Source Code:Modeling Carbon Dioxide Emission Using Functional Linear Regression

```
rm(list=ls())
knitr::opts_chunk$set(echo = TRUE)
library(ggplot2)
library(splines)
library(Matrix)
library(fda)
library(nls2)
library(proto)
library(reshape2)
library(TeachingSampling)
library(tinytex)
```

Load the Data

```
data<-read.csv("dat.csv", header=FALSE, skip = 1)
data<-t(matrix(unlist(data), 2, 64, byrow=T))
year<- data[,1]
emission<- data[, 2]
n<- length(year)
time<- year-1950
```

Emission Plot, Figure1

```
# Figure1
emission_plot<-ggplot(data=data.frame(data), aes(x=data[,1]))+
  geom_point(aes(y=data[,2]))+
  scale_x_continuous(breaks = seq(1950, 2013, by=10))+
  scale_y_continuous(breaks=seq(800000,2000000,200000))+
   theme_bw()+
  labs( x="Year",
        y=expression(paste("CO"[2],phantom(x), "Emission (in 1000 Metric Tons)")))+
  theme(axis.text.y = element_text(angle=90 ))
```

Estimation of Betas using Functional Regression

First linear operator is created after observing linear and cyclic nature of the data. The coefficients from the first run of functional regression will be used to find stable and converging values of the coefficients.

```
p <- 2
refrange<- c(min(time),max(time))
norder  <- 6  ## cubic spline has order 4
nubasis <- 14  ### 10 interior knots
ubasis  <- create.bspline.basis(refrange, nubasis, norder)
ufd1 <- smooth.basis(time, emission, ubasis)$fd
```

```
#Smooth the data by using the minimum value of Lambda

#smoothing the data penalized by differential Operator with initial estimates of

#beta's that satisfies the condition as described in mathematical model

Oper= c(0, 0, (pi/21)^2, 0.05)

#convert this vector to the linear differential oeprator

Lfdobj=vec2Lfd(matrix(Oper, 1), refrange)

# Finding smoothing parameter and degree of freedom and values of generalized cross
#validation coefficient GCV associated with each value of lambda


loglam = seq(-5,5,0.25)
nlam = length(loglam)
dfsave = rep(NA,nlam)
gcvsave = rep(NA,nlam)

#loop through the smoothing values, storing degree of freedom and GCV along the way
for (j in 1:nlam){
  cat(paste('log10(lambda) =',loglam[j],'\n'))
   lambda= 10^loglam[j]
   fdparobj = fdPar(ubasis, Lfdobj, lambda)
  smoothlist= smooth.basis(0:63, emission, fdparobj)
  dfsave[j] = smoothlist$df
  gcvsave[j] = sum(smoothlist$gcv)
  }
#Smooth the data by using the minimum value of Lambda
lambda<- 10^2.75
fdparobj<-fdPar(ubasis, Lfdobj, lambda)
ufd<-smooth.basis(time, emission, fdparobj)$fd

#Derivatives of smooth function
par(mfrow=c(1,1))
der1<- deriv.fd(ufd,1)
der2<- deriv(ufd,2)
der3<- deriv(ufd,3)
der4<- deriv(ufd,4)

 #Estimate Betas using fRegress

bbasis    <- create.constant.basis(refrange)
betafd<-fd(matrix(c(0,0),1,p), bbasis)
betafdPar <- fdPar(betafd)
betalist <- list(betafdPar)
betalist[[1]] <- betafdPar
betalist[[2]] <- betafdPar
xfdlist1  <- list(c(der2, der3))
xfdlist1[[1]] <- der2
xfdlist1[[2]] <- der3
```

```
fRegressList<-fRegress(der4, xfdlist1 , betalist )

betaestlist <- fRegressList$betaestlist
Dyhatfdobj  <- fRegressList$yhatfdobj
```

Extract the Estimated Coefficients

```
betafdPar1 <- betaestlist[[1]]
betafdPar2 <- betaestlist[[2]]
beta1 <- betafdPar1$fd$coefs
beta2 <- betafdPar2$fd$coefs
```

Operator with the estimated Values of betas

```
#Operator with the values of betas
Oper1= c(0, 0, beta1, beta2)

#convert this vector to the linear differential oeprator

Lfdobj1=vec2Lfd(matrix(Oper1, 1), refrange)
```

Generalized Cross Validation with differential operator

GCV plot

```
# Figure 3
gcv_data1<- data.frame(loglam, gcvsave1)
 gcv<- ggplot(gcv_data1, aes(x=loglam))+
   geom_point(aes(y=gcvsave1), col="darkgrey")+
   geom_line(aes(y=gcvsave1))+
   theme_bw()+
   theme(axis.text.y = element_text(angle=90))+
   xlab(expression(log10(lambda)))+
 ylab(expression(GCV(lambda)))
 ggsave("gcv.pdf", width=7.5, height=4, dpi=300)
```

Functional regression with iteration

We started the operator with $\beta_1 = -0.0759 \quad \text{and} \quad \beta_2 = 0.038$. The converging values of betas are obtained on eight itiration. Final operator will be created by using these converging betas.

```
fbeta1<--0.0696 ## values from 10th itiration
 fbeta2<- 0.0667
Lcoef= c( 0, 0, fbeta1, fbeta2)
```

Functional regression after final operator smoothing

```
Lfdobj= vec2Lfd(matrix(Lcoef, 1), refrange)

lambda<- 10^2.25 # should be 10^2.25
melafdPar <- fdPar(ubasis , Lfdobj, lambda)
class(Lfdobj)
```

```
## [1] "Lfd"
```

```
#  smooth the data
melafd <- smooth.basis(0:63, emission, melafdPar)$fd
lder1<- deriv(melafd,1)
lder2<- deriv(melafd,2)
lder3<- deriv(melafd,3)
lder4<- deriv(melafd,4)
###
p <- 2
lbasis    <- create.constant.basis(refrange)
lbetafd<-fd(matrix(c(0,0),1,p), lbasis)
lbetafdPar <- fdPar(lbetafd)
lbetalist <- list(lbetafdPar)
lbetalist[[1]] <- lbetafdPar
lbetalist[[2]] <- lbetafdPar
lxfdlist1  <- list(c(lder2, lder3))
lxfdlist1[[1]] <- lder2
lxfdlist1[[2]] <- lder3

lfRegressList<-fRegress(lder4, lxfdlist1 , lbetalist )
lbetaestlist <- lfRegressList$betaestlist
lDyhatfdobj  <- lfRegressList$yhatfdobj
lbetafdPar1 <- lbetaestlist[[1]]
lbetafdPar2 <- lbetaestlist[[2]]
lbeta1 <- lbetafdPar1$fd$coefs
lbeta2 <- lbetafdPar2$fd$coefs
c(lbeta1,lbeta2) # these values are used in the manuscript.
```

```
## [1] -0.06964556  0.06686154
```

R^2

```
est<- as.data.frame(stats::predict(lfRegressList$yhatfdobj))
sys<-S.SY(501, 7.8)
est_der4<- est[sys,]
given<- data.frame(eval.fd(time,lder4))$rep1
diff_est<- given-est_der4
SSE<-sum(diff_est^2)
SST<- sum((given-mean(given))^2)
r_square<- (SST-SSE)/SST

# the value of R^2 may vary slightly for different itirarion
# find RMS of residuals along with the fit plot.

#plotfit.fd(emission, 0:63, melafd, ylab= "Emission_Thousands of Metric Tons", xlab= "Year",
#main = "Estimation of Rate of Change of Carbondioxide Emission in the USA")
```

##Smooth Graphs

```
sm_fit<- eval.fd(time, ufd) ## fitted smooth values
mm<- smooth.basis(0:63, emission, melafdPar); names(mm)
```

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

```
opr_fit<- eval.fd(time,melafd)
me1<- 1.96* sqrt(mm$SSE/(64-mm$df))
lower1<- opr_fit[,1]-me1
upper1<- opr_fit[,1]+me1

## Create Data Frame
opr_data<- data.frame(year, data[,2], sm_fit[,1],opr_fit[,1], lower1, upper1)
names(opr_data)<- c("year", "emission", "smooth_choice","smooth_operator", "lower", "upper")

#Figure 4
opr_plot<- ggplot(opr_data, aes(year)) +
  geom_point(aes(y=emission), col="darkgrey")+
geom_line(aes(y = smooth_operator), lty=1)+
  geom_line(aes(y = lower), lty=3)+
  geom_line(aes(y = upper), lty=3)+
  scale_x_continuous(breaks = seq(1950, 2013, by=10))+
  scale_y_continuous(breaks=seq(800000,2000000,200000))+
  theme_bw()+
  ylab(expression(paste('CO'[2], phantom(x),'Emission (in 1000 Metric Tons)' )))+
  xlab("Year")+
  theme(axis.text.y = element_text(angle=90))

#ggsave("opr_plot.pdf", width=7.5, height=4, dpi=300)
```

Fourth Derivative and its estimates curves

```
# Figure 5
par(mfrow=c(1,1))
png("der4_base.png", width=6, height=6, units="in", res=300)
x <- seq(1950,2013,1)
plot(lDyhatfdobj,ylim=c(-600, 800),xaxt='n',lty=2,xlab="Year",
     ylab="Fourth Derivatives",main="")
```

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

```
axis(side = 1, at = 1:64, labels = x, tck = -0.01)
lines(lder4, lty=1)
legend(0.2,810, c("4th derivative", "est 4th derivative"),bty = "n", cex=1,lty=c(1,2))
#dev.off()
```

Differential Equation

Use use nonlinear least square method to estimate the values of intigration contants c1, c2, ..cn in the differential equation.

```
beta1<-0.0696

beta2<- -0.0667
t<-time
m1<-nls(emission~c1*exp((0.0667/2)*t)*cos((sqrt(4*0.0696 -0.0667^2)/2)*t)+c2*exp((0.0667/2)*t)*
          sin((sqrt(4*0.0696 -0.0667^2)/2)*t)+c3*t+c4,
        start=list(c1=10000,c2=-3000,c3=-100,c4=26000))
sum<-summary(m1)

sum$coefficients[,1]
```

```
##        c1        c2        c3        c4
##  21516.07   5487.86  15161.69 729200.57
```

```
##

m2<-nls(emission~c1*exp((0.0667/2)*t)*cos((sqrt(4*0.0696 -0.0667^2)/2)*t)+c2*exp((0.0667/2)*t)*
          sin((sqrt(4*0.0696 -0.0667^2)/2)*t)+c3*t^2+c4*t+c5,
        start=list(c1=10000,c2=-3000,c3=-100,c4=2000,c5=26000))
sum2<-summary(m2)
sum2$coefficients[,1]
```

```
##          c1          c2          c3          c4          c5
##  14092.2684   8953.0733   -187.6738  26623.3423 614196.0903
```

Prediction and prediction interval

code credit :https://www.r-bloggers.com/

predictnls-part-1-monte-carlo-simulation-confidence-intervals-for-nls-models/

```
###Prediction Function for NLS model
predictNLS <- function(
  object,
  newdata,
  level = 0.95,
  nsim = 10000,
  ...
)
{
  require(MASS, quietly = TRUE)

  ## get right-hand side of formula
  RHS <- as.list(object$call$formula)[[3]]
  EXPR <- as.expression(RHS)

  ## all variables in model
  VARS <- all.vars(EXPR)

  ## coefficients
```

```r
COEF <- coef(object)

## extract predictor variable
predNAME <- setdiff(VARS, names(COEF))

## take fitted values, if 'newdata' is missing
if (missing(newdata)) {
  newdata <- eval(object$data)[predNAME]
  colnames(newdata) <- predNAME
}

## check that 'newdata' has same name as predVAR
if (names(newdata)[1] != predNAME) stop("newdata should have name '", predNAME, "'!")

## get parameter coefficients
COEF <- coef(object)

## get variance-covariance matrix
VCOV <- vcov(object)

## augment variance-covariance matrix for 'mvrnorm'
## by adding a column/row for 'error in x'
NCOL <- ncol(VCOV)
ADD1 <- c(rep(0, NCOL))
ADD1 <- matrix(ADD1, ncol = 1)
colnames(ADD1) <- predNAME
VCOV <- cbind(VCOV, ADD1)
ADD2 <- c(rep(0, NCOL + 1))
ADD2 <- matrix(ADD2, nrow = 1)
rownames(ADD2) <- predNAME
VCOV <- rbind(VCOV, ADD2)

## iterate over all entries in 'newdata' as in usual 'predict.' functions
NR <- nrow(newdata)
respVEC <- numeric(NR)
seVEC <- numeric(NR)
varPLACE <- ncol(VCOV)

## define counter function
counter <- function (i)
{
  if (i%%10 == 0)
    cat(i)
  else cat(".")
  if (i%%50 == 0)
    cat("\n")
  flush.console()
}

outMAT <- NULL

for (i in 1:NR) {
  counter(i)
```

```
    ## get predictor values and optional errors
    predVAL <- newdata[i, 1]
    if (ncol(newdata) == 2) predERROR <- newdata[i, 2] else predERROR <- 0
    names(predVAL) <- predNAME
    names(predERROR) <- predNAME

    ## create mean vector for 'mvrnorm'
    MU <- c(COEF, predVAL)

    ## create variance-covariance matrix for 'mvrnorm'
    ## by putting error^2 in lower-right position of VCOV
    newVCOV <- VCOV
    newVCOV[varPLACE, varPLACE] <- predERROR^2

    ## create MC simulation matrix
    simMAT <- mvrnorm(n = nsim, mu = MU, Sigma = newVCOV, empirical = TRUE)

    ## evaluate expression on rows of simMAT
    EVAL <- try(eval(EXPR, envir = as.data.frame(simMAT)), silent = TRUE)
    if (inherits(EVAL, "try-error")) stop("There was an error evaluating the simulations!")

    ## collect statistics
    PRED <- data.frame(predVAL)
    colnames(PRED) <- predNAME
    FITTED <- predict(object, newdata = data.frame(PRED))
    MEAN.sim <- mean(EVAL, na.rm = TRUE)
    SD.sim <- sd(EVAL, na.rm = TRUE)
    MEDIAN.sim <- median(EVAL, na.rm = TRUE)
    MAD.sim <- mad(EVAL, na.rm = TRUE)
    QUANT <- quantile(EVAL, c((1 - level)/2, level + (1 - level)/2))
    RES <- c(FITTED, MEAN.sim, SD.sim, MEDIAN.sim, MAD.sim, QUANT[1], QUANT[2])
    outMAT <- rbind(outMAT, RES)
  }

  colnames(outMAT) <- c("fit", "mean", "sd", "median", "mad", names(QUANT[1]), names(QUANT[2]))
  rownames(outMAT) <- NULL

  cat("\n")

  return(outMAT)
}


 pred_mat<-predictNLS(m2, newdata= data.frame(t=0:73))
```

```
## .........10.........20.........30.........40.........50
## .........60.........70....
```

Solution of Differential equation and scatter plot of data

```
#equation and scatter plot of data. Figure 6
  beta1<- 0.0696  # Sign change
```

```
 beta2<- -0.0667


 f=function(t){

    14092.2684*exp(-beta2/2*t)*cos(sqrt(4*beta1-beta2^2)/2*t)+8953.0733*exp(-beta2/2*t)*
     sin(sqrt(4*beta1-beta2^2)/2*t)-187.6738 *t^2+26623.3423 *t+614196.0903
 }

  t<-c(0:73)
 emission<- data[, 2]
 dif<-f(t)

 Year<- 1950: 2023
 dif_data<-data.frame(Year, c(emission, rep(NA, 10)),f(t),
                       pred_mat)
 names(dif_data)<- c("Year", "emission", "prediction", "fit",
                     "mean", "sd", "median", "mad", "lower", "upper")
 #mad: mean absolute deviation
 dif_plot_new<-ggplot(dif_data, aes(Year)) +
   geom_point(aes(y=emission),color="darkgrey")+
   geom_line(aes(y=prediction))+
   geom_line(aes(y=lower), lty=2)+
   geom_line(aes(y=upper), lty=2)+
   scale_x_continuous(breaks = seq(1950, 2023, by=10))+
   theme_bw()+
   ylab(expression(paste('CO'[2], phantom(x), 'Emission (in 1000 Metric Tons)')))+
   xlab("Year")+
   scale_y_continuous(breaks=seq(800000,2000000,200000))+
   theme(axis.text.y = element_text(angle=90))
ggsave(dif_plot_new, file="dif_plot_new.pdf",width=7.5, height=4, dpi=300)
```

```
## Warning: Removed 10 rows containing missing values (geom_point).
```

##Simulation

We get a massage “No id variables; using all as measure variables”. This is because of melt function.

```
rm(list=ls())
M=rep(0,20)
H=matrix(M,nrow=20,ncol=64,byrow=TRUE)
G=matrix(M, nrow=20,ncol=64,byrow = TRUE )

#Simulate 50 datasets of similar types(increasing and cyclic)
for(k in 1:20)
{
  beta1=(pi/21)^2
  beta2<-runif(20, min=-2*beta1,max=2*beta1)
  t=seq(1,64,1)
  c1=35000
  c2=-105000
  c3=-100
  c4=20000
  c5=600000
  for (i in 1:64)
  {
    G[k,i]=c1*exp(-beta2[k]/2*i)*cos(sqrt((4*beta1-beta2[k]^2)/2)*i)+
      c2*exp(-beta2[k]/2*i)*sin(sqrt((4*beta1-beta2[k]^2)/2)*i)+c3*i^2+c4*i+c5
  }
}
## Multiple graphs of all 20 simulated data # graph in the appendix

multi_graph<- data.frame(t(G))
v<-melt(multi_graph)
```

```
## No id variables; using all as measure variables
```

```
plot.20<- data.frame(time=rep(1950:2013,20), v)
graphs.20<-ggplot(plot.20, aes(x=time))+
  geom_line(aes(y= value, color=variable),show.legend=F)+
  #scale_x_continuous(breaks = seq(1950, 2013, by=1))+
  #geom_point(aes(y=value), alpha=0.01)+
 facet_wrap(~variable)+

  ylab(expression(paste('Simulated CO'[2], phantom(x), 'Emission (in 1000 Metric Tons)')))+
  xlab("Year")
#ggsave(graphs.20, file="simulated_data_20_plots.pdf", dpi=300)

### Graph of One simulated data # Figure 2
one_sim_plot<- ggplot(data.frame(time=seq(1,64,1),multi_graph), aes(x=time))+
  geom_point(aes(y=X1))+

  scale_y_continuous(breaks=seq(800000,2000000,200000))+
  theme_bw()+
  labs( x="Year",
        y=expression(paste("CO"[2],phantom(x), "Emission (in 1000 Metric Tons)")))+
  theme(axis.text.y = element_text(angle=90 ))
```