Stochastic Models - Tutorial 2

Moira Lampe

Gaussian model selection in the simplified Georgopalos setting

In this setting we measure n times the same cell but each time with a different angle of movement $u_i = 2\pi(i/n)$ for $i = 0 \dots (n-1)$

We decompose the regression function on a Fourier basis until size p with $2*p + 1 \leq n$

① Create a matrix X of size n times $2*p+1$. For $u_i = 2\pi(i/n)$, the coefficient $X_{i,j}$ is given as follows

- $X_{i+1,j} = 1$ if $j = 1$
- $X_{i+1,j} = \cos(k u_i)$ if $j = 2*k$ (same as p?)
- $X_{i+1,j} = \sin(k u_i)$ if $j = 2*k+1$

$$X = \begin{pmatrix} 1 & \cos(u_0) & \sin(u_0) & \cos 2u_0 & \dots & \sin(k u_0) \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \cos(u_i) & \sin(u_i) & \cos 2u_i & & \\ \vdots & \vdots & \vdots & \vdots & & \\ 1 & \cos(u_{n-1}) & \sin(u_{n-1}) & \cos 2u_{n-1} & \dots & \sin(k u_{n-1}) \end{pmatrix}$$

(i = row, j = column)

$k=1$	$k=1$	$k=2$	$\dots$
$j=2$	$j=3$	$j=4$	$j=2k+1$

The code in R:

```
#Question 1
#create matrix
#make a function

n = 5
p = 3

X = matrix(0, nrow = n, ncol = 2*p + 1) #this is an empty matrix
#number of rows is n because every row represents a time point for the same cell
print(X)


ui = 2*pi*(0:(n-1))/n
print(ui)

for (k in 0:p){
    } for (row in 1:nrow(X)){
        for (column in 1:ncol(X)){
          if (column = 1) #column here is the same as j
            X[row, column] = 1
         } else if (column = 2*k) #k is the same as p
            X[row, column] = cos(k*ui)
         } else if (column = 2*k + 1)
            X[row, column] = sin(k*ui)
```

=> My idea was to use a nested for loop, but somehow it did not work out.

```
print(X)

#something is wrong with the syntaxing so it is not working; can you help me figure out the mistake?

#I tried a different method which worked and continued with it, but I would still like to know what
#is wrong with the above code.

X = matrix(0, nrow = n, ncol = 2*p + 1)
X[,1] = rep(1,n) #fill the first columns with ones
for(k in 1:p){
  X[,2*k] = cos(k*ui)
  X[,2*k + 1] = sin(k*ui)
}

print(X)
```

```
> print(X)
     [,1]      [,2]       [,3]      [,4]       [,5]      [,6]       [,7]
[1,]    1  1.000000  0.0000000  1.000000  0.0000000  1.000000  0.0000000
[2,]    1  0.309017  0.9510565 -0.809017  0.5877853 -0.809017 -0.5877853
[3,]    1 -0.809017  0.5877853  0.309017 -0.9510565  0.309017  0.9510565
[4,]    1 -0.809017 -0.5877853  0.309017  0.9510565  0.309017 -0.9510565
[5,]    1  0.309017 -0.9510565 -0.809017 -0.5877853 -0.809017  0.5877853
```

② By computing the different scalar products, show that the columns of X are orthogonal but not of norm 1 & renormalize them.
↳ This gives you the matrix X'.

* **Scalar product**: a real number that is the product of the lengths of 2 vectors & the cosine of the angle between them.

$$\langle v_1, v_2 \rangle = v_{1x} \cdot v_{2x} + v_{1y} \cdot v_{2y}$$

$$= \|v_1\| \cdot \|v_2\| \cdot \cos(\theta)$$

$$\Rightarrow \|v_1\| = \sqrt{x_1^2 + y_1^2} = 1 \quad \text{for unitary vectors}$$

↳ so $\langle M, C \rangle = \cos(\theta)$

%*% = matrix multiplication

t() is used to calculate transpose of a matrix

Scaled Orthonormal/Orthogonal matrices

↳ an orthogonal matrix say Q, is a square matrix that satisfies

$$\underbrace{Q^*}_{\text{conjugate-transpose}} Q = I$$

How to prove that a matrix is orthogonal:

* A matrix is orthogonal if: $Q^T = Q^{-1}$

$$QQ^T = QQ^{-1}$$

$$QQ^T = I$$

As we can see from the output in the r code, the matrix is orthogonal, but not with norm 1. Therefore we have to renormalize the new matrix.

To normalize a vector: means to make v into a unit vector

$\vec{u} = \frac{\vec{v}}{\|\vec{v}\|}$ ($\vec{u}$: unit vector; $\vec{v}$: old vector; $\|\vec{v}\|$: own magnitude)

(if S, we would divide all the components of $\vec{v}$ by S)

$$\|\vec{v}\| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$$

So $\|\vec{u}\| = 1$ (if we did everything correct, this should be true)

R code:

```
#Question 2

#By computing the different scalar products, show that the columns of X are orthogonal
# but not norm 1 and renormalize them: this gives you the matrix X'

#scalar product

I = t(X) %*% X

#we can see that the matrix is orthogonal if it is equal to the identity matrix

print(I)

#we can see that we get an identity matrix, but we can see that it is not by norm 1
#therefore, we still have to normalize it

X_prime = matrix(0, nrow = n , ncol = 2*p + 1) #again we create an empty matrix
for (j in 1:ncol(X)){ #we go per column, because we take each column as a vector
    X_prime[,j] =  X[,j]/(sqrt(sum(X[,j]^2))) #we take each column as a vector
}
```

③ Give, for a given $d < p$, the projection estimator of the regression function composed of the first $2*d+1$ Fourier coefficients. Transform this into a function in r.

```r
#Question 3

d=seq(1,p-1,1) #to make sure d is smaller than p

proj_estim = function(d, X_prime, Y){ #this is our fourier estimator
  nrow = length(Y)
  four_coeff = 2*d + 1
  estimator = rep(0, nrow) #this is an empty vector with zeros
  for (i in 1:four_coeff){
    estimator = estimator + sum(X_prime[,i]*Y)*X_prime[,i]
  }

  return(estimator)

}
```

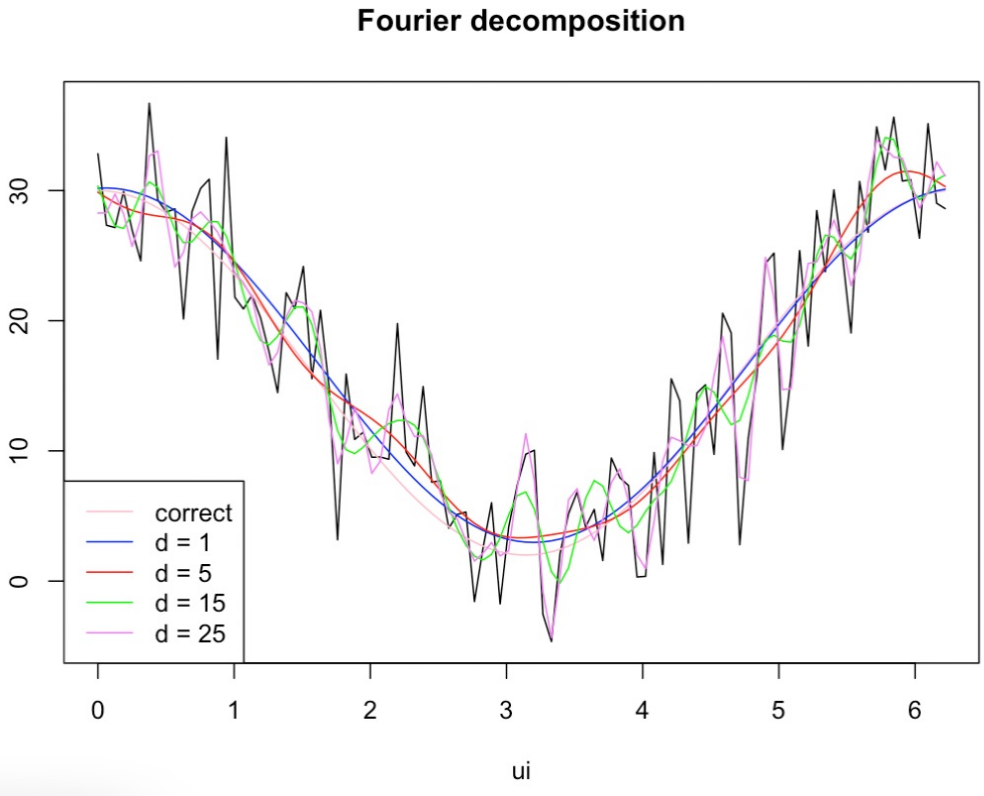
④ Simulate 2 different experiments:

$$Y_i = 16 + 14\cos(u_i) + 5\varepsilon_i \quad \& \quad Y_i = 10 * \exp(-(u_i-\pi)^2/0.2) + 1*\varepsilon_i$$

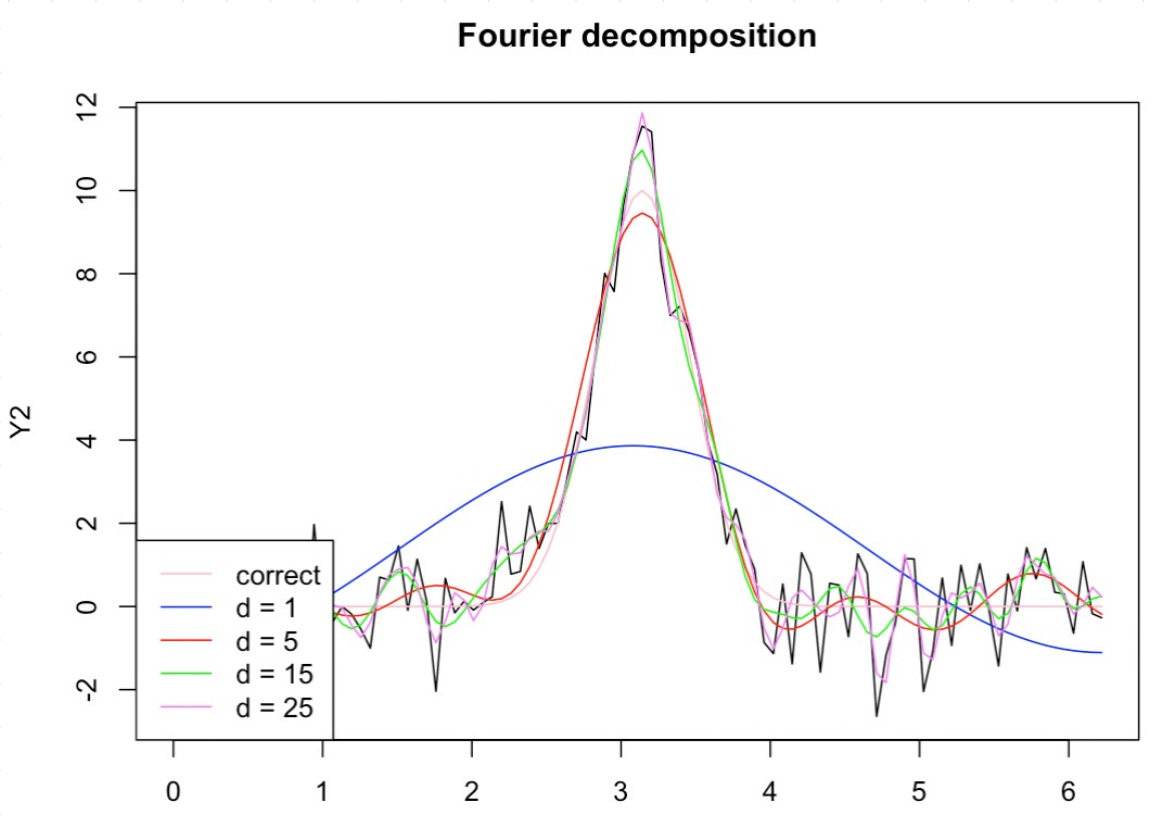
with ε_i iid $\mathcal{N}(0,1)$

Plot the data, the true function to estimate and 4 or 5 different projection estimators in each cases. Explain the problem of overfitting & the problem of taking a model of too low dimension.

```r
#Question 4

ei = rnorm(n)

Y1 = 16 + 14*cos(ui) + 5*ei

#plot the data
plot(ui, Y1, type='l',main='Fourier decomposition')
lines(ui,16+14*cos(ui),col='pink')
lines(ui, proj_estim(1, X_prime, Y1),col = 'blue')
lines(ui, proj_estim(5, X_prime, Y1),col = 'red')
lines(ui, proj_estim(15,X_prime, Y1),col = 'green')
lines(ui, proj_estim(25,X_prime, Y1),col = 'violet')
legend('bottomleft',c('correct','d = 1','d = 5','d = 15','d = 25'),lty=c(1, 1, 1, 1, 1),col=c('pink','blue','red','green','violet'))

Y2 = 10*exp(-(ui-pi)^2/(0.2)) + 1*ei

#plot the data
plot(ui, Y2, type='l',main='Fourier decomposition')
lines(ui,10*exp(-(ui-pi)^2/(0.2)),col='pink')
lines(ui, proj_estim(1, X_prime, Y2),col = 'blue')
lines(ui, proj_estim(5, X_prime, Y2),col = 'red')
lines(ui, proj_estim(15,X_prime, Y2),col = 'green')
lines(ui, proj_estim(25,X_prime, Y2),col = 'violet')
legend('bottomleft',c('correct','d = 1','d = 5','d = 15','d = 25'),lty=c(1, 1, 1, 1, 1),col=c('pink','blue','red','green','violet'))
```



Y_1



y_2

⑤ Make a function in R which, for a given p, computes the Mallows' Cp criterion for all the models ($d \leq p$) and gives the estimator which minimizes the criterion.

NB: the behavior would be similar for other models with log-likelihood & AIC criterion.

Mallows' Cp criterion: is used to assess the fit of a regression model that has been estimated using OLS.

$$\hat{d} = \underset{d}{\operatorname{argmin}} \underbrace{\| y - \Pi_v y \|^2}_{\text{least-square}} + \overbrace{2\sigma^2 d}^{\text{penalty}}$$

(we assume that sigma is known)

```r
#Question 5

LS = function(Y,X,d){ #LS = Least Squares
  return(sum((Y-projection(Y,X,d))^2)) #Sum of least squares
}

Mallows_Cp = function(Y, X_prime, sigma){
  CPs = rep(0,length(d))
  for (i in 1:length(d))
  CPs[i] = LS(Y,X,d[i]) + 2*d[i]*sigma^2

  return(CPs)
}
```

⑥ => I don't understand the question.