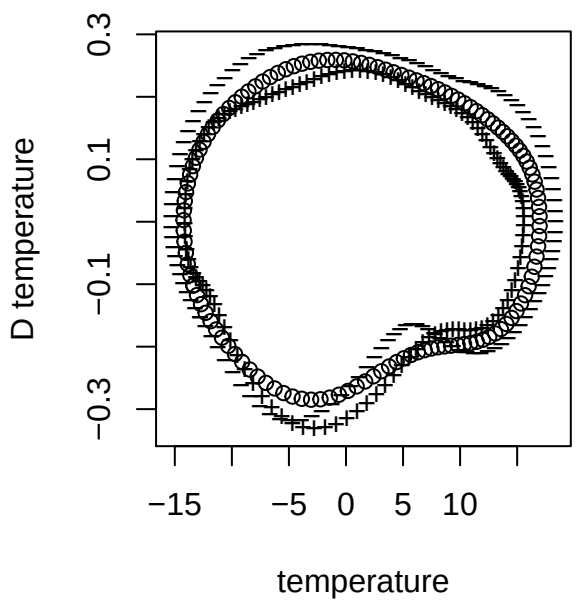
```
# But we can also plot the whole cycle

par(mfrow=c(1,1))
plot(Dtemp.pca,harm=1:3,cycle=TRUE,xlab='temperature',ylab='D temperature')
```

PCA function 1 (Percentage of variability 88.9)



PCA function 2 (Percentage of variability 8.5)






```
# PC1 = over-all temperature and little variation in derivatives.
# PC2 = high summer-winter variation and also large variation in derivatives
```