Supplementary Materials

1 Additional Figures

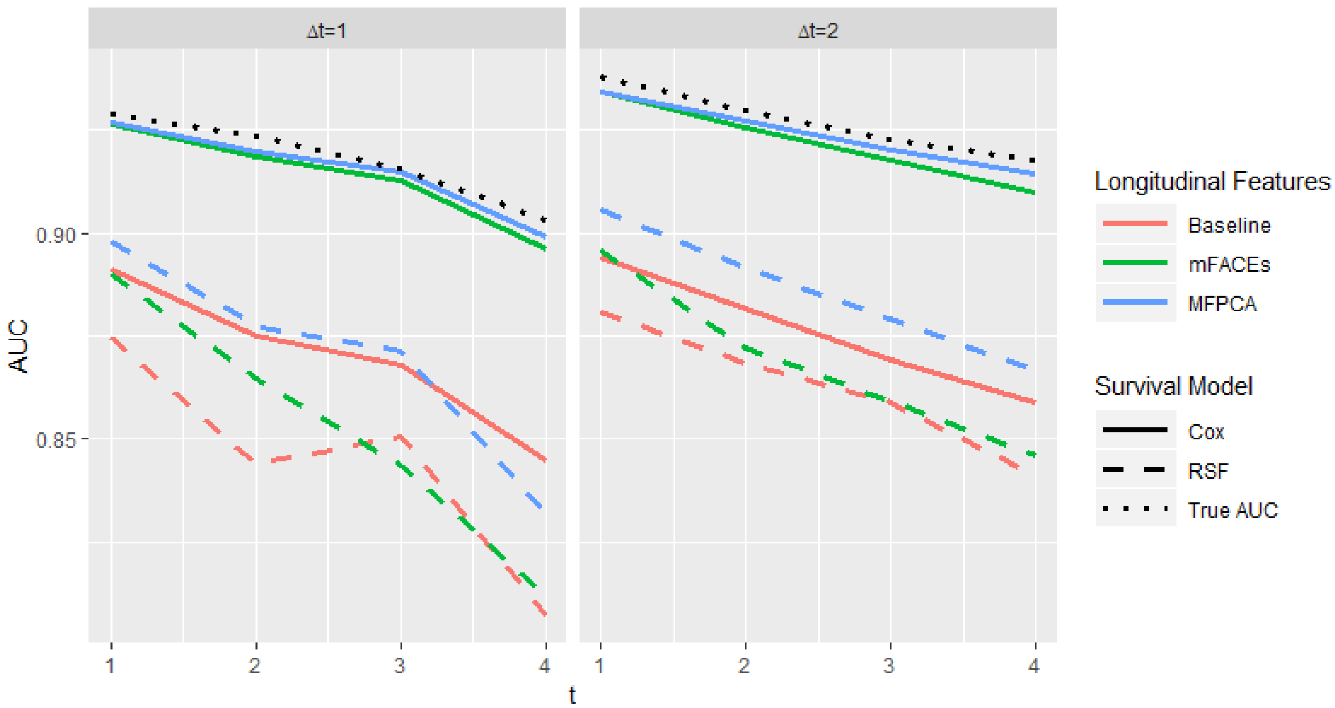

Figure S1: Comparison of AUC's from Scenario 1. Each model is composed of the method used to extract features of the longitudinal data (represented by color) and the survival model (represented by linetype). Baseline models exclude the longitudinal data.

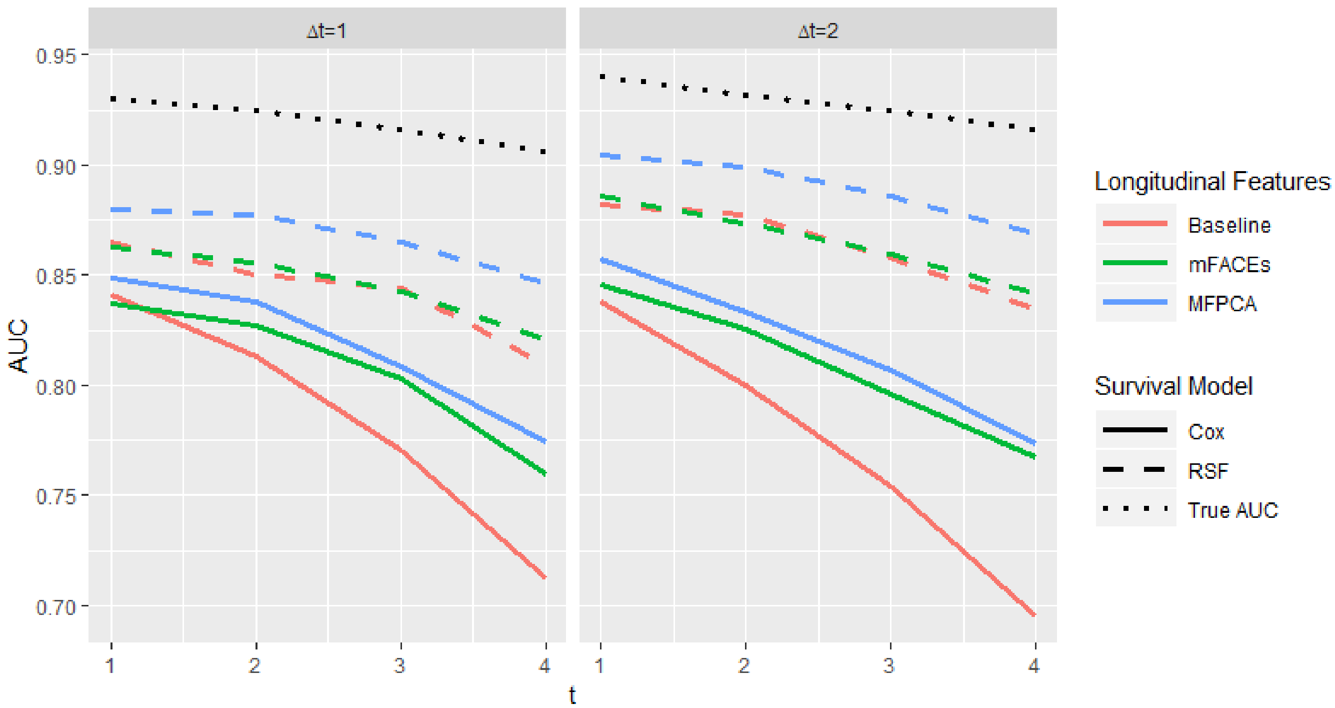

Figure S2: Comparison of AUC's from Scenario 2. Each model is composed of the method used to extract features of the longitudinal data (represented by color) and the survival model (represented by linetype). Baseline models exclude the longitudinal data.

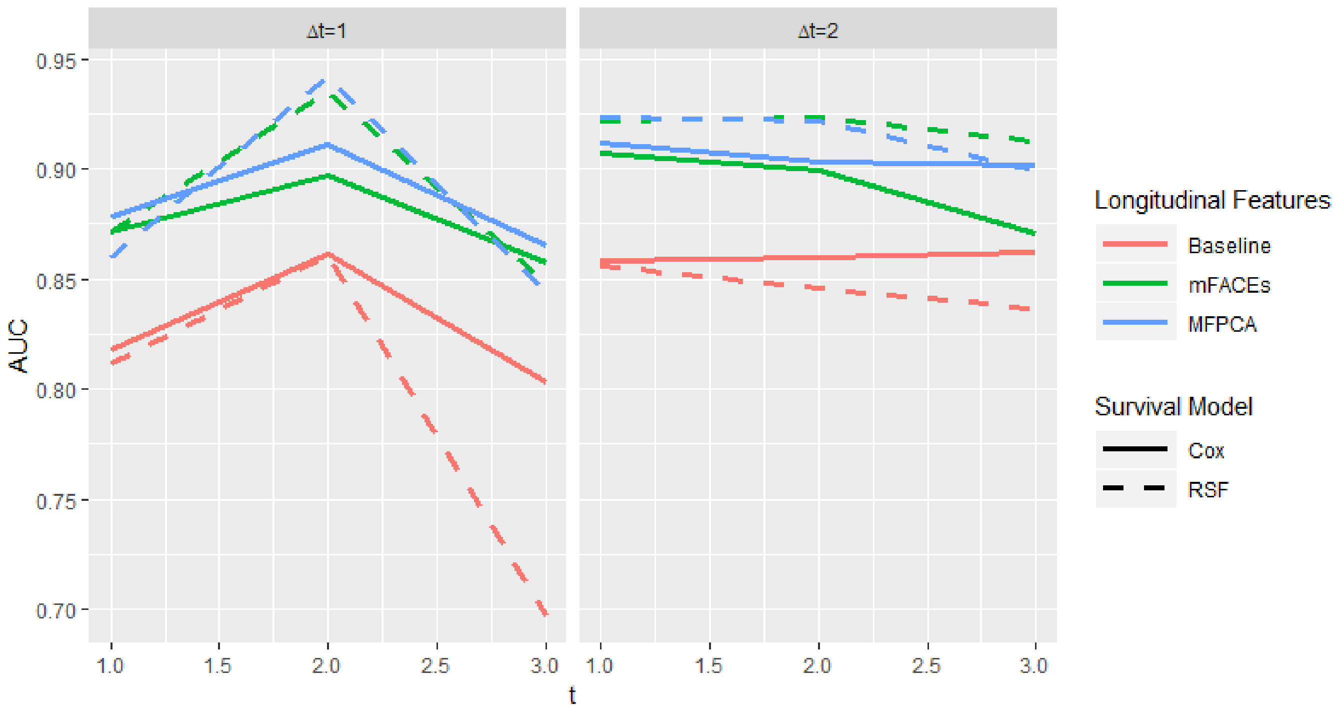

Figure S3: Comparison of AUC's from models of the ADNI study. Each model is composed of the method used to extract features of the longitudinal data (represented by color) and the survival model (represented by linetype). Baseline models exclude the longitudinal data.

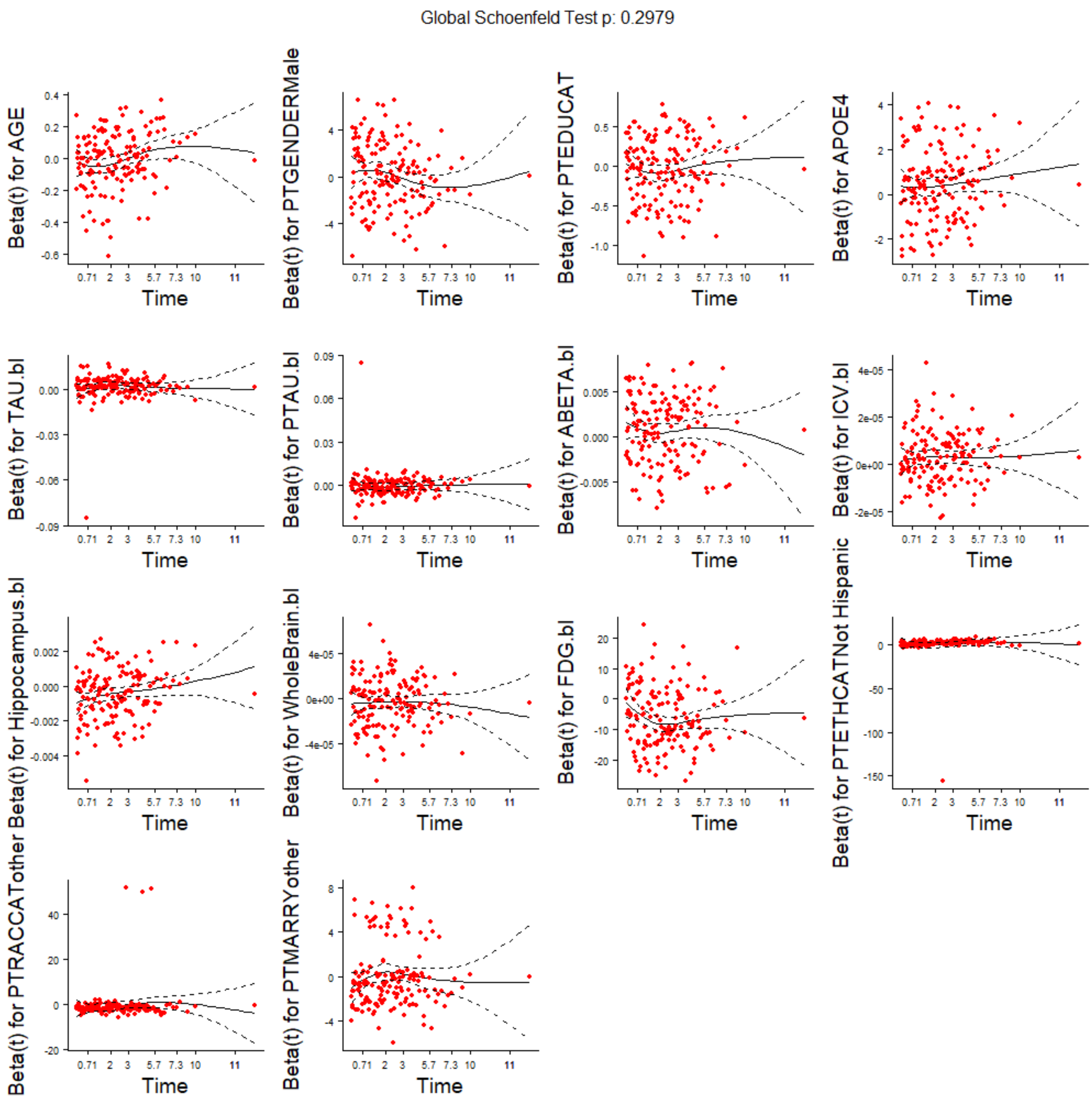

Figure S4: Scaled Schoenfeld residuals plotted against transformed time for each baseline covariate.

2 Simulation study with sparse data setting

We repeated the simulation for Scenario 1 with 30% of the longitudinal measurements randomly deleted. The AUC and Brier score are presented in Table S1 of the supplement. In these results, we do not observe a significant difference in the AUC and Brier scores between the MFPCA and mFACEs methods. Therefore, we cannot confirm that mFACEs performs better when the data is sparse in our simulation setting.

			RSF		Cox	
t	Δt	True AUC	AUC	BS	AUC	BS
MFPCA						
1	1	0.929	0.898	0.055	0.927	0.044
1	2	0.938	0.904	0.089	0.934	0.067
2	1	0.924	0.875	0.062	0.918	0.051
2	2	0.930	0.889	0.103	0.927	0.078
3	1	0.915	0.867	0.073	0.915	0.059
3	2	0.923	0.874	0.123	0.919	0.091
4	1	0.903	0.828	0.088	0.897	0.071
4	2	0.918	0.859	0.144	0.912	0.107
mFACEs						
1	1	0.929	0.894	0.055	0.927	0.044
1	2	0.938	0.900	0.091	0.934	0.067
2	1	0.924	0.867	0.063	0.919	0.051
2	2	0.930	0.882	0.106	0.926	0.079
3	1	0.915	0.858	0.074	0.913	0.060
3	2	0.923	0.870	0.126	0.917	0.093
4	1	0.903	0.824	0.089	0.896	0.073
4	2	0.918	0.853	0.149	0.910	0.109

Table S1: Sparse simulation setting: AUC and BS are averaged across 100 simulated datasets with 30% of visits removed.

3 Code

3.1 MFPCA code for simulation study

```
library(MASS)
library(MFPCA)
library(randomForestSRC)
library(pec)
library(survival)
source('functions.r')

n.sim = 100      #number of simulation runs
n = 300          #sample size
n.train = 200    #n.test = n - n.train

#dynamic prediction information
obstime = seq(0,10,0.5) # longitudinal measurement time
Tstart = c(1,2,3,4) # landmark time for prediction
deltaT = c(1,2) # prediction windowns
argvals = obstime

scenario = "none"   #options: ["none","interaction"]


for(i.run in 1:n.sim){
    print(i.run)
    set.seed(123+i.run)

    ### Simulation ###
    sim.data = sim_mjm_linear(n, obstime=obstime, opt=scenario)
    long = sim.data$long  # longitudinal data
    surv = sim.data$surv  # survival data


    # split data to training and testing
    train.id = c(1:n.train)
    test.id = c((n.train+1):n)

    train.surv = surv[surv$id%in%c(1:n.train), ]
    test.surv = surv[!surv$id%in%c(1:n.train), ]

    # subject ids
    patID = surv$id
    nPat = length(patID)

    # transfer longitudinal outcomes from long to wide
    multivar = array(NA, c(n, length(obstime), 3))
```

```
for(i in 1:nPat){
    visits = which(obstime %in% (long$obstime[long$id == patID[i]]))
    multivar[i,visits, 1] = long$Y1[long$id == patID[i]]
    multivar[i,visits, 2] = long$Y2[long$id == patID[i]]
    multivar[i,visits, 3] = long$Y3[long$id == patID[i]]
}
multivar.train = multivar[train.id, , ]


# univariate FPCA via PACE
Xi.train = L = phi.train = meanFun.train = NULL
x.tmp = NULL
for(p in 1:3){
    tmp.ufpca = uPACE(multivar.train[,,p], argvals, nbasis=7)
    x.tmp = cbind(x.tmp,tmp.ufpca$scores[,1])
    Xi.train = cbind(Xi.train, tmp.ufpca$scores) # FPC scores
    L = c(L, dim(tmp.ufpca$scores)[2])
    phi.train[[p]] = t(tmp.ufpca$functions@X) # FPC eigenfunctions
    meanFun.train[[p]] = tmp.ufpca$mu@X # estimated mean functions
}

# multivariate FPCA
mFPCA.train = mFPCA(Xi=Xi.train, phi=phi.train, p=3, L=L, I=n.train)
rho.train = mFPCA.train$rho  #MFPC scores
pve = mFPCA.train$pve
psi = mFPCA.train$psi
Cms = mFPCA.train$Cms

colnames(rho.train) = paste0("rho.",(1:ncol(rho.train)))
train.surv.temp = cbind(train.surv[,2:5],rho.train)

#RSF Model
rsf.fit = rfsrc(Surv(time,event)~., data=train.surv.temp,
                ntree=1000, seed=i.run)

#Cox Model
cox.fit = coxph(Surv(time, event)~., data = train.surv.temp, model=T, x=T, y=T)


# dynamic prediction
DP.id = DP.prob.rsf = DP.prob.cox = DP.prob.truecox = timeEvent= trueProb = NULL
ith = 0
for(t in Tstart){
    tmp.id = test.surv[test.surv$time>t, "id"]
    tmp.surv.data = test.surv[test.surv$time>t, ] # filter the data
    tmp.data = multivar[tmp.id, , ] # subset longitudinal outcomes
    tmp.data[,-c(1:which(t==obstime)),] = NA
```

```
        # univariate FPC
        Xi.test = NULL
        for(p in 1:3){
            tmp.ufpca = uPACE(multivar.train[,,p], argvals, tmp.data[,,p], nbasis = 7)
            Xi.test = cbind(Xi.test, tmp.ufpca$scores)
        }

        # estimate MFPC scores for test subjects
        rho.test = mfpca.score(Xi.test, Cms)
        colnames(rho.test) = paste0("rho.",(1:ncol(rho.test)))
        test.surv.temp = cbind(tmp.surv.data[,2:5],rho.test)


        # prediction for different time windowes
        for(dt in deltaT){
            ith = ith + 1
            DP.id[[ith]] = tmp.id
            timeEvent[[ith]] = tmp.surv.data[, c("time", "event")]
            trueProb [[ith]] = tmp.surv.data$true.prob[, (which((t+dt)==obstime)-1)]
            DP.prob.rsf[[ith]] = cond.prob.pec(rsf.fit, test.surv.temp, t, (t+dt))
            DP.prob.cox[[ith]] = cond.prob.pec(cox.fit, test.surv.temp, t, (t+dt))
        }
    }
    DP.prob = DP.prob.rsf
    save(sim.data, surv, train.id, trueProb, DP.id, DP.prob, timeEvent,
         file=paste(c("output/mfpca/",scenario,"/mfpca_rsf",i.run,".rdata"), collapse=""))

    DP.prob = DP.prob.cox
    save(sim.data, surv, train.id, trueProb, DP.id, DP.prob, timeEvent,
         file=paste(c("output/mfpca/",scenario,"/mfpca_cox",i.run,".rdata"), collapse=""))

}
```

3.2 mFACEs code for simulation study

```
library(MASS)
library(mfaces)
library(randomForestSRC)
library(pec)
library(survival)
source('functions.r')

n.sim = 100      #number of simulation runs
n = 300          #sample size
n.train = 200    #n.test = n - n.train

#dynamic prediction information
```

```r
obstime = seq(0,10,0.5) # longitudinal measurement time
Tstart = c(1,2,3,4) # landmark time for prediction
deltaT = c(1,2) # prediction windowns
argvals = seq(0,max(obstime),length=10*max(obstime)+1)

scenario = "none"   #options: ["none","interaction"]


for(i.run in 1:n.sim){
    print(i.run)
    set.seed(123+i.run)


    ### Simulation ###
    sim.data = sim_mjm_linear(n, obstime=obstime, opt=scenario)
    long = sim.data$long
    surv = sim.data$surv


    ###split data to training and testing
    train.id = c(1:n.train)
    test.id = c((n.train+1):n)

    train.surv = surv[surv$id%in%train.id, ]
    test.surv = surv[surv$id%in%test.id, ]
    train.long = long[long$id%in%train.id, ]
    test.long = long[long$id%in%test.id, ]


    y1 = data.frame("subj"=train.long$id,"argvals"=train.long$obstime,"y"=train.long$Y1)
    y2 = data.frame("subj"=train.long$id,"argvals"=train.long$obstime,"y"=train.long$Y2)
    y3 = data.frame("subj"=train.long$id,"argvals"=train.long$obstime,"y"=train.long$Y3)

    y1 = y1[which(!is.na(y1$y)),]
    y2 = y2[which(!is.na(y2$y)),]
    y3 = y3[which(!is.na(y3$y)),]

    multivar.train = list(y1,y2,y3)

    ### mFACEs
    mfaces.fit = mface.sparse(multivar.train, argvals.new = argvals, knots=7,
                              newdata = multivar.train, calculate.scores = TRUE)

    train.surv.tmp = cbind(train.surv[,2:5],mfaces.fit$scores$scores)


    ### Random Survival Forest
    rsf.fit = rfsrc(Surv(time,event)~., data=train.surv.tmp,
```

```
                ntree=1000, seed=i.run)


### Cox Model
cox.fit = coxph(Surv(time,event)~., data=train.surv.tmp, model=TRUE, x=TRUE, y=TRUE)



#dynamic prediction
DP.id = DP.prob = DP.prob.rsf = DP.prob.cox = timeEvent = trueProb = NULL
ith = 0
for(t in Tstart){
    tmp.surv.data = test.surv[test.surv$time>t, ]
    tmp.id = tmp.surv.data[["id"]]
    test.long = test.long[which(test.long$id %in% tmp.id),]
    test.long = test.long[which(test.long$obstime<=t),]

    y1 = data.frame("subj"=test.long$id,"argvals"=test.long$obstime,"y"=test.long$Y1)
    y2 = data.frame("subj"=test.long$id,"argvals"=test.long$obstime,"y"=test.long$Y2)
    y3 = data.frame("subj"=test.long$id,"argvals"=test.long$obstime,"y"=test.long$Y3)

    y1 = y1[which(!is.na(y1$y)),]
    y2 = y2[which(!is.na(y2$y)),]
    y3 = y3[which(!is.na(y3$y)),]

    multivar.test = list("y1"=y1,"y2"=y2,"y3"=y3)

    mfaces.pred = predict(mfaces.fit, multivar.test, scores=TRUE)

    test.surv.tmp = cbind(tmp.surv.data[,2:5],mfaces.pred$scores$scores)

    # prediction for different time windowes
    for(dt in deltaT){
        ith = ith + 1
        DP.id[[ith]] = tmp.id
        timeEvent[[ith]] = tmp.surv.data[, c("time", "event")]
        trueProb [[ith]] = tmp.surv.data$true.prob[, (which((t+dt)==obstime)-1)]
        DP.prob.rsf[[ith]] = cond.prob.pec(rsf.fit, test.surv.tmp, t, (t+dt))
        DP.prob.cox[[ith]] = cond.prob.pec(cox.fit, test.surv.tmp, t, (t+dt))
    }
}
DP.prob = DP.prob.rsf
save(sim.data, surv, train.id, trueProb, DP.id, DP.prob, timeEvent,
     file=paste(c("output/mfaces/",scenario,"/mfaces_rsf",i.run,".rdata"), collapse=""))

DP.prob = DP.prob.cox
save(sim.data, surv, train.id, trueProb, DP.id, DP.prob, timeEvent,
     file=paste(c("output/mfaces/",scenario,"/mfaces_cox",i.run,".rdata"), collapse=""))
```

```
}
```

3.3 Calculation of AUC and Brier score

```
library(survival)
library(abind)
source('functions.R')

method = "mfpca"
scenario = "none"
path = paste0("output/",method,"/",scenario,"/")

n.sim = 100       #number of simulation runs
n = 300           #sample size
n.train = 200     #n.test = n - n.train

#dynamic prediction information
obstime = seq(0,10,0.5)
Tstart = c(1,2,3,4) # landmark time for prediction
deltaT = c(1,2) # prediction windowns

row.names = cbind(rep(Tstart, each=length(deltaT)), rep(deltaT, length(Tstart)))

models = c("RSF","Cox")

listofTables = vector(mode="list", length=2)

i.model=1
for(model in models){

    #initalize empty matrix to store AUC/BS results
    auctrue.m = matrix(NA, nrow=nrow(row.names), ncol=n.sim)
    auc.m = matrix(NA, nrow=nrow(row.names), ncol=n.sim)
    BS.m = matrix(NA, nrow=nrow(row.names), ncol=n.sim)
    MSE.m = matrix(NA, nrow=nrow(row.names), ncol=n.sim)
    for(i.run in 1:n.sim){
        load(paste0(path,method,"_",model,i.run,".rdata"))

        train.surv = surv[which(surv$id %in% train.id),]

        #estimate km curve for BS calculation
        km = survfit(Surv(time, event)~1, data=train.surv)
        survest = stepfun(km$time, c(1, km$surv))

        ith=0
        for(t in Tstart){
            for(dt in deltaT){
```

```
            ith = ith + 1
            tp = t + dt

            test.surv = surv[surv$id%in%DP.id[[ith]], ]
            N_vali = nrow(test.surv)


            #TRUE AUC
            roc = tdROC( X = 1-trueProb[[ith]], Y = timeEvent[[ith]]$time,
                         delta = timeEvent[[ith]]$event,
                         tau = tp, span = 0.05,
                         nboot = 0, alpha = 0.05,
                         n.grid = 1000, cut.off = 0.5)

            auctrue.m[ith, i.run] = roc$AUC$value


            #AUC
            roc = tdROC( X = 1-DP.prob[[ith]], Y = timeEvent[[ith]]$time,
                         delta = timeEvent[[ith]]$event,
                         tau = tp, span = 0.05,
                         nboot = 0, alpha = 0.05,
                         n.grid = 1000, cut.off = 0.5)

            auc.m[ith, i.run] = roc$AUC$value


            #BRIER SCORE
            D = rep(0, N_vali)
            D[test.surv$time<=tp&test.surv$event==1] = 1
            pi = 1-DP.prob[[ith]]

            km_pts = survest(test.surv$time)/survest(t)
            W2 <- D/km_pts
            W1 <- as.numeric(test.surv$time>tp)/(survest(tp)/survest(t))
            W <- W1 + W2

            BS_pts <- W * (D - pi)^2
            BS.m[ith, i.run]  = sum(na.omit(BS_pts)) / N_vali
        }
    }
}

table = cbind(row.names,  apply(auctrue.m, 1, mean, na.rm=T),
    apply(auc.m, 1, mean, na.rm=T),apply(BS.m, 1, mean, na.rm=T),
    apply(MSE.m, 1, mean, na.rm=T))

colnames(table) = c("t", "dt", "True.AUC", "AUC", "BS", "MSE")
```

```
    print(model)
    print(table)
    listofTables[[i.model]] = table
    i.model = i.model+1
}

table = cbind(listofTables[[1]][,1:5],listofTables[[2]][,4:5])
print(table)
save(table, file = paste0("output/output_",method,'_',scenario,'.rdata'))
```

3.4 Data simulation and other support functions

```
    # simulation data for multivariate joint model linear (opt=["none","interaction"])
sim_mjm_linear = function(I,  obstime = 1:10, miss = FALSE, miss.rate = 0.1, opt = "none"){

    # I : number of subjects
    # J : number of visits
    # obstime: observation times
    # miss: whether introduce missing (missing complete at random) in longitudinal data.
    ##Different from drop-out
    # miss.rate: missing rate.

    J = length(obstime)
    N = I*J

    #### longitudinal submodel ####
    beta0 = c(1.5,2,0.5)
    beta1 = c(2,-1,1)
    betat = c(1.5, -1, 0.6)
    b.var = c(1,1.5,2)
    e.var = c(1,1,1)
    rho = c(-0.2,0.1,-0.3)
    b.Sigma = diag(b.var)
    b.Sigma[1,2] = b.Sigma[2,1] = sqrt(b.var[1]*b.var[2])*rho[1]
    b.Sigma[1,3] = b.Sigma[3,1] = sqrt(b.var[1]*b.var[3])*rho[2]
    b.Sigma[2,3] = b.Sigma[3,2] = sqrt(b.var[2]*b.var[3])*rho[3]

    # sample covariate
    X = rep(rnorm(I, 3, 1), each=J)
    # sample random effect
    ranef = mvrnorm(I, c(0,0,0), b.Sigma)
    id = rep(1:I,each=J)
    ranef = ranef[id,]
    # construct longitudinal submodel
    eta.long = matrix(0, nrow=N, ncol=3)
    for(i in 1:3)
```

```
    eta.long[,i] = beta0[i] + beta1[i]*X + ranef[,i]


#### survival submodel ####
#interaction
if(opt=="interaction"){
    gamma = c(-4,-2,4)
    alpha = c(0.2, -0.2, 0.4)
    x1 = rbinom(I, size = 1, prob=.5)
    x2 = rnorm(I)
    x3 = x1*x2
    W = cbind(x1,x2,x3)
    eta.surv = W%*%gamma + c(alpha%*%t(eta.long[!duplicated(id),]))
}

else{
    gamma = c(-4,-2)
    alpha = c(0.2, -0.2, 0.4)
    x1 = rbinom(I, size = 1, prob=.5)
    x2 = rnorm(I)
    W = cbind(x1,x2)
    eta.surv = W%*%gamma + c(alpha%*%t(eta.long[!duplicated(id),]))
}

#### simulate survival time ####
scale = exp(-7)
S = runif(I)
Ti = rep(NA, I)
alpha.beta = alpha%*%betat
f = function(tau){
    h = function(t) {
        scale *exp(eta.surv[i] + c(alpha.beta)*t)
    }
    S[i] - exp(-stats::integrate(h, 0, tau)$value)
}
f = Vectorize(f)

for(i in 1:I){
    Ti[i] = uniroot(f, c(0, 100))$root
}

#### simulate true survival probability ####
pre.surtime = function(tau){
    h = function(t) {
        scale *exp(eta.surv[i] + c(alpha.beta)*t)
    }

    exp(-stats::integrate(h, 0, tau)$value)
```

```
    }

    true.prob = matrix(NA, nrow=I, ncol=length(obstime[-1]))
    for(i in 1:I){
        ith = 0
        for(tau in obstime[-1]){
            ith = ith + 1
            true.prob[i, ith] = pre.surtime(tau)
        }
    }

    colnames(true.prob) = as.character(obstime[-1])

    #---------------------------------------
    # simulate censor time
    C = runif(I,min=obstime[3], max=obstime[5]+20)
    time <- pmin(Ti, C) #observed time is min of censored and true
    event <- ifelse(time==Ti, 1, 0) #0: censored ; 1: event;

    # prepare data
    visit = rep(1:J, I)
    obstime = rep(obstime, I)
    erro = mvrnorm(N, c(0,0,0), diag(e.var))
    Y = matrix(0, nrow=N, ncol=3)
    for(i in 1:3)
        Y[,i] = eta.long[,i] + betat[i]*obstime  + erro[,i]


    long.all = data.frame(id=id, visit=visit, time =
        rep(time, each=J), event = rep(event, each=J),
        Y1=Y[,1], Y2=Y[,2], Y3=Y[,3],obstime=obstime,
        X=X, ranef=ranef, W = W[rep(1:I, each=J)], erro=I(erro))

    long = long.all

    # introduce missing complete at random
    if(miss){
        miss.index = sample(which(long$obstime>obstime[2]), miss.rate*N)
        long = long[!c(1:N)%in%miss.index, ]
    }

    surv = data.frame(id = c(1:I),time=time, event=event, W = W, true.prob=I(true.prob))

    # remove observations after event or censoring
    long = long[long$obstime<long$time, ]

    return(list(long=long, surv=surv, long.all=long.all))
}
```

```r
# univariate FPCA via principal analysis by conditional estimation(PACE)
uPACE = function(testData, domain, predData=NULL, nbasis = 10, pve = 0.9, npc = NULL){

    tmp = funData(domain, testData)
    if(is.null(predData)){
        tmp2 = NULL
    }else{
        tmp2 = funData(domain, predData)
    }

    res = PACE(tmp, tmp2, pve=pve, npc= npc, nbasis=nbasis)
    return(res)
}

# multivariate FPCA based on results from uPACE
mFPCA = function(Xi, phi, p , L, I=I){

    # eigenanalysis on matrix M
    M = t(Xi)%*%Xi/(I-1)
    eigen.M = eigen(M)
    values = eigen.M$values
    pve = cumsum(values)/sum(values)
    Cms = eigen.M$vectors
    index = unlist(lapply(1:length(L), function(x) rep(x, L[x])))

    # MFPCA score
    rho = mfpca.score(Xi, Cms)

    # MFPCA eigenfunction
    psis = NULL
    for(j in 1:p){
        psi = NULL
        for(m in 1:dim(Cms)[2]){
            psi = cbind(psi, phi[[j]]%*%Cms[which(index==j),m])
        }
        psis[[j]] = psi
    }

    out = list(eigenvalue = values, Cms = Cms, pve = pve, index=index, rho = rho, psis=psis)

    return(out)
}
```

```r
# mfpc score calculation
mfpca.score = function(predXi, Cms){
    rho = matrix(NA, nrow = nrow(predXi), ncol=dim(Cms)[2])
    for(i in 1:nrow(predXi)){
        for(m in 1:dim(Cms)[2]){
            rho[i,m] = predXi[i,]%*%Cms[,m]
        }
    }
    return(rho)
}


# mfpc trajectories prediction
mfpca.pred = function(score, meanf, psi, n.rho=NULL){
    p = length(psi)
    n = nrow(score)

    if(is.null(n.rho)){
        n.rho = ncol(score)
    }

    pred = array(NA, c(n, length(meanf[[1]]), p))
    for(m in 1:p){
        pred[,,m] = matrix(meanf[[m]], nrow=n,
            ncol = length(meanf[[m]]), byrow = T ) + score[,1:n.rho]%*%t(psi[[m]][, 1:n.rho])
    }

    out = pred
    return(out)
}


#risk predict using predictSurvProb instead of survfit to handle RSF. (REQUIRES PEC)
cond.prob.pec = function(model, newdata, Tstart, Tpred){
    risk.Tstart = as.numeric(predictSurvProb(model,newdata=newdata,times=Tstart))
    risk.Tpred = as.numeric(predictSurvProb(model,newdata=newdata,times=Tpred))
    return(risk.Tpred/risk.Tstart)
}
```