FQLearn Example

Robert Pehlman

May 10, 2017

This is a short example demonstrating some code to read in and use FQLearn. The installation of the package FQ Learn can be finicky. It requires Rcpp_0.12.10, which can be downloaded from CRAN if you don't have it. Package should be installed before running any of this and package installation is dependent on packages Rcpp and RcppArmadillo. Example code to run before vignette.

```
library(Rcpp)
library(RcppArmadillo)
install.packages("FQLearn_1.0.tar.gz", repos = NULL, type = "Source")
```

Code generates some data from a gaussian process and provides noisy observations as output in the object train$w. All of the lines up to GenerateDataFQ() create parameters needed to generate data. GenerateDataFQ() is a function that generates data in the form used by the package. The likelihood at the true values is calculated, and then estimateProcess() (from FQLearn) is called. estimateProcess() is the workhorse of the FQLearn package. It can accept a method argument, (which is set to method = "PSD" by default), and can make use of a second order algorithm using method = "RTRSR1" if the user has package manifoldOptim installed (must be downloaded and installed from source currently not on CRAN).

```
library(fda)
library(far)
library(mvtnorm)
##
## Requires Rcpp_0.12.10
##
library(Rcpp)
library(RcppArmadillo)
library(MASS)
library(FQLearn)
#library(manifoldOptim)

setting = "S2"  #S1, S2, S3
sparsity = "spar"  #spar, mod, den

L = 10;
p = 5
noTimePoints = 51

n = 100

if(setting == "S1"){
  coefficientfunction = rep(1,51)
} else if(setting == "S2"){
  coefficientfunction = c(rep(0,45), rep(1,6))
} else if(setting == "S3"){
  coefficientfunction = -25:25/50
}
```

```
if(sparsity == "spar"){
  c_=5
  d_=15
} else if(sparsity == "mod"){
  c_=15
  d_=10
} else if(sparsity == "den"){
  c_=25
  d_=5
}

nMax2 = noTimePoints;
maxGrid2 = seq (0, 1, length=nMax2);
op2 = create.bspline.basis (nbasis=L);
bMat2 = eval.basis (maxGrid2, op2);
OBASIS = orthonormalization (bMat2, basis=F);
OBASIS = OBASIS*sqrt(nMax2);

tau = 1
nu = 1      ##### Standard deviations of Deltas!  NOT VARIANCES!!!!  Currently constant

Gamma0 = array(rep(0,(L*p)),c(p, L))

for(i in 1:p){
  margin = round((L-p) / 2)
  Gamma0[i, (margin + i)] = 1
}
Gamma0
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    0    1    0    0    0    0    0    0     0
## [2,]    0    0    0    1    0    0    0    0    0     0
## [3,]    0    0    0    0    1    0    0    0    0     0
## [4,]    0    0    0    0    0    1    0    0    0     0
## [5,]    0    0    0    0    0    0    1    0    0     0
```

```
lambda0 = sqrt(2^seq(1:p)/(2^round(p/2)))
#lambda0 = 2^seq(1:p)/(2^round(p/2))
lambda0
```

```
## [1] 0.7071068 1.0000000 1.4142136 2.0000000 2.8284271
```

```
GenerateDataFQ <- function(n, Betastar, Betastara, c, d, L, p, trtFlag = FALSE) {

  randstarts = 1  # Number of random starts
  params = p

  # Empty array holds all m phi0 functions at 51 times and n observations
  phi0 = array(rep(NA, 51 * p * n), c(51, p, n))
  phi1 = array(rep(NA, 51 * p * n), c(51, p, n))

  a = rep(0, n)
  u = rep(0, n)
  rank = rep(0, n)
  Y = rep(0, n)
```

```
Ya0 = rep(0, n)
Ya1 = rep(0, n)
w = array(rep(0, 51 * n), c(51, n))
z_51 = array(rep(0, 51 * n), c(51, n))

# Empty arrays for random coefficients to p phi0 functions over n obs
Xi0 = array(rep(0, p * n), c(p, n))
Xi1 = array(rep(0, p * n), c(p, n))
Xi1a0 = array(rep(0, p * n), c(p, n))
Xi1a1 = array(rep(0, p * n), c(p, n))

errors = array(rep(0, 51 * n), c(51, n))

condexp = array(rep(0, 2 * p * n), c(2 * p, n))
condexp2 = array(rep(0, 2 * p * n), c(2 * p, n))

for (i in 1:n){

  times = sample(c : (c+d), 1)       # number of observed time points
  indices = sample(1 : 50, times)  # Exact value time points on grid

  indices = sort(indices)
  tgrid = (indices - 1)/50

  # These lines correspond to the phi0_0k functions evaluated at tgrid

  B_T=t(OBASIS[indices,])
  phi0temp = t(Gamma0 %*% B_T)

  phi0[indices, , i] = phi0temp

  Xi0[, i] = rnorm(p, 0, lambda0)

  # creates the error terms
  errors[indices, i] = rnorm(times, 0, tau)


  z = phi0[, , i] %*% Xi0[, i]

  # creates the final w(T) for one observation
  w[, i] = z + errors[, i]

  phi0_51  = t(Gamma0 %*% t(OBASIS))

  z_51[,i] = phi0_51 %*% Xi0[, i]


  Y[i] = t(z_51[,i]) %*% Betastar /51
}

## Omitted pending need -  "condexp" = condexp, "condexp2" = condexp2, may need to define B if use
listout <- list("phi0" = phi0,"w" = w, "indices" = indices, "tgrid" = tgrid, "Y" = Y,
                "Xi0" = Xi0, "z_51" = z_51)
```

```
  return(listout)
}
```

The output of estimateProcess(). The time funcion is not currently working for method = "PSD".

```
set.seed(8675302)
genp = dim(Gamma0)[1]
trueParameters = c(as.vector(Gamma0), rep(0,(p*4)), lambda0, tau, 1, 1)
trueParams2 = c(as.vector(t(Gamma0)), lambda0, tau)

train = GenerateDataFQ(n, coefficientfunction, coefficientfunction, c_, d_, L, genp)

print("loglike at true params: ")
```

```
## [1] "loglike at true params: "
```

```
trueParamsNoTxt =  c(as.vector(Gamma0), rep(0,(p*4)), lambda0, tau, 1, 1)
logLikelihood(trueParamsNoTxt, train$w, a = NULL, u = NULL, p, L, trtFlag = FALSE)
```

```
## [1] -2489.15
```

```
result = estimateProcess(noTimePoints = 51, data = train$w, nBasisFns = 10,
                         nEigFns = 5, meanZeroFlag = TRUE, trtFlag = FALSE,
                         seedValue = NULL, noIter = 300, tol = 0.01)

# Likelihood at estimates
result$logLikelihood
```

```
## [1] 2467.487
```

```
# Parameter estimates
result$estGamma
```

```
##              [,1]        [,2]        [,3]         [,4]        [,5]
## [1,]  0.009769883 -0.10149411 -0.96272025 -0.107224945  0.03654590
## [2,] -0.014545867 -0.02282421  0.11960426 -0.989657228 -0.01813311
## [3,] -0.002331032  0.02724101  0.04811170  0.012771207  0.91191665
## [4,]  0.005964974  0.01327082  0.01463501 -0.063616114  0.33761347
## [5,] -0.010331716  0.02068427 -0.01051679 -0.005660537  0.21655897
##             [,6]        [,7]          [,8]         [,9]       [,10]
## [1,]  0.02243208 -0.01360808 -1.428918e-01  0.035043111 -0.16615037
## [2,]  0.06127083  0.02037318 -8.006639e-03  0.002859555 -0.03115431
## [3,]  0.39652648 -0.07846330  1.673681e-05  0.009738482 -0.04120807
## [4,] -0.85784405 -0.37550922  8.057189e-04 -0.015646934 -0.06629625
## [5,] -0.31685140  0.92284473 -6.543246e-03 -0.009748540 -0.01580902
```

```
result$estLambda
```

```
## [1] 0.5545702 1.0501737 1.6118336 2.5985128 3.0815688
```

```
result$estTau
```

```
## [1] 0.9247415
```

```
# Elapsed time
result$time
```

```
## function (x, ...)
```

```
## UseMethod("time")
## <bytecode: 0x0000000013f73860>
## <environment: namespace:stats>
```

```
### Not Run unless you have manifoldOptim installed

result = estimateProcess(noTimePoints = 51, data = train$w, nBasisFns = 10, nEigFns = 5,
                         meanZeroFlag = TRUE, trtFlag = FALSE, method = "RTRSR1", seedValue = NULL,
                         noIter = 300, tol = 1)
```

```
## [1]  51 100
## [1] 2489.15
## GENERAL PARAMETERS:
## name          :Product Manifold, IsIntrApproach:             1
## IntrinsicDim  :             41,  ExtrinsicDim  :            56
## HasHHR        :              0,  UpdBetaAlone  :             0
## HasLockCon    :              0
## 0-th manifold parameters (the number is 1) :
## GENERAL PARAMETERS:
## name          :        Stiefel,  IsIntrApproach:             1
## IntrinsicDim  :             35,  ExtrinsicDim  :            50
## HasHHR        :              0,  UpdBetaAlone  :             0
## HasLockCon    :              0
## Stiefel PARAMETERS:
## n             :             10,  p             :             5
## metric        :      EUCLIDEAN,  retraction    :            QF
## VecTran       :PARALLELIZATION
## 1-th manifold parameters (the number is 1) :
## GENERAL PARAMETERS:
## name          :      Euclidean,  IsIntrApproach:             0
## IntrinsicDim  :              6,  ExtrinsicDim  :             6
## HasHHR        :              0,  UpdBetaAlone  :             0
## HasLockCon    :            114
## Euclidean PARAMETERS:
## row           :              6
## GENERAL PARAMETERS:
## Stop_Criterion:         GRAD_F[YES], Tolerance     :              1[YES]
## Max_Iteration :           1000[YES], Min_Iteration :              0[YES]
## OutputGap     :              1[YES], DEBUG         :       NOOUTPUT[YES]
## TRUST REGION TYPE METHODS PARAMETERS:
## initial_Delta :              1[YES], Acceptence_Rho:            0.1[YES]
## Shrinked_tau  :           0.25[YES], Magnified tau :              2[YES]
## minimum_Delta :   2.22045e-016[YES], maximum_Delta :           1000[YES]
## Min_Inner_Iter:              0[YES], Max_Inner_Iter:           1000[YES]
## theta         :            0.1[YES], kappa         :            0.1[YES]
## useRand       :              0[YES]
## RTRSR1 METHOD PARAMETERS:
## isconvex      :              0[YES]
##    user  system elapsed
##    7.36    0.04    7.50
## [1] 2489.15
## [1] 2459.545
## [1] "Prob Converged?"
## [1] TRUE
## [1] "Time?"
```

```
## [1] 7.5
```

```
# Likelihood at estimates
result$logLikelihood
```

```
## [1] 2459.545
```

```
# Parameter estimates
result$estGamma
```

```
##              [,1]        [,2]         [,3]        [,4]        [,5]
## [1,]  0.009432657  0.10566400  0.976297838  0.10368769 -0.03954588
## [2,]  0.015103814  0.02981778 -0.103128740  0.98031244  0.15966307
## [3,] -0.005484460  0.02879635  0.055932047 -0.14670678  0.97335279
## [4,] -0.003743375 -0.01190545  0.004198615  0.05824458 -0.14513355
## [5,]  0.011528416 -0.01150434  0.017851534 -0.01630416  0.03509991
##              [,6]        [,7]          [,8]         [,9]        [,10]
## [1,] -0.01693198  0.01325385  0.1473803019  0.008305411  0.03195892
## [2,] -0.03413824 -0.01349899  0.0077183403 -0.004950001  0.01762813
## [3,]  0.15337450  0.04160768 -0.0037564923  0.005982315 -0.04205851
## [4,]  0.98595914  0.01697111  0.0006572921  0.019322495  0.05092346
## [5,]  0.02392805 -0.99850026  0.0061048944  0.004437490 -0.01691049
```

```
result$estLambda
```

```
## [1] 0.5540211 1.0345836 1.4652248 2.5298381 3.2348194
```

```
result$estTau
```

```
## [1] 0.9239959
```

```
# Elapsed time
result$time
```

```
## [1] 7.5
```