1 Introduction to the Data

The primary sample set includes 8211 NBA games without overtime(s), covering from November 1st 2002 to November 28th 2014. The data of games with overtime(s) are also used to generate some of the results in the paper. In this file, the data of 8211 games without overtime(s) will be taken as an example to illustrate the code. For the games with overtime(s), the results can be generated by using the same code, but with the first-2880-second data.

The raw data of the 8211 games contain home team scores and away team scores at a frequency of every 30 seconds; the first record is taken at second 0, when both team scores are zeros, and the last record is taken at the 2880^{th} second, when the regulation period ends. Thus, 97 observations can be obtained for each team in one game. (Note: game 300130023 has non-zero score at $t = 0$ for the home team, which is 2. Considering its score recorded at the 30^{th} second is 2, the score at $t = 0$ can be changed to 0, without causing other effects. The above table is based on the data after this change.)

The data used in the paper is a 97×8211 panel (the ".csv" file is showing the data as 8211×97, in order to completely load the data with large dimension) of the score difference between the two teams, such that $X_j(t) = Home\ Score_j(t) - Away\ Score_j(t)$, where $t = 0, 0.5, 1, 1.5, ..., 48$, in terms of minutes, and $j = 1, 2, ..., 8211$.

Some statistical description of the score difference, X_j, are summarized in the following table:

Summary Statistics

Time	Mean	SD	Skewness	Kurtosis	Min	Max	Time	Mean	SD	Skewness	Kurtosis	Min	Max
1/96	0.07	1.48	-0.06	2.37	-5	4	49/96	2.10	10.29	-0.00	3.12	-36	44
2/96	0.11	2.09	-0.02	2.73	-7	8	50/96	2.11	10.39	0.01	3.12	-39	46
3/96	0.16	2.60	-0.03	2.84	-9	10	51/96	2.13	10.50	0.02	3.12	-37	46
4/96	0.21	3.03	-0.05	2.92	-11	10	52/96	2.17	10.62	0.01	3.13	-37	49
5/96	0.27	3.45	-0.06	2.91	-12	13	53/96	2.20	10.70	0.01	3.10	-38	49
6/96	0.30	3.79	-0.03	2.99	-14	15	54/96	2.22	10.80	0.02	3.04	-38	46
7/96	0.35	4.09	-0.04	2.98	-14	15	55/96	2.23	10.92	0.02	3.06	-36	47
8/96	0.42	4.37	-0.08	2.96	-15	16	56/96	2.31	11.02	0.01	3.05	-38	47
9/96	0.42	4.65	-0.00	3.01	-18	17	57/96	2.37	11.14	-0.00	3.09	-42	51
10/96	0.47	4.87	0.01	3.07	-18	21	58/96	2.41	11.27	-0.00	3.11	-42	54
11/96	0.50	5.13	0.00	3.09	-17	23	59/96	2.47	11.37	0.00	3.13	-42	54
12/96	0.58	5.40	0.00	3.12	-19	23	60/96	2.51	11.45	-0.00	3.16	-44	56
13/96	0.65	5.59	0.01	3.11	-20	25	61/96	2.58	11.55	-0.01	3.16	-46	54
14/96	0.69	5.80	-0.02	3.05	-20	25	62/96	2.63	11.65	-0.00	3.18	-43	54
15/96	0.74	6.00	-0.02	2.99	-20	24	63/96	2.63	11.75	-0.01	3.15	-43	51
16/96	0.81	6.22	-0.02	2.96	-20	23	64/96	2.65	11.86	-0.02	3.13	-46	51
17/96	0.84	6.40	-0.02	3.00	-23	25	65/96	2.68	11.98	-0.02	3.16	-46	49
18/96	0.90	6.55	-0.01	2.97	-25	25	66/96	2.71	12.07	-0.02	3.20	-48	51
19/96	0.94	6.72	-0.00	2.94	-23	25	67/96	2.79	12.13	-0.03	3.19	-46	50
20/96	0.98	6.89	-0.01	2.96	-24	24	68/96	2.77	12.16	-0.02	3.18	-44	51
21/96	1.05	7.04	-0.00	2.98	-24	26	69/96	2.83	12.24	-0.02	3.20	-46	51
22/96	1.10	7.17	0.01	3.00	-25	27	70/96	2.86	12.27	-0.02	3.23	-46	49
23/96	1.15	7.33	0.01	3.03	-27	28	71/96	2.89	12.32	-0.02	3.23	-49	49
24/96	1.17	7.50	-0.00	3.04	-26	32	72/96	2.95	12.41	-0.03	3.22	-49	51
25/96	1.18	7.62	-0.01	3.02	-25	34	73/96	2.97	12.47	-0.02	3.20	-47	51
26/96	1.20	7.73	0.01	3.05	-27	34	74/96	3.01	12.53	-0.02	3.18	-49	51
27/96	1.24	7.87	0.03	3.06	-28	37	75/96	3.02	12.56	-0.02	3.18	-47	49
28/96	1.29	8.02	0.01	3.07	-29	37	76/96	3.03	12.63	-0.02	3.16	-47	49
29/96	1.37	8.15	0.01	3.04	-31	35	77/96	3.07	12.67	-0.03	3.14	-47	50
30/96	1.40	8.28	0.00	3.02	-29	35	78/96	3.10	12.73	-0.03	3.13	-44	48
31/96	1.46	8.37	0.01	3.07	-30	36	79/96	3.12	12.81	-0.02	3.15	-45	49
32/96	1.47	8.49	0.01	3.06	-30	37	80/96	3.12	12.85	-0.03	3.12	-45	47
33/96	1.53	8.58	0.01	3.06	-32	37	81/96	3.12	12.91	-0.04	3.15	-47	48
34/96	1.56	8.70	0.00	3.04	-30	38	82/96	3.14	12.97	-0.03	3.16	-47	51
35/96	1.62	8.83	-0.00	3.07	-32	38	83/96	3.18	13.00	-0.03	3.17	-49	51
36/96	1.66	8.92	-0.01	3.04	-30	37	84/96	3.18	13.07	-0.02	3.13	-44	52
37/96	1.70	9.03	0.01	3.06	-33	39	85/96	3.24	13.09	-0.02	3.12	-45	52
38/96	1.73	9.12	0.02	3.07	-33	39	86/96	3.23	13.11	-0.02	3.11	-43	53
39/96	1.76	9.25	0.01	3.08	-32	41	87/96	3.26	13.14	-0.02	3.13	-46	51
40/96	1.82	9.35	0.01	3.08	-31	41	88/96	3.31	13.17	-0.02	3.12	-47	50
41/96	1.86	9.47	0.01	3.09	-31	41	89/96	3.31	13.19	-0.03	3.13	-47	52
42/96	1.89	9.59	0.02	3.11	-32	41	90/96	3.30	13.25	-0.03	3.10	-48	52
43/96	1.95	9.73	0.02	3.11	-31	43	91/96	3.32	13.25	-0.04	3.09	-48	52
44/96	1.98	9.83	0.02	3.09	-31	43	92/96	3.32	13.24	-0.05	3.11	-47	54
45/96	1.98	9.91	0.01	3.06	-34	41	93/96	3.29	13.20	-0.04	3.10	-48	54
46/96	2.00	10.00	0.02	3.07	-34	42	94/96	3.30	13.18	-0.04	3.11	-48	55
47/96	2.07	10.09	0.00	3.09	-36	41	95/96	3.33	13.22	-0.04	3.10	-50	55
48/96	2.10	10.20	-0.00	3.10	-36	44	1	3.31	13.12	-0.05	3.03	-50	55

2 Probabilities of Winning Given Leading at Time t, for Both Teams

The paper presents a summary for both teams' probabilities of winning, given leading at certain time points. The function computing these conditional probabilities is shown as follows.

```
getprwin <- function(x,final,lcon=c("1440","2160","2700","2820","2850"),
                     recnames=c("allHwin","allAwin","c1H","c1A","c2H","c2A","c3H","c3A",
                                "c4H","c4A","c5H","c5A"),
                     resnames=c("allHwin","allAwin","c1Hwin","c1Awin","c2Hwin","c2Awin",
                                "c3Hwin","c3Awin","c4Hwin","c4Awin","c5Hwin","c5Awin"))
{
  # This function computes the conditional probabilities of winning for each team.
  # "x" is the 97*8211 data matrix;
  # "final" is the score difference at the 2880th second;
  # "lcon" is a vector of time conditions, in terms of seconds.
  conx <- x[lcon,]
  nr <- 2*(length(lcon)+1)
  record <- matrix(0,nr,ncol(x))
  i <- 1
  while (i<=ncol(x))
  {
    if (final[i] > 0)
      record[1,i] <- 1
    if (final[i] < 0)
      record[2,i] <- 1
    for (k in 1:length(lcon))
    {
      if (conx[k,i] > 0)
        record[(2*k+1),i] <- 1
      if (conx[k,i] < 0)
        record[(2*k+2),i] <- 1
    }
    i <- i+1
  }
  rownames(record) <- recnames
  colnames(record) <- colnames(x)
  res <- record
  rownames(res) <- resnames
  for (j in 1:length(lcon))
  {
    res[(2*j+1),]<-record[1,]*record[(2*j+1),]
    res[(2*j+2),]<-record[2,]*record[(2*j+2),]
  }
  sumrec <- apply(record,1,sum)
  sumres <- apply(res,1,sum)
  prwin <- sumres/sumrec
  prwin[1] <- sumres[1]/ncol(x)
  prwin[2] <- sumres[2]/ncol(x)
  return(list(record=record,res=res,sumrec=sumrec,sumres=sumres,prwin=prwin))
}
```

The results can be generated as follows.

```
xf <- x[nrow(x),]

prwin <- getprwin(x,xf)$prwin
```

```
T3_prwin <- matrix(prwin,2,dimnames=list(c("Home Team","Away Team"),
                                          c("t=0","t=48/96","t=72/96",
                                            "t=90/96","t=94/96","t=95/96")))
```

Probabilities of Winning Given Leading at Time t, for Both Teams

	t=0	t=48/96	t=72/96	t=90/96	t=94/96	t=95/96
Home Team	0.6054	0.7989	0.8730	0.9389	0.9662	0.9778
Away Team	0.3946	0.6628	0.7980	0.9091	0.9476	0.9648

3 Hausman Type Test for BM

A Hausman-type test is employed in the paper to test the Brownian Motion assumption (Becker et al., 2007, Lemma 5). The test procedure and results are shown as follows.

Under the null hypothesis,

$$X_j(t) \overset{\mathrm{d}}{=} N(\mu t, \sigma^2 t),\ \forall j,\ \ \mu \in \Re,\ \ \sigma \in \Re^+,\ \ t \in [0,1],$$

and $X_j(t+\tau) - X_j(t)$ is independent of $X_j(t) - X_j(t-\tau)$ for $1 \geq t > \tau \geq 0$, and

$$X_j(t) - X_j(\tau) \overset{\mathrm{d}}{=} N(\mu(t-\tau), \sigma^2(t-\tau)).$$

A more general version test is used in the paper, with game specific μ_j and σ_j.

First, obtain the increments within 30 seconds for each game.

$$difx_j = X_j(\tfrac{1}{96}) - X_j(0), X_j(\tfrac{2}{96}) - X_j(\tfrac{1}{96}), X_j(\tfrac{3}{96}) - X_j(\tfrac{2}{96}), ..., X_j(1) - X_j(\tfrac{95}{96}),\ \forall j.$$

```
difx <- apply(x,2,diff) # the increments within 30 seconds
```

Then, estimate the means and the standard deviations of the increments for each game with Maximum Liklihood Estimation.

$$(\hat{\mu}_j, \hat{\sigma}_j^2) = \arg\max_{(u,\sigma^2)} \Pi_{t=1}^{96} \frac{1}{\sqrt{2\pi\frac{\sigma^2}{96}}} \exp\left\{-\frac{\left[X_j\left(\frac{t}{96}\right) - X_j\left(\frac{t-1}{96}\right) - \frac{\mu}{96}\right]^2}{2\frac{\sigma^2}{96}}\right\}.$$

```
estmle <- function(y,mu0=1,sigma0=1)
{
  # MLE estimation for the mean and the variance.
  # "y" is the increments within a given bandwidth, in one game;
  # "mu0" is the initial guess for the mean;
  # "sigma0" is the initial guess for the standard deviation.
  LL <- function(mu,sigma) lik <- -sum((-(y-mu)^2/(2*sigma^2))-log(2*pi*sigma^2)/2)
  res <- mle2(LL,start=list(mu=mu0,sigma=sigma0),data=list(y))
  mu <- coef(summary(res))["mu","Estimate"]
  sigma2 <- (coef(summary(res))["sigma","Estimate"])^2
  return(list(mu=mu,sigma2=sigma2))
}
```

```
n <- ncol(x)
# Number of games

res <- t(sapply(1:n,function(i) estmle(difx[,i])))
```

With the results from the estimation, Z_j's and V_j's can be computed, such that

$$Z_j = \max_t X_j(t),\ \forall j;\ V_j = 2Z_j(Z_j - X_j(1)),\ \forall j.$$

```
lg <- 1
# Normalize the length of game to 1

tau <- 1/96
# The bandwidth of taking the increments

mu_t <- as.numeric(res[,"mu"])

sigma2_t <- as.numeric(res[,"sigma2"])

mu <- mu_t/tau

sigma2 <- sigma2_t/tau

Z <- apply(x,2,max)

xf <- x[nrow(x),]

V <- 2*Z*(Z-xf)
```

And thus, the test statistic can be computed, such that

$$W_n = \sqrt{3n}\frac{S_n - 1}{S_n + 2} \xrightarrow{\text{d}} N(0,1), \ \text{as} \ n \to \infty, \text{where } S_n = \frac{\sum_{j=1}^{n}\left(\frac{X_j(1)-\hat{\mu}_j}{\hat{\sigma}_j}\right)^2}{\sum_{j=1}^{n}\frac{V_j}{\hat{\sigma}_j^2}}.$$

```
Sn <- sum((xf[c(1:n)]-mu)^2/sigma2)/sum(V[c(1:n)]/sigma2)
Wn <- sqrt(3*n)*(Sn-1)/(Sn+2)
table <- matrix(c("30-sec",sprintf("%.1f",Wn),n),nrow=1,
                dimnames=list(c("Statistics"),c("Time Unit","$W_n$","Sample Size")))
```

Hausman Test Statistics

	Time Unit	$W_n$	Sample Size
Statistics	30-sec	-78.5	8211

4 Functional Data Recovery

The following functions recover the score difference to functional data.

```
getEst_nopen <- function(x,t,K,PD,rangeval,dropK=1,create_basis=create.bspline.basis)
{
    # This function evaluates the objective function, m, given a number
    # of basis functions, K, without roughness penalty;

    # The function requires the number of basis functions, K, not exceeding
    # the number of observing point, t.
    #--------------------------------------------------------------------

    # "x" is the data matrix;
    # "t" is the vector of observing time points, in terms of normalized time domain
    # (normalized to [0,1]);
    # "K" is the number of basis functions;
    # "PD" is the order of derivatives for the penalty;
    # "rangeval" is the normalized time domain ([0,1]);
    # "dropK" is drop the 'dropK'th basis function (drop the 1st one);
    # "create_basis" is the type of basis function (bspline).
    #------------------------------------------------------------------

    # The function return a list:
    # "xfdPar" is a functional parameter object;
    # "xfd" is a functionl data object;
    # "m" is the value of the objective function, 1-by-1 numeric;
    # "bias" is the mean of the squared bias.

    xbasis <- create_basis(rangeval,K,dropind=dropK)
    xfdPar <- fdPar(xbasis,PD,0)
    xfd <- smooth.basis(t,x,xfdPar)$fd
    allC <- rbind(0,xfd$coefs)
    estfd <- fd(allC[-1,],xbasis)
    estfdt <- eval.fd(t,estfd)
    rownames(estfdt) <- t
    m <- mean(apply((estfdt-x)^2,2,mean))
    bias <- mean(apply((estfdt-x)^2,2,mean))
    cat("\n","K=",K,"; m=",m)
    return(list(xfdParX=xfdPar,xfd=xfd,m=m,bias=bias))
}
```

```
getEst_pen <- function(para,x,t,PD,rangeval,dropK=1,create_basis=create.bspline.basis)
{
    # This function evaluates the objective function, m, given a number
    # of basis functions, K, and a smoothing paramether, lambda.
    #------------------------------------------------------------------

    # "para" is a length-2 list, of the given K and log(lambda);
    # "x" is the data matrix;
    # "t" is the vector of observing time points, in terms of normalized time domain
    # (normalized to [0,1]);
    # "K" is the number of basis functions;
    # "PD" is the order of derivatives for the penalty;
    # "rangeval" is the normalized time domain ([0,1]);
    # "dropK" is drop the 'dropK'th basis function (drop the 1st one);
    # "create_basis" is the type of basis function (bspline).
    #------------------------------------------------------------------

    # The function return a list:
    # "xfdPar" is a functional parameter object;
    # "xfd" is a functionl data object;
    # "allpen" is a vector of roughness penalty;
    # "m" is the value of the objective function, 1-by-1 numeric;
    # "bias" is the mean of the squared bias;
    # "pen" is the mean of the roughness penalty.

    lambda <- 10^(para$loglam)
    K <- para$K
    xbasis <- create_basis(rangeval,K,dropind=dropK)
    xfdPar <- fdPar(xbasis,PD,lambda)
    xfd <- smooth.basis(t,x,xfdPar)$fd
    allC <- rbind(0,xfd$coefs)
    Rmat <- bsplinepen(xbasis,Lfdobj=2,rng=rangeval)
    allpen <- sapply(1:ncol(allC),function(i)
        crossprod(allC[,i],Rmat%*%allC[,i]))
    estfd <- fd(allC[-1,],xbasis)
    estfdt <- eval.fd(t,estfd)
    rownames(estfdt) <- t
    m <- mean(apply((estfdt-x)^2,2,mean)+lambda*allpen)
    bias <- mean(apply((estfdt-x)^2,2,mean))
    pen <- mean(allpen)
    cat("\n","K=",K,"; lambda=",lambda,"; m=",m)
    return(list(xfdParX=xfdPar,xfd=xfd,allpen=allpen,m=m,
                bias=bias,pen=pen))
}
```

```
getallres <- function(allK_Lam,x,t,PD,rangeval)
{
    # This function collects the results from the evaluation of the
    # "getEst_pen" function, on a K-lambda grid.
    #------------------------------------------------------------------

    # "allK_Lam" is a list expending all combinations of alternative K's and lambda's;
    # "x" is the data matrix;
    # "t" is the vector of observing time points, in terms of normalized time domain
    # (normalized to [0,1]);
    # "PD" is the order of derivatives for the penalty;
    # "rangeval" is the normalized time domain ([0,1]).
    #------------------------------------------------------------------

    # The function return a list:
    # "res" is a list of results from the following "f" function, for each K;
    # "allm" is a length(lambda)-by-length(K) matrix of the objective function values;
    # "allbias" is a length(lambda)-by-length(K) matrix of the mean of squared bias;
    # "allpen" is a length(lambda)-by-length(K) matrix of the mean of roughness penalty.

    f <- function(klam)
    {
        # This function evaluates the "getEst_pen" function for a vector
        # of lambda, under the same K.
        #--------------------------------------------------------------

        # "klam"-- a matrix of K and lambda pairs, with the same K and
        # all alternative lambda's.
        #--------------------------------------------------------------

        # The function return a list:
        # "res" -- a list of results from the "getEst_pen" function, for
        # each pair of the given parameters;
        # "m"   -- a vector of the objective function values;
        # "bias"-- a vector of the mean of squared bias;
        # "pen" -- a vector of the mean of roughness penalty.
        res <- sapply(1:ncol(klam),function(i)
            getEst_pen(klam[,i],x,t,PD,rangeval),simplify=F)
        m <- sapply(1:length(res),function(i) res[[i]]$m)
        bias <- sapply(1:length(res),function(i) res[[i]]$bias)
        pen <- sapply(1:length(res),function(i) res[[i]]$pen)
        return(list(res=res,m=m,bias=bias,pen=pen))
    }
    resf <- sapply(1:length(allK_Lam),function(i)
        f(allK_Lam[[i]]),simplify=F)
    allm <- sapply(1:length(allK_Lam),function(i) resf[[i]]$m)
    allbias <- sapply(1:length(allK_Lam),function(i) resf[[i]]$bias)
    allpen <- sapply(1:length(allK_Lam),function(i) resf[[i]]$pen)
    return(list(allm=allm,allbias=allbias,allpen=allpen))
}
```

```
getEst <- function(x,t,optpara,PD,rangeval,dropK=1,
                   create_basis=create.bspline.basis)
{
    # This function recover functional data wth the optimal parameters.
    #--------------------------------------------------------------------

    # "x" is the data matrix;
    # "t" is the vector of observing time points, in terms of normalized time domain
    # (normalized to [0,1]);
    # "optpara" is a length-2 list of the optimal parameters, K and lambda;
    # "PD" is the order of derivatives for the penalty;
    # "rangeval" is the normalized time domain ([0,1]);
    # "dropK" is drop the 'dropK'th basis function (drop the 1st one);
    # "create_basis" is the type of basis function (bspline).
    #--------------------------------------------------------------------

    # The function return a list:
    # "xfdPar" is a functional parameter object;
    # "xfd" is a functionl data object.
    lambda <- optpara$lambda
    K <- optpara$K
    xbasis <- create_basis(rangeval,K,dropind=dropK)
    xfdPar <- fdPar(xbasis,PD,lambda)
    xfd <- smooth.basis(t,x,xfdPar)$fd
    return(list(xfdParX=xfdPar,xfd=xfd))
}
```

The recovery process requires the following arguments.

```
rangeval <- c(0,1)
# Normalizd time domain

t <- seq(0,1,by=1/96)
# Normalized observing time points

PD <- 2
# Use the second derivatives to compute the roughness penalty matrix.

K_from <- 99
K_to <- 200
allK <- seq(K_from,K_to,by=1)
# K vector

loglam_from <- -20
loglam_to <- -1
nlam <- 200
loglam <- seq(loglam_from,loglam_to,len=nlam)
# log(Lambda) vector

K_Lam <- function(K,loglam)
```

```
  sapply(1:length(loglam),function(i) list("K"=K,"loglam"=loglam[i]))
allK_Lam <- sapply(1:length(allK),function(i) K_Lam(allK[i],loglam),simplify=F)
# Get a K-lam grid
```

With the arguments defined above, the optimal number of basis functions, K, and the optimal smoothing parameter, λ, can be obtained from the following process.

```
#====================================
# Recovery without roughness penalty
#====================================
res_nopen <- sapply(1:length(allK),function(i)
  getEst_nopen(x,t,allK[i],PD,rangeval),simplify=F)

minm <- sapply(1:length(res_nopen),function(i) res_nopen[[i]]$m)
# Get the minimal value of the objective function for each K

K_minm <- cbind("K"=allK,"m"=minm)
optK <- allK[which.min(K_minm[,"m"])]
# Get the optimal K

optlam <- 0
# Get the optimal lambda

#=================================
# Recovery with roughness penalty
#=================================
res <- getallres(allK_Lam,x,t,PD,rangeval)

minm <- apply(res$allm,2,min)
# Get the minimal value of the objective function over lambda, for each K

K_minm <- cbind("K"=allK,"m"=minm)
optK <- K_minm[,"K"][which.min(K_minm[,"m"])]
# Get the optimal K

lam_m <- cbind("lambda"=(10^loglam),"m"=res$allm[,which.min(minm)])
optlam <- lam_m[,"lambda"][which.min(lam_m[,"m"])]
# Get the optimal lambda for the optimal K
```

With the optimal K and λ, we can recover the functional data,

```
optpara <- list(K=optK,lambda=optlam)

Fitting <- getEst(x,t,optpara,PD,rangeval)
```

and plot the results.

```
plot(Fitting$xfd)
# Plot all fitted functions.

plot(mean(Fitting$xfd))
# Plot the mean of all fitted functions.

plotfit.fd(x,t,Fitting$xfd,lwd=1.2,index=NULL,type="l")
# Plot fitted functions and observations.

plot(K_minm,type="b",xlab=expression(italic(""*K*"")),ylab=expression(italic(""*m*"")),
     lwd=2,cex=0.6,cex.lab=1.5,cex.axis=1.5)
# Plot m-K

plot(lam_m,type="b",xlab=expression(italic(""*lambda*"")),
     ylab=expression(italic(""*m*"")),lwd=2,cex=0.6,cex.lab=1.5,cex.axis=1.5)
# Plot m-Lambda
```

5 Momentum with Fitted Data

A discussion on momentum and winning probabilities is presented in the paper, using the fitted functional data. The computation procedure of the coresponding statistics is shown as follows.

```
incr <- 6/96
# The bandwidth of taking increments

cm <- 11.5
# The critique value of determining momentum
```

The increments within the given bandwidth can be obtained.

The following function takes records for momentum and the results of the game.

```
get_m_win <- function(x,difx,m=cm)
{
  # This function takes records for momentum and the results of the game.
  # "x" is the 97*8211 matrix of the fitted data;
  # "difx" is the matrix of increments within the given bandwidth;
  # "m" is the critique value for defining momentum.
  getRec_m <- function(a,b)
  {
    btemp <- b
    i <- 1
    while (i <= length(a))
    {
      if (a[i] > m)
        btemp[i] <- 1
      if (a[i] < (-m))
        btemp[i] <- (-1)
      i <- i+1
    }
    return(btemp)
  }
  all_m <- sapply(1:ncol(difx),function(j) getRec_m(difx[,j],rep(0,nrow(difx))))
  dimnames(all_m) <- dimnames(difx)
  getRec_win <- function(a,b)
  {
    btemp <- b
    if (a > 0.5)
      btemp[1] <- 1
    if (a < (-0.5))
      btemp[2] <- 1
    return(btemp)
  }
  all_win <- sapply(1:ncol(difx),function(j) getRec_win(x[nrow(x),j],rep(0,2)))
  dimnames(all_win) <- list(c("Home Teams Win","Away Teams Win"),colnames(difx))
  return(list(all_m=all_m,all_win=all_win))
}
```

```
all_m <- get_m_win(x,difx)$all_m
# The records of momentum

all_win <- get_m_win(x,difx)$all_win
# The records of game results
```

```
which1M <- which(apply(abs(all_m),2,sum)==1)
# Games with exactly one momentum

which1MH <- which(apply(all_m[,which1M],2,sum)==1)
# Games with the momentum from the home team

which1MA <- which(apply(all_m[,which1M],2,sum)==-1)
# Games with the momentum from the away team

count1M <- length(which1M)
# Number of games with exactly one momentum

count1MH <- length(which1MH)
# Number of games with the momentum from the home team

count1MA <- length(which1MA)
# Number of games with the momentum from the away team

prob1MH <- 100*count1MH/count1M

prob1MA <- 100*count1MA/count1M

count1MHwinH <- all_win["Home Teams Win",which1M][which1MH]
# All games, where the momentum is from the home team with the home teams winning

prob1MHwinH <- 100*sum(count1MHwinH)/count1MH
# If a home team has the momentum, the probability of its winning

count1MAwinA <- all_win["Away Teams Win",which1M][which1MA]
# All games, where the momentum is from the away team with the away teams winning

prob1MAwinA <- 100*sum(count1MAwinA)/count1MA
# If an away team has the momentum, the probability of its winning
```

Exactly One Momentum

	Statistics
Number of games with exactly one momentum	884
% of games with the momentum from the home team	57.24
% of home winnings having the momentum	78.85
% of away winnings having the momentum	70.63