Experiment Journal for FPCA on Cartesian Data

Contents

[Experiment 1: PCA on coefficients of basis function 3](#_Toc408842599)

[1. from functional data to smooth function 3](#_Toc408842600)

[2. Apply standard PCA on coefficients of basis functions 6](#_Toc408842601)

[Experiment 2: Cartesian Velocity Reconstruction 7](#_Toc408842602)

[Description 7](#_Toc408842603)

[Implementation of smoothing and reconstruction: 7](#_Toc408842604)

[Interpretation of the results: 8](#_Toc408842605)

[Results with different parameters: 10](#_Toc408842606)

## Experiment 1: PCA on coefficients of basis function

Experimenter: Han

Date: Dec 19, 2014

### 1. from functional data to smooth function

Methodology: using a set of pre-defined basis function to fit data

Test data: root trajectory of 100 walk\_leftStance samples, the trajectory is represented by a set of 3d points in Cartesian space

Basis function: cubic Bspline function

Test variable: number of basis functions

Evaluation: mean square error (MSE) between samples from smooth function and original data

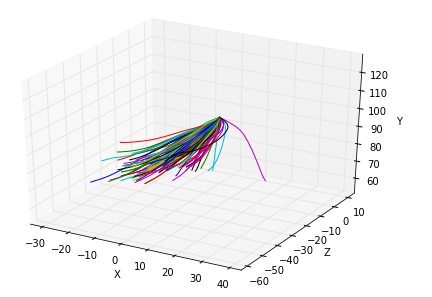


Figure 1.1: Trajectory of 100 root joints of left stance walking

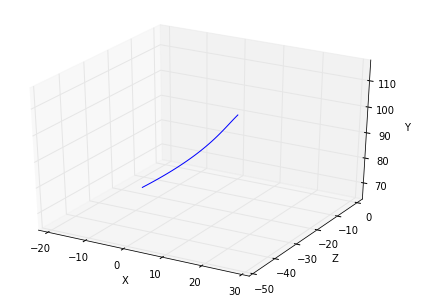


Figure 1.2: Mean curve

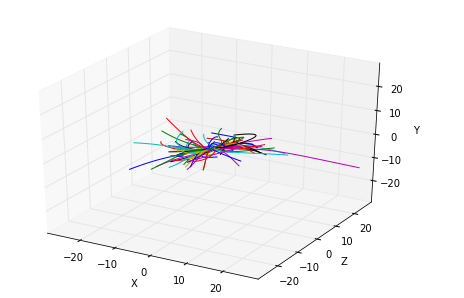


Figure 1.3: Centralized trajectories

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Number of basis functions | 4 | 5 | 6 | 7 | 8 | 9 |
| MSE | 0.464282312651 | 0.261261259745 | 0.200839389084 | 0.163553656099 | 0.133130771358 | 0.110511147885 |

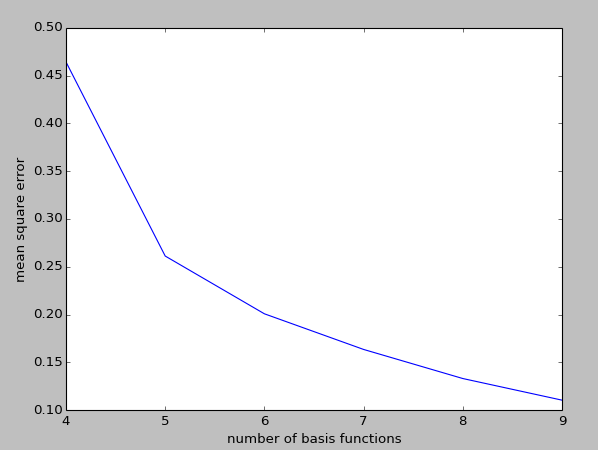
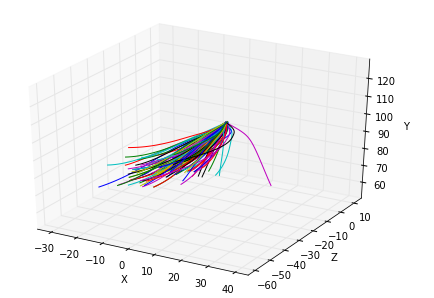
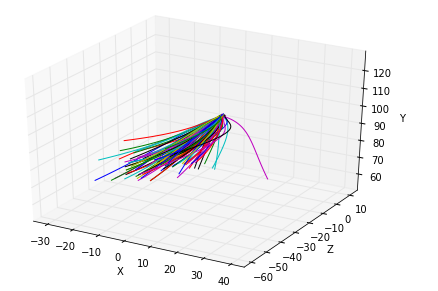
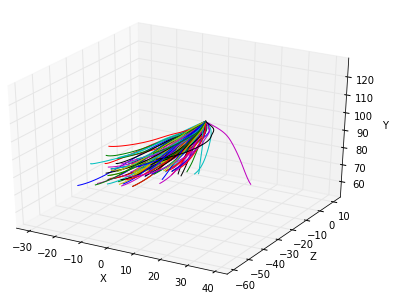
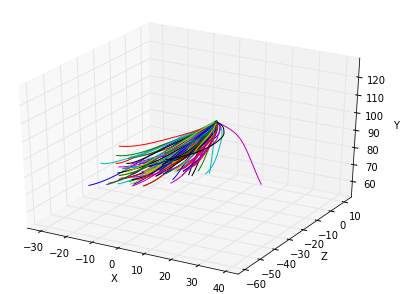


Figure 1.4: The change of MSE with different number of basis functions



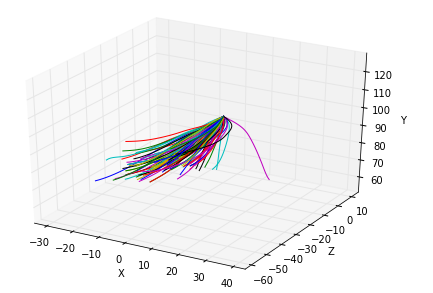
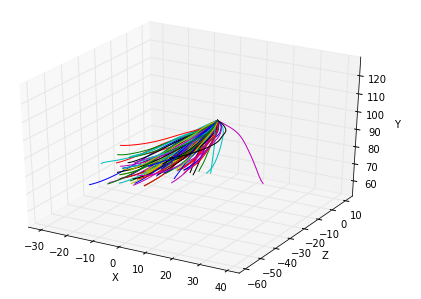
N\_basis = 5

N\_basis = 4



N\_basis = 7

N\_basis = 6



N\_basis = 9

N\_basis = 8

Figure 1.5: Reconstruct data with different number of basis functions

Conclusion: using 5 basis function is sufficient to represent original data.

### 2. Apply standard PCA on coefficients of basis functions

Methodology: concatenating coefficients of each dimension as a long vector for one motion sample, convert fpca as a standard multivariable PCA.

Test data: walk\_leftStance, 620 samples; and 47 frames, 57 dimensions for each sample

Test variable: number of principal components

Evaluation: mean square error between original coefficients and back projected coefficients

|  |  |  |
| --- | --- | --- |
| Maintain of information | Number of principal components | MSE per coefficent |
| 80% | 4 | 0.136978057192 |
| 85% | 5 | 0.11445191908 |
| 90% | 7 | 0.0949070670076 |
| 95% | 11 | 0.0714267637763 |
| 96% | 13 | 0.0633519672534 |
| 97% | 16 | 0.0536051589743 |
| 98% | 20 | 0.0445693885695 |
| 99% | 29 | 0.0317768584377 |

## Experiment 2: Cartesian Velocity Reconstruction

Experimenter: Erik

Date: Dec 19, 2014

### Description

Smoothing and reconstruction of a sample of aligned velocity data of the X, Y, Z coordinates of the root joint for a left step walking motion. The smoothing and reconstruction was done using FDA functions and the reconstruction based on FPCA with 5 harmonics was done in combination with linear model fitting for a linear combination of the resulting harmonics/eigenfunctions.

Properties of the data:

numframes = 46

replications = 20

ndim = 3

degree = 4

### Implementation of smoothing and reconstruction:

|  |
| --- |
| #create smooth function representation for each replication  fdarange = c(0, numframes)  fdabasis = create.bspline.basis(fdarange, nkots, degree)  fdatime = seq(0, numframes, len=numframes)  fdafd = smooth.basis(fdatime, arr, fdabasis)$fd  fdafd$fdnames[[1]] = "Milliseconds"  fdafd$fdnames[[2]] = "Replications"  fdafd$fdnames[[3]] = list("X", "Y","Z")  # set the number of eigenfunctions/harmonics to four and perform fPCA  fdapcaList = pca.fd(fdafd, nharm)  #plot.pca.fd(fdapcaList)  fdarotpcaList = varmx.pca.fd(fdapcaList)  print(dim(fdarotpcaList$scores))  # extract the sample, mean and harmonic coefficients and stack them together  samplecoefs = fdafd$coefs  harmonics <-fdarotpcaList$harmonics  mean <-fdarotpcaList$mean  #b= coef(fdafd[3]['X'])#$coefs  e1x = coef(harmonics[1]['value1'])  e2x =coef(harmonics[2]['value1'])  e3x =coef(harmonics[3]['value1'])  e4x =coef(harmonics[4]['value1'])  e5x =coef(harmonics[5]['value1'])  e1y = coef(harmonics[1]['value2'])  e2y =coef(harmonics[2]['value2'])  e3y =coef(harmonics[3]['value2'])  e4y =coef(harmonics[4]['value2'])  e5y =coef(harmonics[5]['value2'])  e1z = coef(harmonics[1]['value3'])  e2z =coef(harmonics[2]['value3'])  e3z =coef(harmonics[3]['value3'])  e4z =coef(harmonics[4]['value3'])  e5z =coef(harmonics[5]['value3'])  print(dim(mean$coefs))  mx = mean$coefs[,1,1]  my = mean$coefs[,1,2]  mz = mean$coefs[,1,3]  m = c(mx,my,mz)  sx = samplecoefs[,sampleidx,1]  sy = samplecoefs[,sampleidx,2]  sz = samplecoefs[,sampleidx,3]  e1 = c(e1x,e1y,e1z)  e2 = c(e2x,e2y,e2z)  e3 = c(e3x,e3y,e3z)  e4 = c(e4x,e4y,e4z)  e5 = c(e5x,e5y,e5z)  print(dim(mean$coefs))  mx = mean$coefs[,1,1]  my = mean$coefs[,1,2]  mz = mean$coefs[,1,3]  m = c(mx,my,mz)  s = c(sx,sy,sz)  centered = s - m #substract the mean from the sample  #do linear model regression for the stacked xy coeeficient vectors  x = lm(centered ~ e1 + e2 + e3 + e4 + e5)  print(x)  #construct the new coefficients using the calculated weights for each eigen function/harmonic  x = x$coef  newcoefx = x[1]\*e1x + x[2]\*e2x + x[3]\*e3x+ x[4]\*e4x+ x[5]\*e5x +mx  newcoefy = x[1]\*e1y + x[2]\*e2y + x[3]\*e3y+ x[4]\*e4y+ x[5]\*e5y +my  newcoefz = x[1]\*e1z + x[2]\*e2z + x[3]\*e3z+ x[4]\*e4z+ x[5]\*e5z +mz  newfx = fd(coef =newcoefx ,basisobj = harmonics$basis)  newfy = fd(coef =newcoefy ,basisobj = harmonics$basis)  newfz = fd(coef =newcoefz ,basisobj = harmonics$basis) |

### Interpretation of the results:

The data is very noisy and could not be reconstructed correctly. Overfitted splines that use as many control points as frames seem to be visually more accurately reconstructed, however the smoothing is not correct.

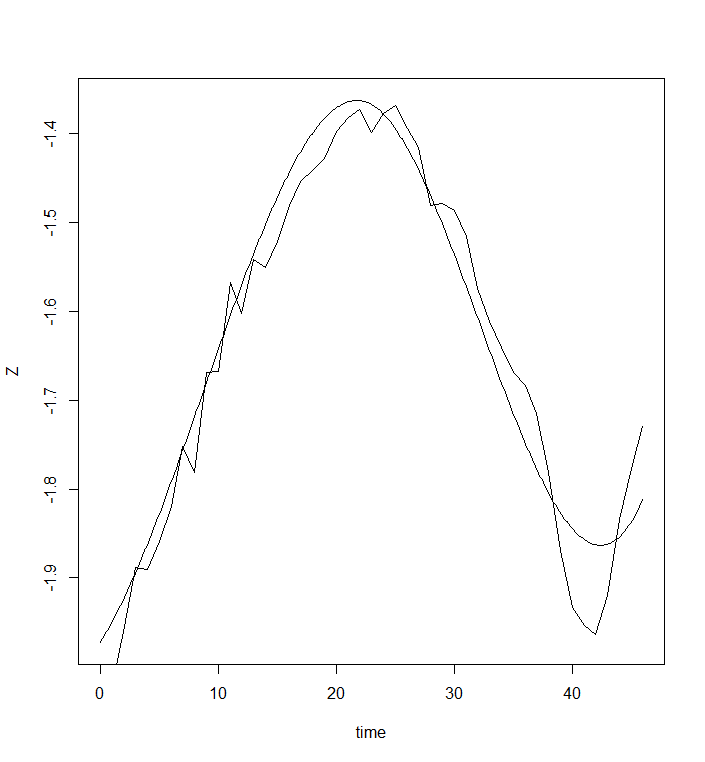
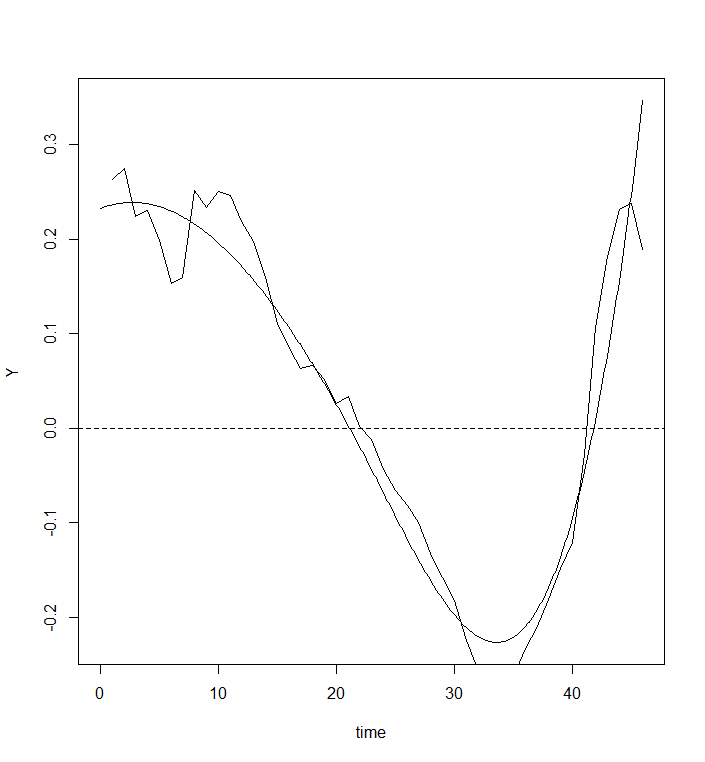
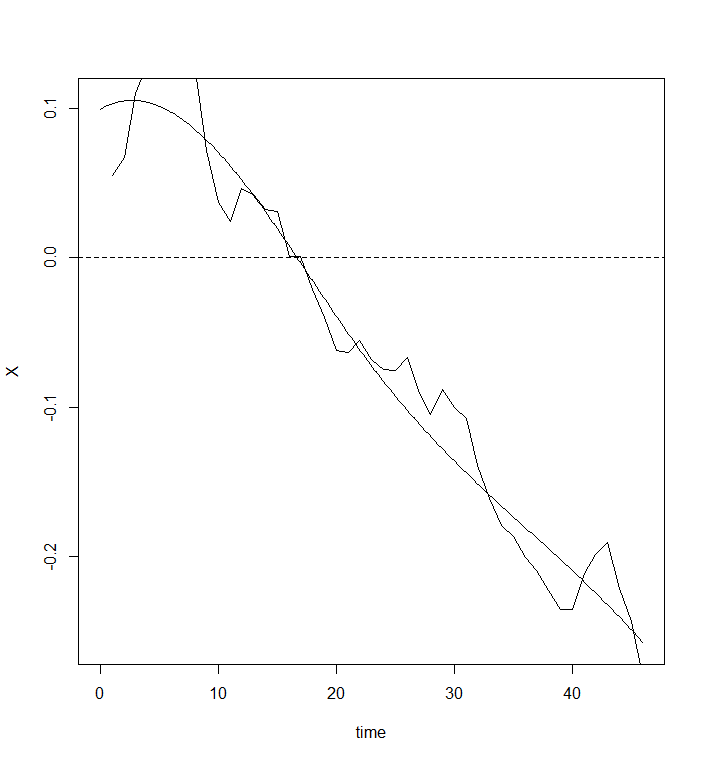
### Results with different parameters:

#### Test 1

Smoothing

numknots = 5, Degree = 4

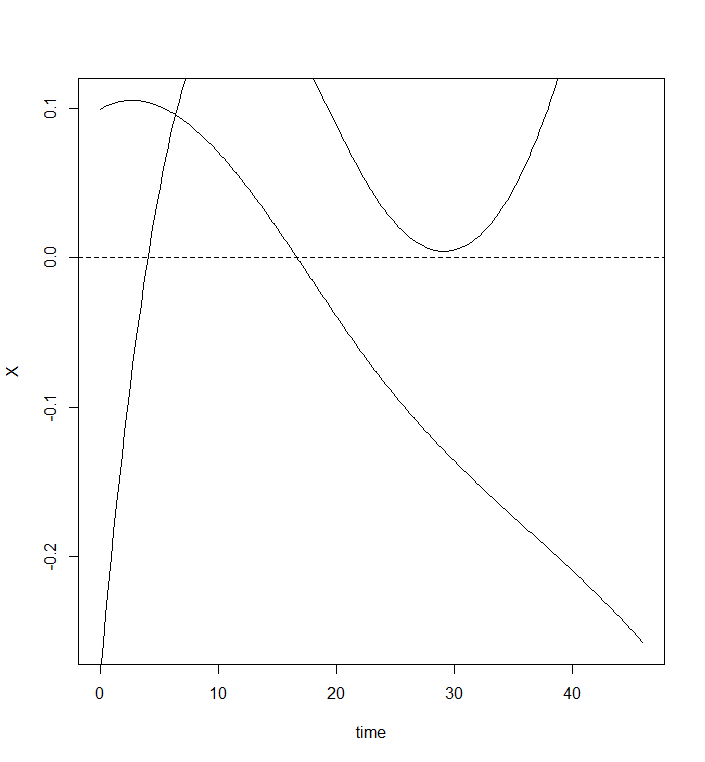
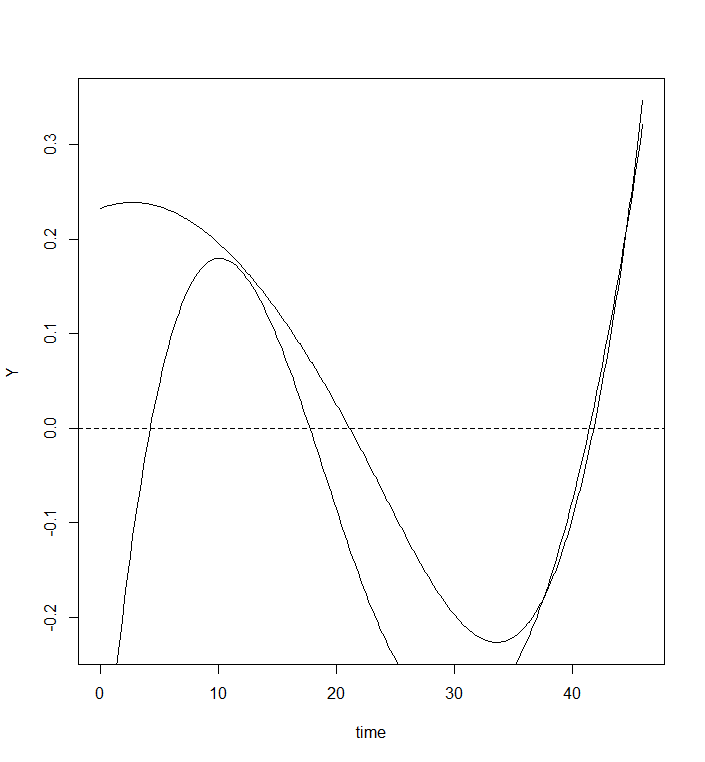
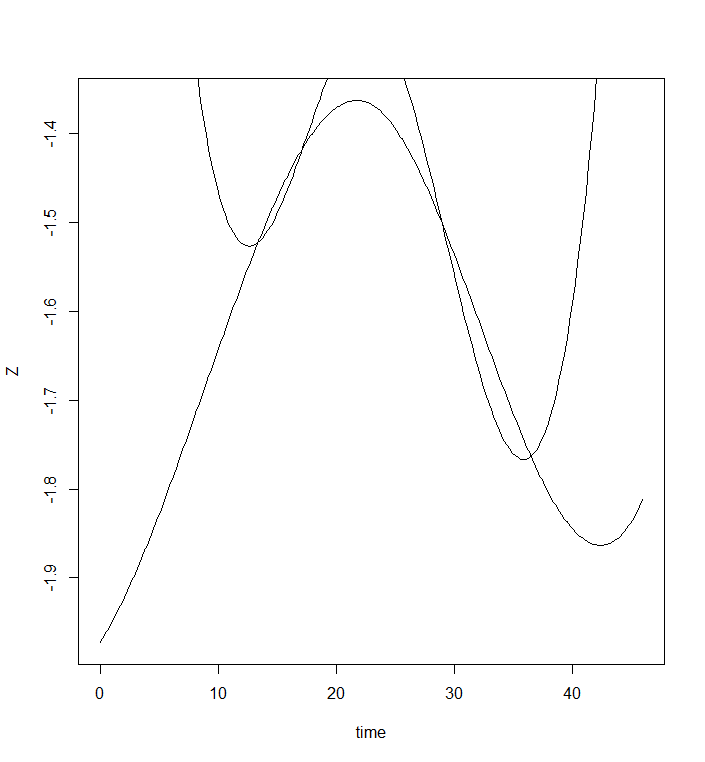
sample 1, Mean squared error: 0.06012635



Reconstruction

numknots = 5, Degree = 4, nharm= 5

sample 1, Mean squared error 0.5101568

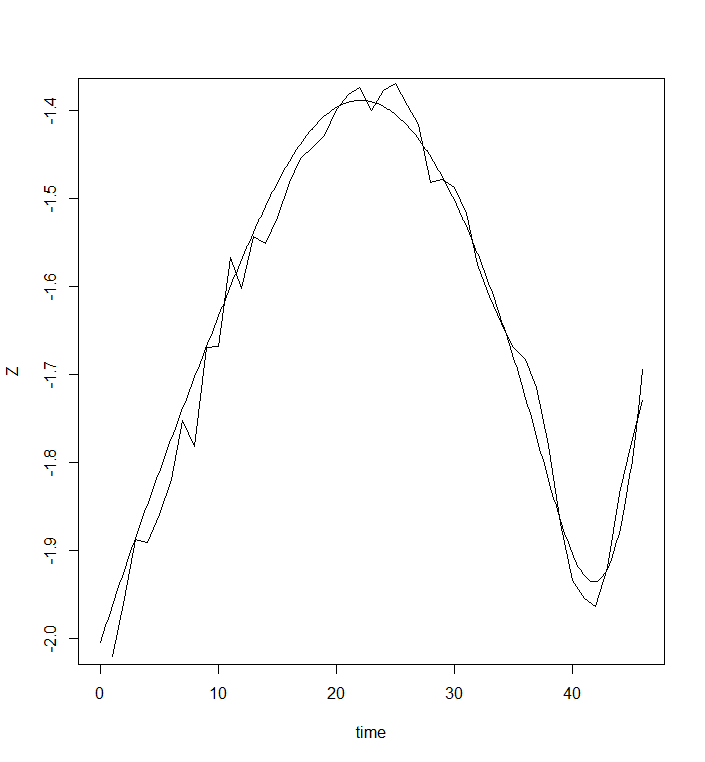
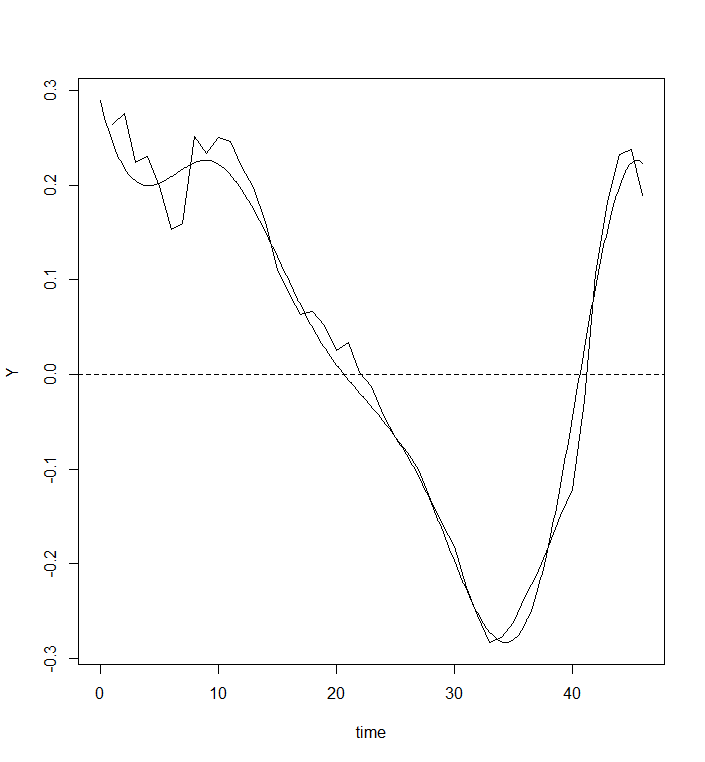
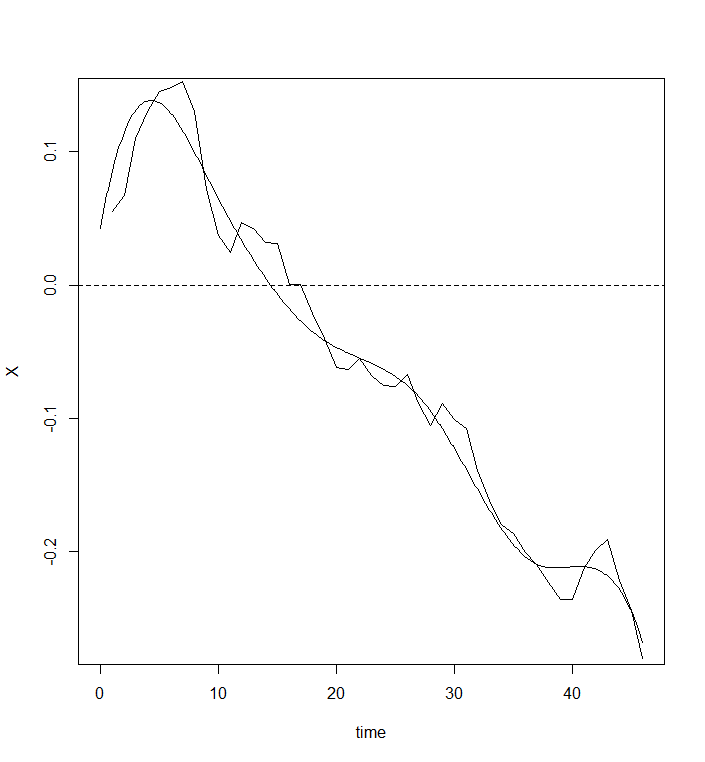
  

#### Test 2

Smoothing

numknots = 8, Degree = 4

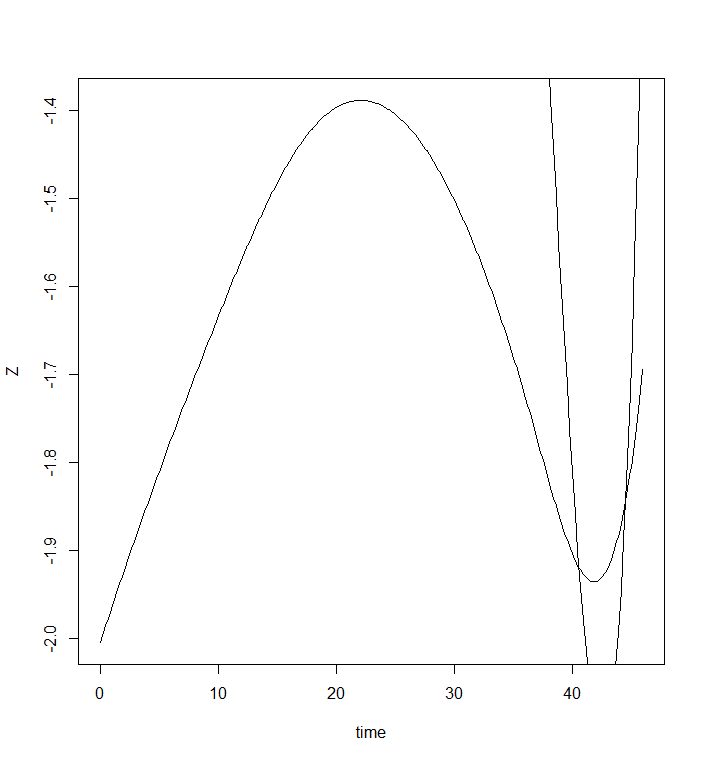
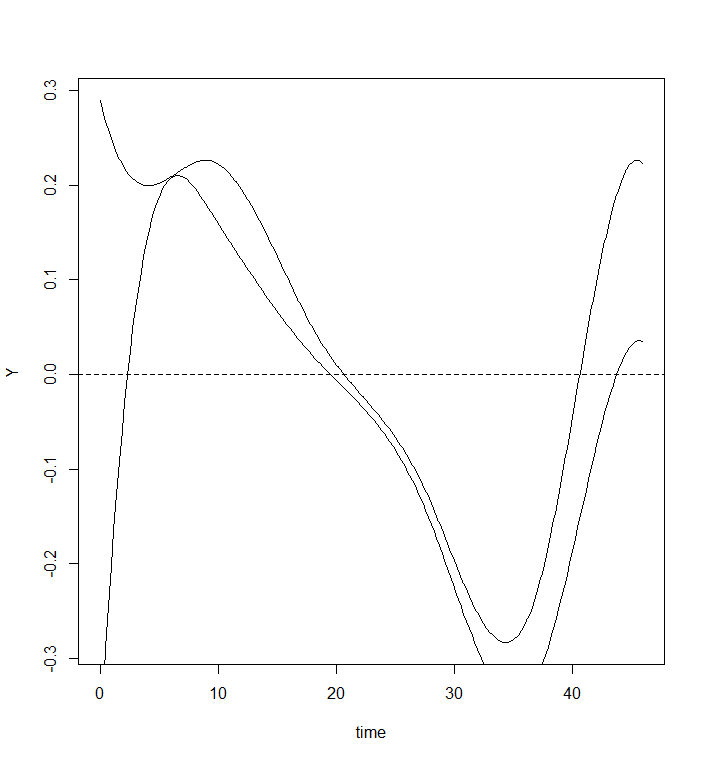
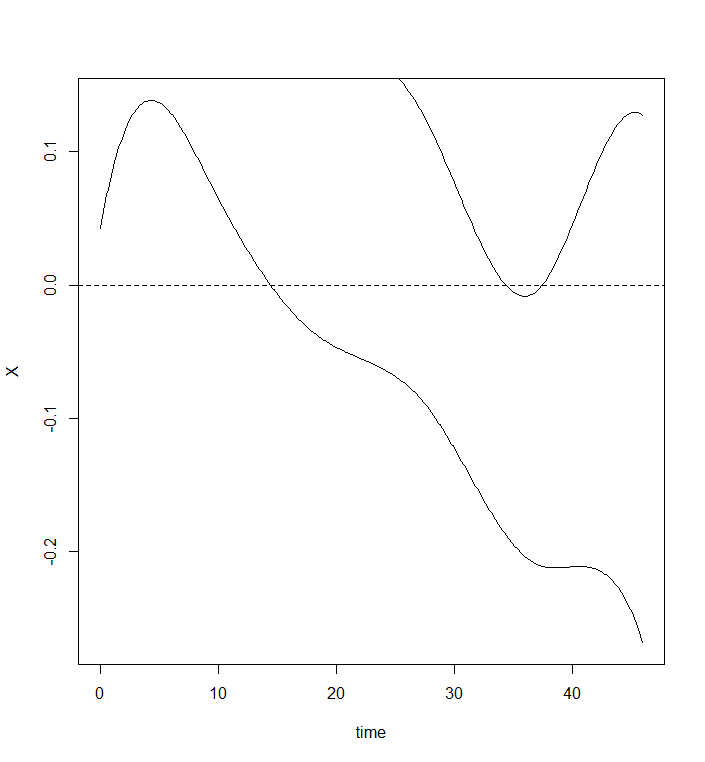
sample 1, Mean squared error 0.0451103



Reconstruction

numknots = 8, Degree = 4, nharm= 5

sample 1, Mean squared error 0.6531706

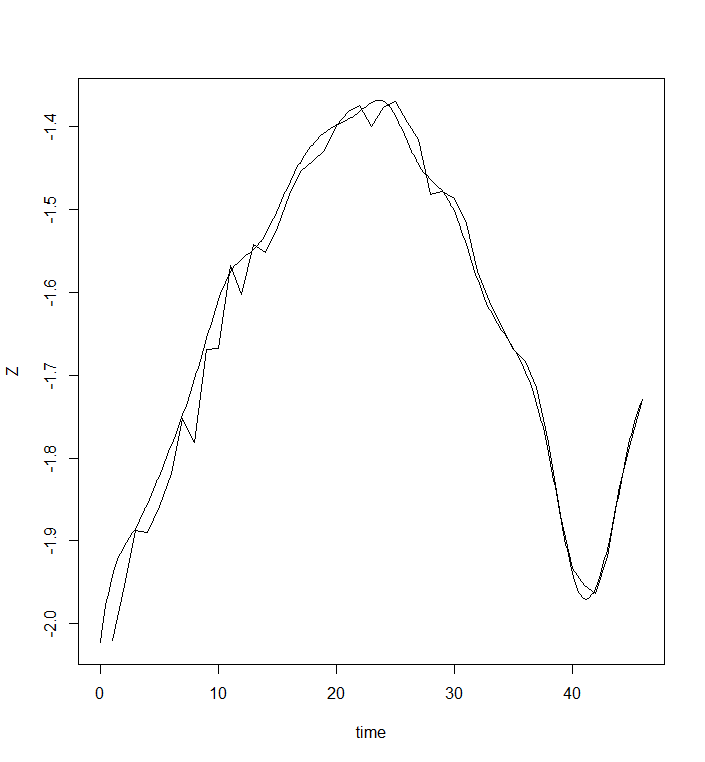
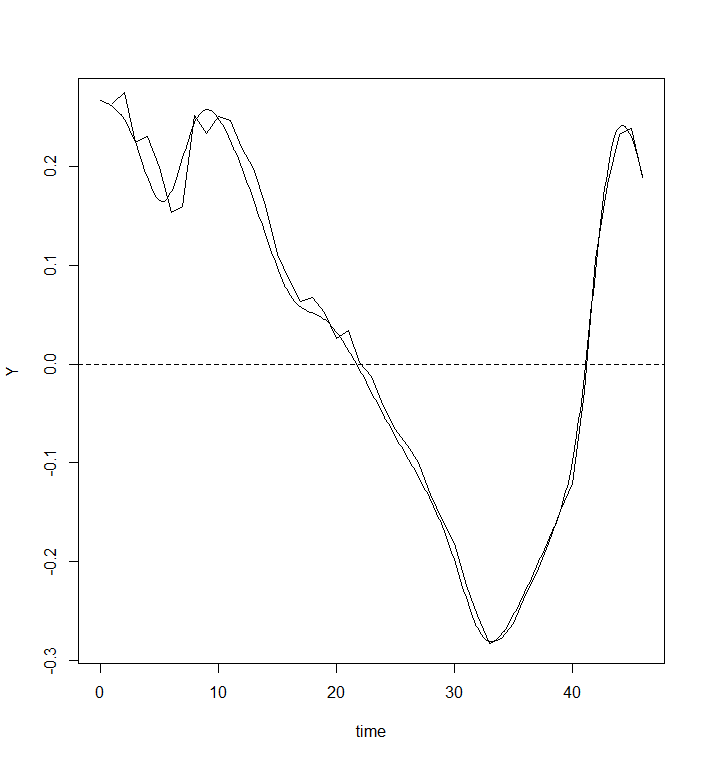
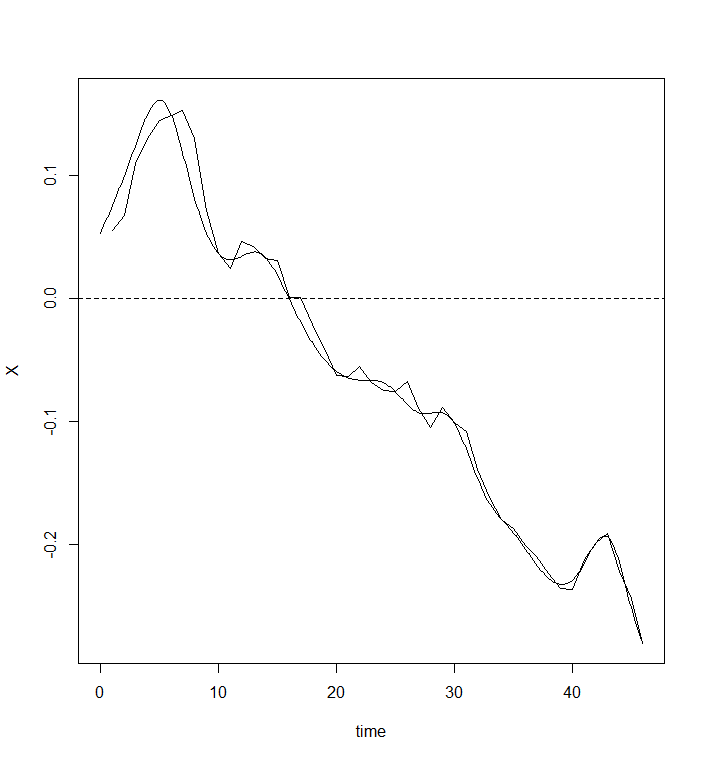


#### Test 3

Smoothing

numknots = 20, Degree = 4

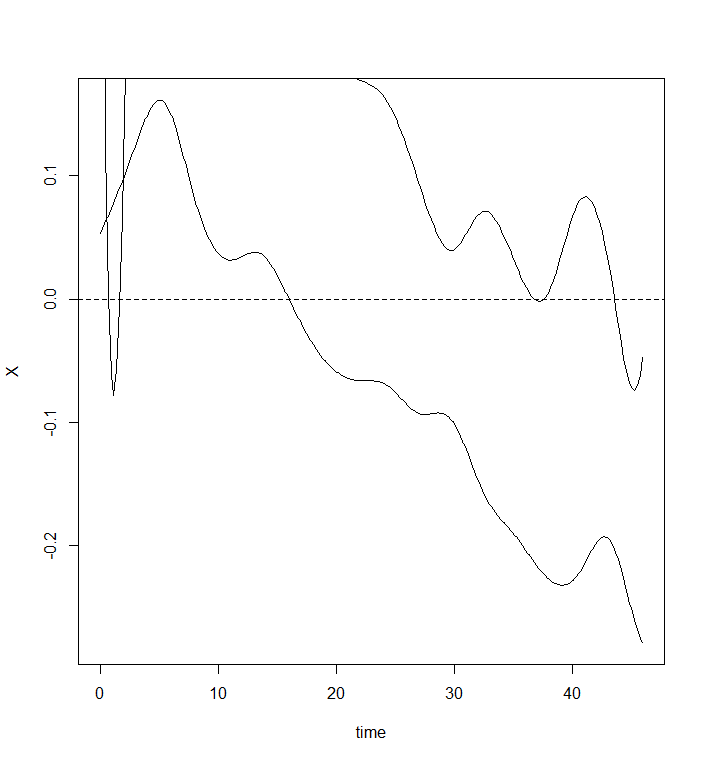
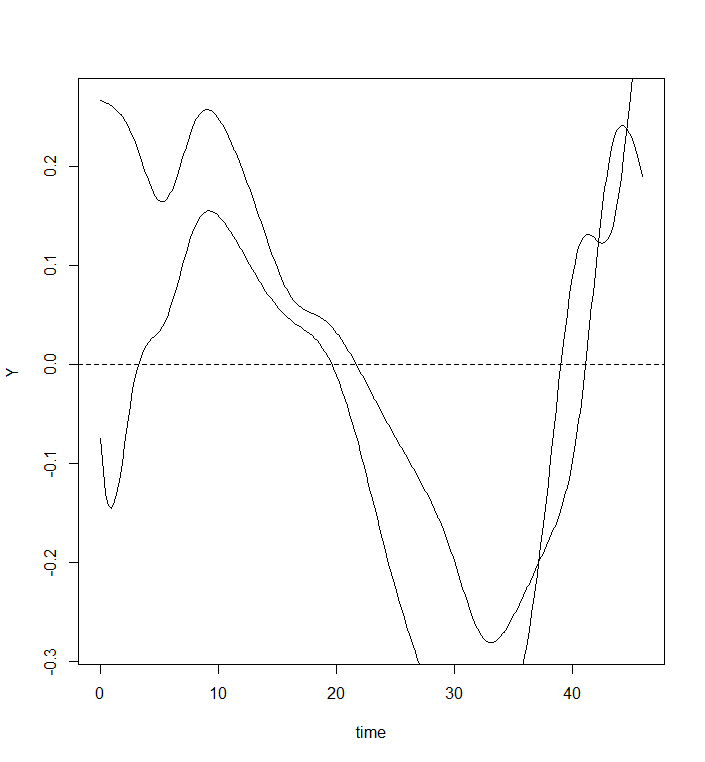
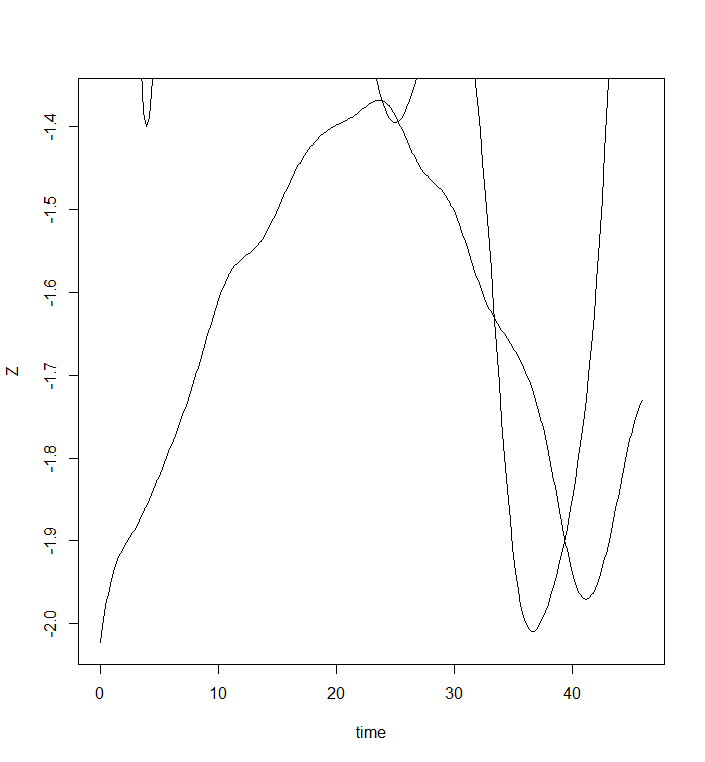
sample 1, Mean squared error 0.0348217



Reconstruction

numknots = 20, Degree = 4, nharm= 5

sample 1, Mean squared error 0.54343

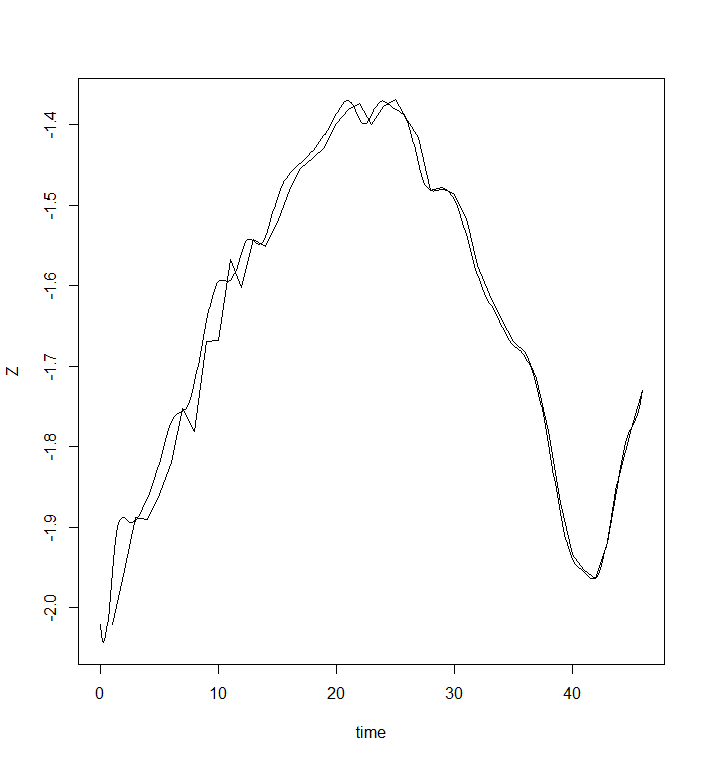
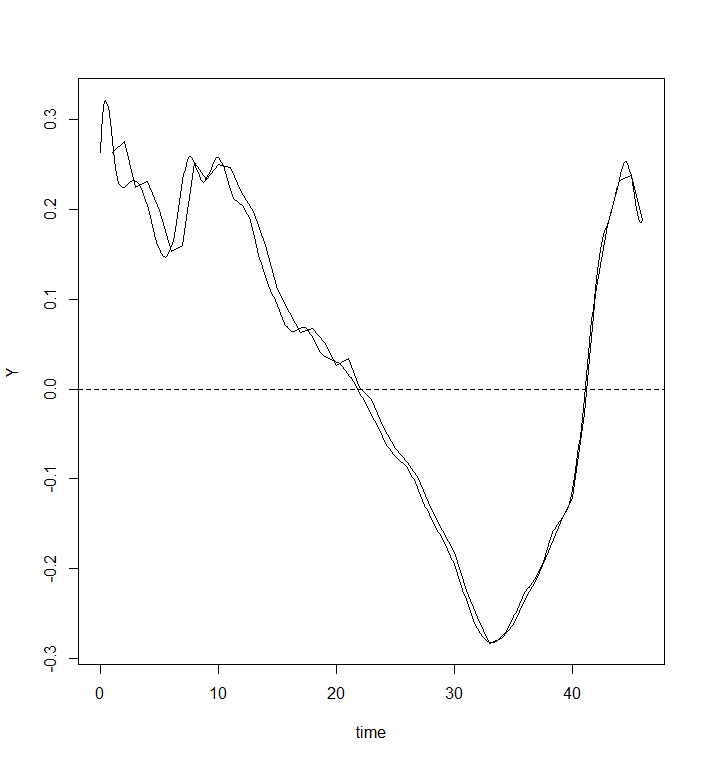
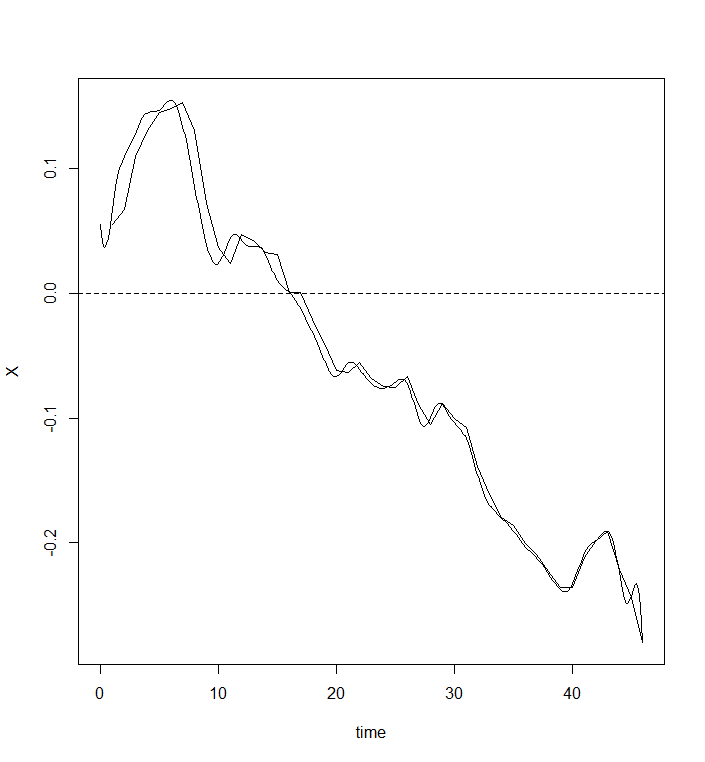
  

#### Test 4

Smoothing

numknots = 40, Degree = 4

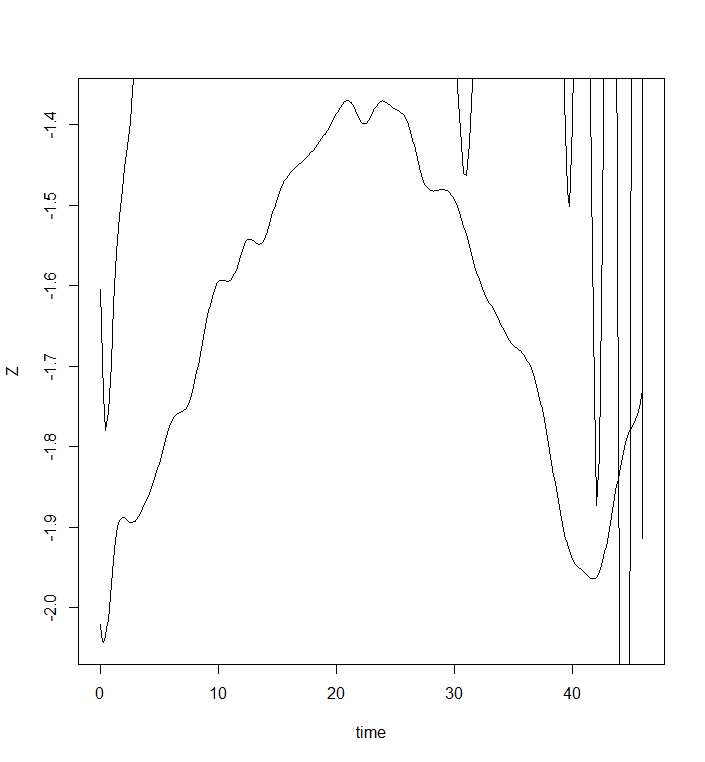
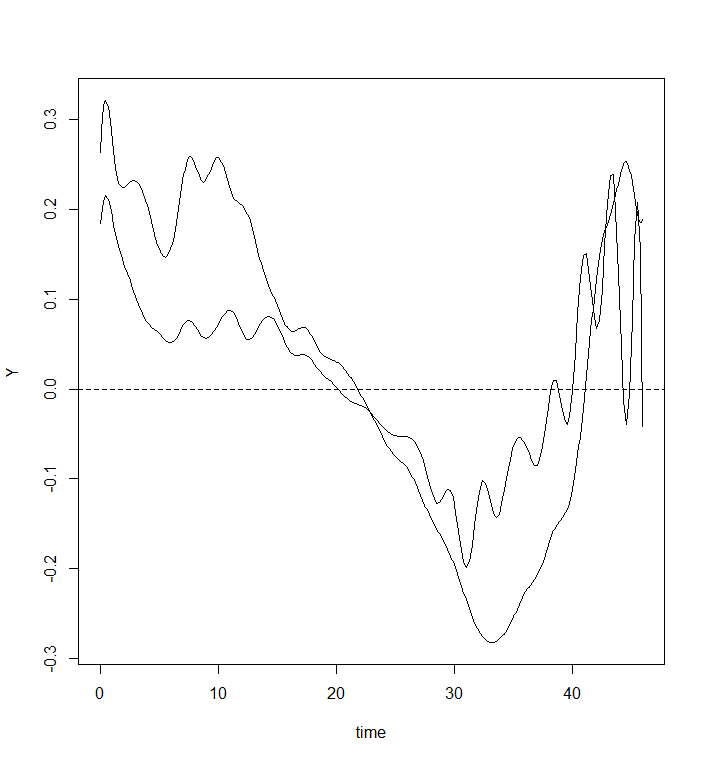
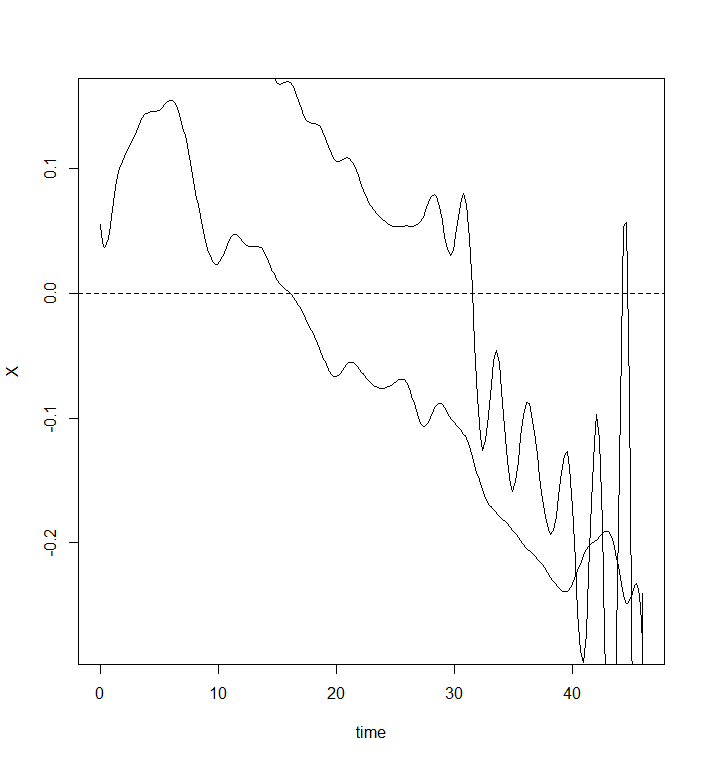
sample 1, Mean squared error 0.03142355



Reconstruction

numknots =40, Degree = 4, nharm= 5

sample 1, Mean squared error 0.5334677

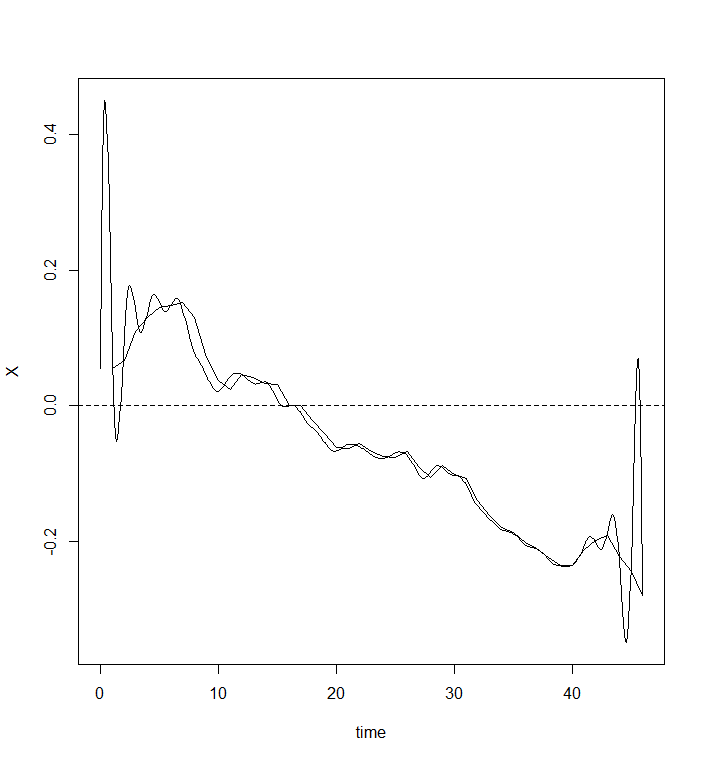
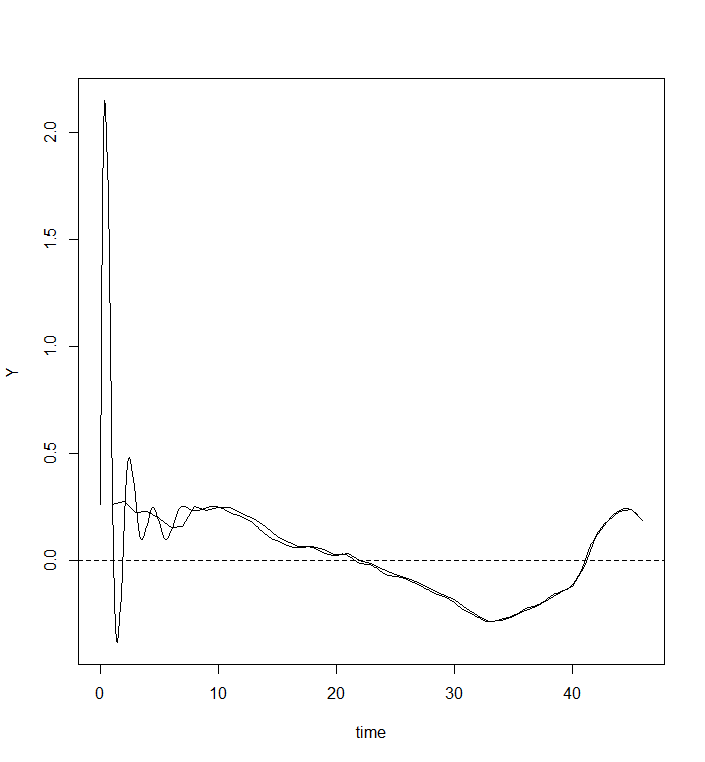
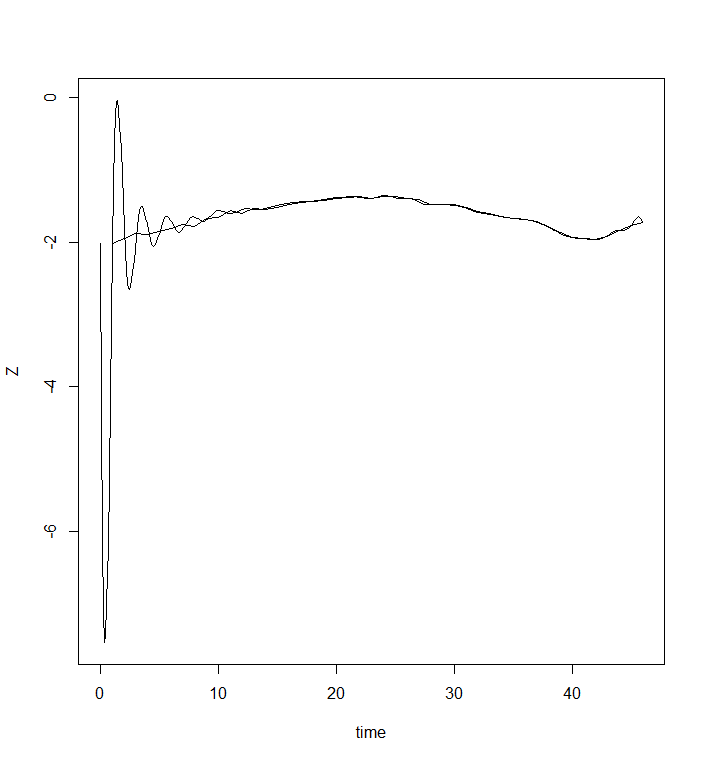


#### Test 5

Smoothing

numknots = 45, Degree = 4

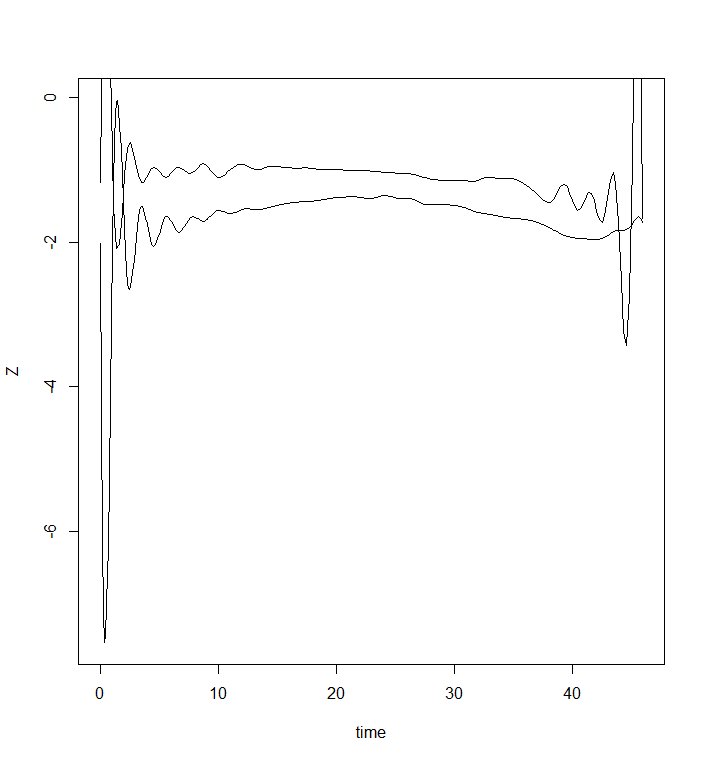
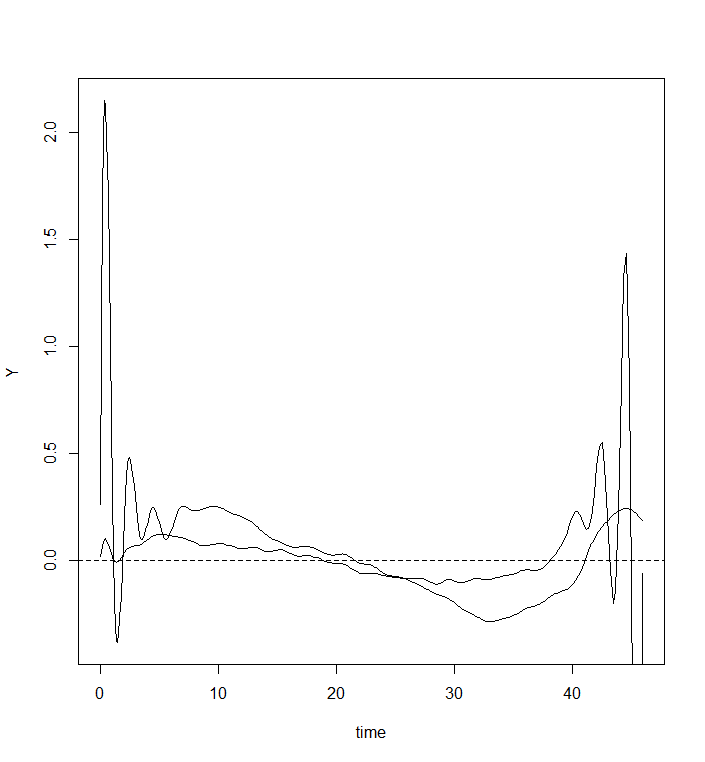
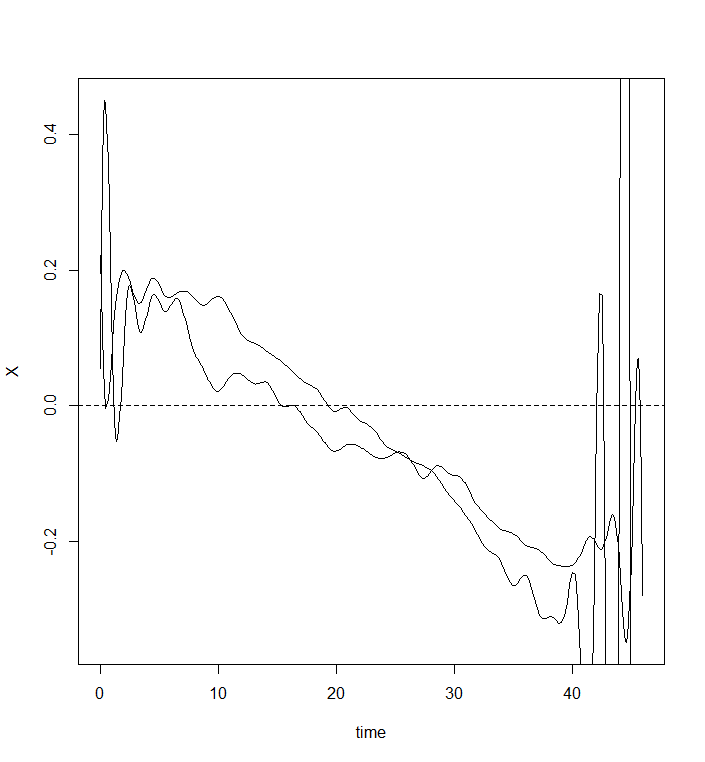
sample 1, Mean squared error 0.0460135

Reconstruction

numknots = 45, Degree = 4, nharm= 5

sample 1, Mean squared error 0.5641276

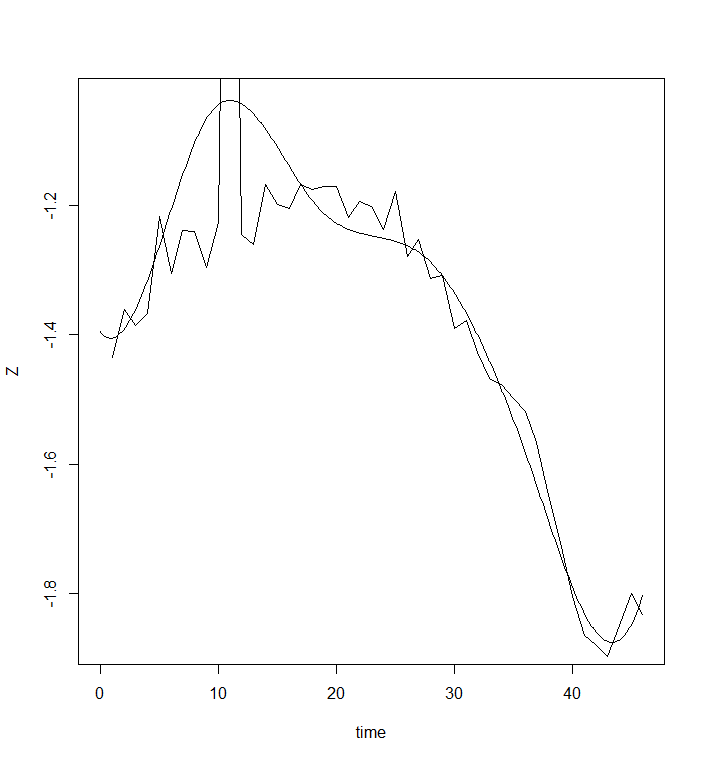
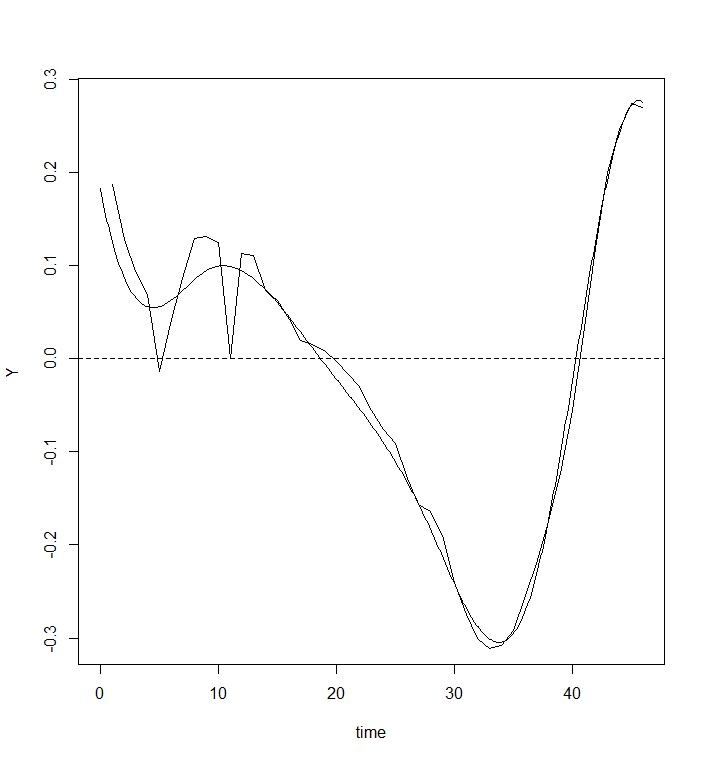
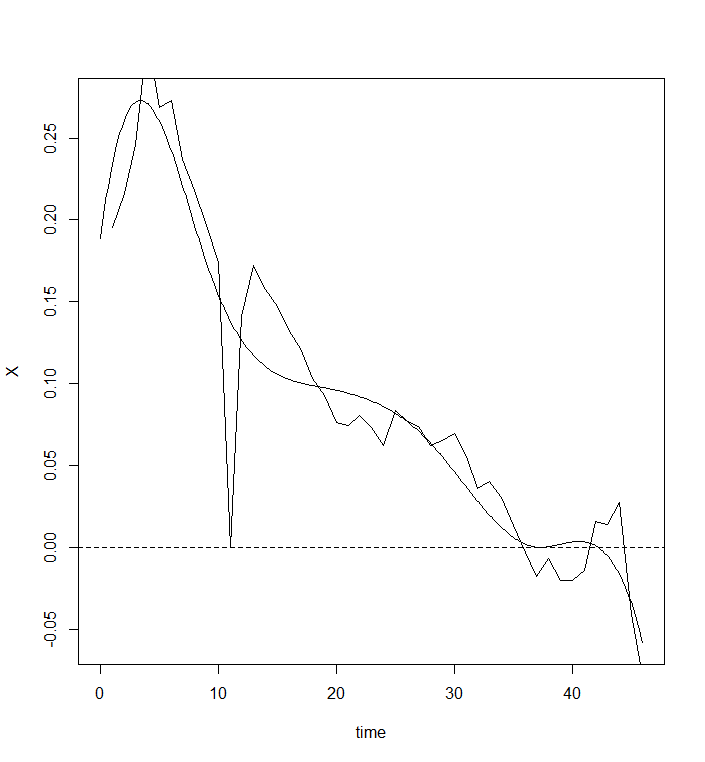


#### Test 6

Smoothing

numknots = 8, Degree = 4

sample 2, Mean squared error 0.09200566



Reconstruction

numknots = 8, Degree = 4, nharm= 5

sample 2, Mean squared error 0.2507706

