Third problem set

Jaume de Dios Pont

March 2, 2017

```
In [1]: require(reshape)
        require(ggplot2)
        require(MASS)
        library(censReg)

Loading required package: reshape
Loading required package: ggplot2
Loading required package: MASS
Loading required package: maxLik
Loading required package: miscTools

Please cite the 'maxLik' package as:
Henningsen, Arne and Toomet, Ott (2011). maxLik: A package for maximum likelihood estimation in R. Compu

If you have questions, suggestions, or comments regarding the 'maxLik' package, please use a forum or '
https://r-forge.r-project.org/projects/maxlik/
```

With the inestimable help of Alexander Chan

1 Question 1

(25pt + 10pt) In this question, we use the file `brader.csv` which contains data from Brader, Valentino and Suhay (2008). The file includes the following variables for $n = 265$ observations:

- the outcome of interest – a four-point scale in response to Do you think the number of immigrants from foreign countries should be increased or decreased?
- tone of the story treatment (positive or negative)
- ethnicity of the featured immigrant treatment (Mexican or Russian)
- respondents' age
- respondents' income

Consider the following ordered logit model for an ordered outcome variable with four levels:

$$\Pr(Y_i \leq j \mid X_i) \;=\; \frac{\exp(\psi_j - X_i^\top \beta)}{1 + \exp(\psi_j - X_i^\top \beta)}$$

for $j = 1, 2, 3, 4$ and $i = 1, ..., n$ where $\psi_4 = \infty$ and $X_i = [\texttt{tone}_i\ \texttt{eth}_i\ \texttt{ppage}_i\ \texttt{ppincimp}_i]^\top$ (i.e. no intercept).

a) (5pt) Write down the likelihood function.

To simplify the notation, the indices* i, j, k *are used consistently,* i *to iterate over the number of observations,*j *to iterate over the number of outcomes, and* k * to iterate over the predictors.

The log-likelihood can be easily seen to be:

$$l = \prod_{i=1}^{n} \frac{\exp(\psi_{Y_i} - X_i^\top \beta)}{1 + \exp(\psi_{Y_i} - X_i^\top \beta)} - \frac{\exp(\psi_{Y_i - 1} - X_i^\top \beta)}{1 + \exp(\psi_{Y_i - 1} - X_i^\top \beta)}$$

To simplify, from here on ϕ will be the inverse link function and ϕ' its derivative. This allows us to write the above expression in a more Matricial form as:

$$\prod_{i=1}^{n}\sum_{j=1}^{4}\tilde{M}_{i,j}\phi\left(\psi_j-\sum_{k=1}^{m}X_{ik}\beta_k\right)$$

where

$$\tilde{M}=MK=M\begin{pmatrix}1&0&0&0\\-1&1&0&0\\0&-1&1&0\\0&0&-1&1\end{pmatrix}$$

And M is the OneHot matrix for the Y_i, i.e. a matrix that has n rows, and each row is the row vector with zeros everywhere but at the position j, where the the observed Y is Y_j.
Below, there is a function that computes $\tilde{M}$ in R:

```
In [2]: compute_M = function (Y,m){

            #This is just a One Hot encoder:
            n = length(Y)
            M1 = t(matrix(rep(c(1:m),n),m))
            M2 = matrix(rep(c(Y),m),n)
            M = (M1==M2)+0

            #Create the matrix K
            K = diag(m+1)
            K = -K[1:m,]+K[2:(m+1),]
            K = K[1:m,1:m+1]

            #Return the product
            M%*%K
        }
```

In practice, however, we are interested in the log-likelihood, which is:

$$L=\sum_{i=1}^{n}\log\sum_{j=1}^{4}\tilde{M}_{i,j}\phi\left(\psi_j-\sum_{k=1}^{m}X_{ik}\beta_k\right)$$

The notation is further simplified if we pre-comute the $\phi\left(\psi_j-\sum_{k=1}^{m}X_{ik}\beta_k\right)$ as ϕ'_{ij} (and the same for ϕ'_{ij}.
Below there is a wrapper in `R` that automatically defines these quantities for our function, so that we can use them :

```
In [3]: wrapper = function(expression){
            function(X,Y,beta,psi,
                     phi = function(x){exp(x)/(1+exp(x))},
                     dphi = function(x){exp(x)/(1+exp(x))**2})
            {
                psi = append(psi,10); #add the 'infinity'
                m = length(psi); #number of outcomes
                n = length(Y) #number of observatons
                M = compute_M(Y,m);

                # transform beta_j to beta_jk (as in part d)
                # whenever necessary
                if(length(beta) == ncol(X))
                    beta2 = matrix(rep(beta,m),ncol=m,byrow=0)
                else beta2 = beta

                #Compute the linear combination inside 'psi'
                linear =  - (X%*%beta2)
                origin =  matrix(rep(psi,n) ,nrow = n, byrow = 1)
                l = origin +  linear

                 phix = phi(l);    phix[,m]=1; #impose the 'infinity'
                dphix = phi(l);  dphix[,m]=0; #impose the 'infinity'

                expression(X,Y,beta,psi,M,phix,dphix);
            }
        }
```

Now we are ready to define the log-likelihood function immediately using the abstraction wrapper:

```
In [4]: log_likelihood = wrapper(
            function(X,Y,beta,psi,M,phi,dphi){
                sum(log(rowSums(M*phi)))
            }
        )
```

The wrapper function is just an abstaction (see it at the end of the file, in the annex), that computes the variables `M`, `phi`= $\phi\left(\psi_j - \sum_{k=1}^m X_{ik}\beta_k\right)$, `dphi`= $\phi'\left(\psi_j - \sum_{k=1}^m X_{ik}\beta_k\right)$. We use it because everything is easier in those variables. When a function goes trough the "wrapper", its arguments change to become (X,Y,beta,psi), and the wrapper computes M, phi and dphi.

b) (10pt) Derive the score functions for β and ψ_j.

To simplify the computations, let $\phi_{ij} = \phi(\psi_j - X_i^\top\beta)$, and $\phi'_{ij} = \phi'(\psi_j - X_i^\top\beta)$. Then we will have:

$$L = \sum_{i=1}^{n}\log\sum_{j=1}^{4}\tilde{M}_{i,j}\phi_{i,j}$$

Since deriveatives and sums commute freely, we can compute the score easily:

$$\frac{\partial L}{\partial \beta_k} = -\sum_{i=1}^{n}\left(\sum_{j=0}^{4}\tilde{M}_{i,j}\phi_{i,j}\right)^{-1}\left(\sum_{j=0}^{4}\tilde{M}_{i,j}\phi'_{i,j}\right)X_{ik}$$

in `R` that is:

```
In [5]: beta_score =  wrapper(
            function(X,Y,beta,psi,M,phi,dphi){
                -(rowSums(M*dphi)/rowSums(M*phi))%*%X
            }
        )
```

For the ψ, the result is equally immediate:

$$\frac{\partial L}{\partial \psi_j} = \sum_{i=1}^{n}\left(\sum_{j=0}^{4}\tilde{M}_{i,j}\phi_{i,j}\right)^{-1}\tilde{M}_{i,j}\phi'_{i,j}$$

which in `R` can be written as:

```
In [6]: psi_score =  wrapper(
            function(X,Y,beta,psi,M,phi,dphi){
                r = (rowSums(M*phi)**-1%*%(M*dphi))
                r[1:(length(r)-1)]
            }
        )
```

(10pt) Using (a) and (b), calculate the maximum likelihood estimates of β and ψ_j and their standard errors via the `optim` function in R. Confirm your results by comparing them to outputs from the `polr` function in the `MASS` package.

Load the data:

```
In [7]: brader = read.csv('Data/brader.csv')
```

Prepare a Handler function for `optim`

```
In [8]: likelihood_handler = function (x,data){
            X = data[[1]]
            Y = data[[2]]
            o = ncol(X)
            beta = x[1:o]
            psi = x[(o+1):length(x)]
            l = length(psi)
            diff_psi = psi[2:l]-psi[1:(l-1)]
            if(min(diff_psi)<=0)
                return (1E5)


            return(-log_likelihood(X,Y,beta,psi))
        }
```

```
In [9]: gradient_handler = function(x,data){
            X = data[[1]]
            Y = data[[2]]
            o = ncol(X)
            beta = x[1:o]
            psi = x[(o+1):length(x)]
            l = length(psi)
            diff_psi = psi[2:l]-psi[1:(l-1)]
            if(min(diff_psi)<=0)
                return(x*0);
            sbeta = (-beta_score(X,Y,beta,psi))
            spsi  = (-psi_score(X,Y,beta,psi))
            return(append(sbeta,spsi))
        }
```

Then we prepare the data as our function needs it

```
In [10]: Y = brader$immigr
         X = data.matrix(brader)[,2:5]
         data = list(X,Y)
         beta = c(0.5, 0, 0, 0)
         psi = c(-1, 0, 1)
         betapsi = append(beta,psi)
```

And we run the optimization

```
In [11]: r = optim(betapsi,likelihood_handler, gr = gradient_handler, data = data,
                   hessian = 1, control = (reltol = 1E-12))
         r$par
         r$value
```

0.75057472585271

0.182514633025783

0.00954246157357417

0.00453801085115179

-1.64987523554466

0.153712692971585

1.37459144576628
    325.94791194349
    Standard Errors:

```
In [12]: diag(solve(r$hessian))**.5
```

0.162040401841744

0.160164676752734

0.00530840672161387

0.0230051608556128

0.430797558900244

0.381474494066442

0.370321674525004

```
In [13]: library(MASS)

         plr <- polr( factor(immigr) ~ tone + eth + ppage + ppincimp,
                            data = brader, method = "logistic", Hess = 1)
         summary(plr)
         -log_likelihood(X,Y,plr$coefficients,plr$z)
```

```
Call:
polr(formula = factor(immigr) ~ tone + eth + ppage + ppincimp,
    data = brader, Hess = 1, method = "logistic")

Coefficients:
            Value Std. Error t value
tone     0.749446   0.230241  3.2550
eth      0.166225   0.227063  0.7321
ppage    0.009345   0.007131  1.3105
ppincimp 0.004149   0.029511  0.1406

Intercepts:
    Value   Std. Error t value
1|2 -1.6671  0.5664    -2.9434
2|3  0.1318  0.5401     0.2441
3|4  1.3535  0.5466     2.4761

Residual Deviance: 651.8892
AIC: 665.8892
```

325.944618379387

** d. (10pt) Bonus question. ** The standard ordered logit model is sometimes called the proportional odds model because it assumes the effect of X_i is constant across levels on the odds ratio scale. One approach to relax this assumption is to allow the coefficients to vary across levels, i.e.,

$$\Pr(Y_i \leq j \mid X_i) \;=\; \frac{\exp(\psi_j - X_i^\top \beta_j)}{1+\exp(\psi_j - X_i^\top \beta_j)}$$

for $j = 1, 2, 3, 4$ and $i = 1, ..., n$ where $\beta_4 = 0$ (i.e. the fourth group is a reference group), and $\psi_4 = \infty$. For this model, derive the likelihood and score functions, and use `optim` to obtain the maximum likelihood estimates of β_j and ψ_j and their standard errors for the `brader` data.

A direct modification of the above exercise, substituting β_k for $\beta_{k,j}$ works: We will have:

$$L = \sum_{i=1}^{n} \log \sum_{j=1}^{4} \tilde{M}_{i,j}\phi\left(\psi_j - \sum_{k=1}^{m} X_{ik}\beta_{kj}\right)$$

therefore, since the formula is essentialy the same (and thanks to `R` not really caring much about the shape of the objects – which is nice!), the same `log_likelihood` coded above still works!

Since deriveatives and sums commute freely, we can compute the score easily:

$$\frac{\partial L}{\partial \beta_{kj}} = -\sum_{i=1}^{n}\left(\sum_{j=0}^{4} \tilde{M}_{i,j}\phi_{i,j}\right)^{-1} \tilde{M}_{i,j}\phi'_{i,j}X_{ik}$$

in this case we must modify the function, to create a new function:

```
In [14]: extra_beta_score = wrapper(
             function(X,Y,beta,psi,M,phi,dphi){
                 #this corresponds to a matrix that has the same elements in every
                 #row, and are the (sum **)^{-1} in the formula above
                 M1 = matrix(rep(rowSums(M*phi),ncol(M)),ncol=ncol(M),byrow=0)
                 -t(M*dphi/M1)%*%X
             }
         )
```

For the ψ, the result is equally immediate:

$$\frac{\partial L}{\partial \psi_j} = \sum_{i=1}^{n}\left(\sum_{j=0}^{4} \tilde{M}_{i,j}\phi_{i,j}\right)^{-1} \tilde{M}_{i,j}\phi'_{i,j}$$

As for the likelihood, the same function does the job

Now the code: Create the initial conditions from the optimization in the previous case:

```
In [15]: beta0 = rep(plr$coefficients , 4)
         psi0 = plr$zeta
         betapsi0 = c(beta0,psi0)
```

create a new likelihood handler

```
In [16]: extra_likelihood_handler = function (x,data){
             X = data[[1]]
             Y = data[[2]]
             o = ncol(X)
             beta = matrix(x[1:o*4],ncol=4, byrow = 0)
             psi = x[(o*4+1):length(x)]
             l = length(psi)
```

```
    diff_psi = psi[2:l]-psi[1:(l-1)]
    if(min(diff_psi)<=0)
        return (1E5)


    return(-log_likelihood(X,Y,beta,psi))
}

r = optim(betapsi0,extra_likelihood_handler,  data = data,
          hessian = 1, control = (reltol = 1E-12))

matrix(r$par[1:16],byrow=0,ncol=4)
r$par[17:19]
```

```
    0.6943061    0.63885579    0.72700705    0.835971250
    -0.3021024   0.14222286    0.10165316    -0.060505633
    -0.1915581   -0.02427104   0.93475166    -0.377267011
    0.7014210    -0.03038976   0.01123405    0.007652073
-1.66549494997964
0.152774935941614
1.36057070649973
```

2 Question 2

(30 pt.) Understanding the basics of maximum likelihood makes it easier to learn a wide range of models. Here you will learn about a model not covered in class. The {*tobit model*} *is often used to model a censored outcome variable, where we only observe values of the outcome above a certain threshold τ while values below that threshold take on a single value, say 0. A classic example is labor market participation: you only observe a worker's wage if their potential earnings is high enough to be over minimium wage or otherwise employable.*

Consider the case where $\tau = 0$. In this case, the model can be written as follows.

$$Y_i \quad = \quad \begin{cases} Y_i^* & \text{if} \quad Y_i^* > 0 \\ 0 & \text{if} \quad Y_i^* \leq 0 \end{cases} \qquad \text{where} \quad Y_i^* = X_i^\top \beta + \epsilon_i,$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$. Note that Y_i is the observed outcome variable and Y_i^* is the underlying latent variable, which is not (always) observable.

a.*(10pt) Derive the likelihood and log-likelihood functions of the model for a simple random sample of size N.*

We will (for a moment) forget that log-likelihood is in fact terribly ill posed for variables that are a mixture of a continuous and a discrete variable, and proceed as if we didn't know that:

We have that:

$$P(Y_k^* > 0) = P(Y_k > 0) = \Phi\left(\frac{x^T\beta}{\sigma}\right)$$

Moreover, when $Y_k > 0$

$$\rho(Y_k^* | Y_k^* > 0) = \rho(Y_k | Y_k > 0) = (P(Y_k > 0))^{-1}\phi\left(\frac{x^T\beta - Y_k}{\sigma}\right)$$

For the other case we have:

$$P(Y_k = 0) = P(Y_k^* < 0) = 1 - \Phi\left(\frac{x^T\beta}{\sigma}\right)$$

Therefore the likelihood is

$$L(Y_i) = \begin{cases} 1-\Phi\left(\frac{X_i^T\beta}{\sigma}\right), & Y=0 \\ \phi\left(\frac{X_i^T\beta - Y_i}{\sigma}\right), & Y>0 \end{cases}$$

We will pretend what we did above is right for the sake of the exercise, but a careful second look shows that we have brutally mixed probabilities and probability densities in an almost non-sensical way. Moreover note that the density is not even continuous, so any attempt to apply any of the theorems done at class is not justified (at least in the way we did them at class).

```
In [17]: tobit_log_likelihood = function(X,Y,beta,sigma){
             x = X%*%beta/sigma
             PN = 1-pnorm(x)
             PP =  dnorm(x)
             PF = PN;
             PF[Y>0] = PP
             sum(log(PF))
         }
```

b. *(10pt) A natural quantity of interest for this model is the expected value of the outcome variable conditional on a set of predictor levels, i.e., $E(Y_i \mid X_i = x)$. Prove that the exact expression for this quantity is the following.*

$$E(Y_i \mid X_i = x) = \Phi\left(\frac{x^\top\beta}{\sigma}\right)x^\top\beta + \sigma\phi\left(\frac{x^\top\beta}{\sigma}\right),$$

where $\phi(\cdot)$ and $\Phi(\cdot)$ denote the PDF and CDF of a standard normal random variable, respectively. \ {**Hint:**} **You may find the following property of the standard normal PDF helpful: $\partial\phi(z)/\partial z = -z\cdot\phi(z)$.**

The expectation can be computed as the following integral:

$$E(Y_i) = \int_0^\infty \frac{d}{dy}\Phi\left(\frac{X_i^T\beta - y}{\sigma}\right)ydy$$

Where we neglected the case where $Y_i = 0$

In order to compute it, we will perform a change of variables, with $z = \frac{X_i^T\beta - y}{\sigma}$

$$\begin{aligned} E(Y_i) = \int_{-\infty}^{\frac{X_i^T\beta}{\sigma}} \frac{d}{dz}\Phi(z)(X_i^T\beta - z\sigma)dy =& \Phi\left(\frac{X_i^T\beta}{\sigma}\right)X_i^T\beta + \sigma\int_{-\infty}^{\frac{X_i^T\beta}{\sigma}} -z\frac{d}{dz}\Phi(z)dz = \\ =& \Phi\left(\frac{X_i^T\beta}{\sigma}\right)X_i^T\beta + \sigma\int_{-\infty}^{\frac{X_i^T\beta}{\sigma}} \frac{d}{dz}\phi(z)dz = \\ =& \Phi\left(\frac{X_i^T\beta}{\sigma}\right)X_i^T\beta + \sigma\phi\left(\frac{X_i^T\beta}{\sigma}\right) \end{aligned}$$

c. (10pt) Nielsen (2013, {*ISQ***})** *investigates the impact of human rights violations on the flows of foreign aid from donor states to recipient states. The file* {`nielsenaid.csv`} `contains a small subset of the original dataset and consists of the following variables measured for each donor-recipient dyad:`

- `lneconaid --- log of economic aid commitments in U.S. dollars in 2000`
- `l.physint --- physical integrity violations index (e.g. toture) in 1999`
- `l.polity2 --- Polity IV democracy score in 1999`
- `l.lneconaid --- one-year lagged value of lneconaid`

- l.lnrgdp --- log GDP per capita in 1999 candidate
- l.lnpop -- log population in 1999
- colony --- indicator of former colony
- recep --- recipient country name degree
- donor --- donor country name

Fit the above tobit model, where

$$\begin{aligned} Y_i &= \texttt{lneconaid}, \\ X_i &= [\texttt{l.physint}, \texttt{l.polity2}, \texttt{l.lneconaid}, \texttt{l.lnrgdp}, \texttt{l.lnpop}, \texttt{colony}]^\top, \end{aligned}$$

and estimate $E[Y_i \mid X_i = x]$ and its asymptotic 95% confidence interval, where x equals the medians for all predictors except {colony}, and the mode for {colony}. (For this question, you may ignore possible error correlation across observations, such as clustering. You may use a canned procedure to obtain the MLE of β and σ as well as their sampling variance matrix; but for **{extra credit}, calculate those via {optim}** on your own.)

```
In [18]: df = read.csv('Data/nielsenaid.csv')
         model = censReg( lneconaid ~ l.physint+l.polity2+l.lneconaid+l.lnrgdp+l.lnpop+colony,
                          left = 0, right = Inf, data = df)
         summary(model)
```

```
Call:
censReg(formula = lneconaid ~ l.physint + l.polity2 + l.lneconaid +
    l.lnrgdp + l.lnpop + colony, left = 0, right = Inf, data = df)

Observations:
         Total  Left-censored     Uncensored Right-censored
          3129           1872           1257              0

Coefficients:
            Estimate Std. error t value  Pr(> t)
(Intercept)  0.62113    0.75383   0.824   0.4100
l.physint    0.07140    0.03844   1.858   0.0632 .
l.polity2    0.01395    0.01209   1.154   0.2486
l.lneconaid  1.21494    0.02580  47.095  < 2e-16 ***
l.lnrgdp    -0.52425    0.07526  -6.966 3.27e-12 ***
l.lnpop      0.11769    0.05297   2.222   0.0263 *
colony      -0.12036    0.36842  -0.327   0.7439
logSigma     1.08275    0.02182  49.628  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Newton-Raphson maximisation, 6 iterations
Return code 1: gradient close to zero
Log-likelihood: -3879.845 on 8 Df
```

```
In [19]: coef =
         c(1,mean(df$l.physint),
         mean(df$l.polity2),
         mean(df$l.lneconaid),
         mean(df$l.lnrgdp),
         mean(df$l.lnpop),
         (mean(df$colony)<.5)+0) #Mode of colony
```

Now we compute the value of $Y*$ and its standard deviation. The $[1, 7]$ is because the last variable is logSigma, which we do not need to use. The idea is that we first compute the covariance matrix of $\hat{\beta}$, which is the inverse of the hessian, and then the variance of $\langle\hat{\beta}, x\rangle$ is precisely $ x ^T COV() x$

```
In [20]: Ystar = model$estimate[1:7]%*%coef
         Ystarsd= (-coef%*%solve(model$hessian)[1:7,1:7]%*%coef)**.5
         Ystar
         Ystarsd
```

```
-0.4422193
0.3715102
```

From this we know that the lower bound of a 95% confidence interval must be 0. Therefore, the upper bound can be compyted with the quantile of the normal (assimptotically).

```
In [21]: Ystar+qnorm(.95)*Ystarsd
```

```
0.1688606
```

```
In [23]: tobit_likelihood_handler = (function(X,Y){
             function(betasigma){
                 beta = betasigma[1:7]
                 sigma = betasigma[8]
                 -tobit_log_likelihood(X,Y,beta,sigma)
             }
         }
         )(X,Y)
         start = c(0.6,0.07,0.01,1.2,-0.5,0.1,0.12,2.7)
         r = optim(start,tobit_likelihood_handler,
                   hessian = 1, control = (reltol = 1E-12))
         r$par
```

```
    Error in X %*% beta: requires numeric/complex matrix/vector arguments
Traceback:


    1. optim(start, tobit_likelihood_handler, hessian = 1, control = (reltol = 1e-12))

    2. (function (par)
  . fn(par, ...))(c(0.6, 0.07, 0.01, 1.2, -0.5, 0.1, 0.12, 2.7))

    3. fn(par, ...)

    4. tobit_log_likelihood(X, Y, beta, sigma)   # at line 5 of file <text>
```

3 Question 3

(20 pt.) Consider a supervised learning problem in which you sample Y_i and X_i from $p(X, Y)$. You assume an additive noise model such that: $Y_i = f(X_i) + \epsilon_i$, and $E[\epsilon_i|X_i] = 0$. We wish to come up with $\hat{f}(X)$ that is the best approximation to $f(X)$ in some sense.

a (8pt.) First choose your loss function, $L(Y_i, \hat{f}(X_i))$ to be squared loss. Show that the minimizer of this loss is found by choosing $\hat{f}(X_i) = E[Y_i|X_i]$. (It suffices to show this pointwise, i.e. conditional on a particular choice of X).

Pointwise we have

$$\begin{aligned} E(L[Y_i, \hat{f}(X_I))] =& E[(Y_i - f(X_i))^2|X_I] = \\ =& E[(Y_i - \bar{Y}_i + \bar{Y}_i - \hat{f}(X_i))^2|X_I] = \\ =& E[(Y_i - \bar{Y}_i)^2 + (\bar{Y}_i - \hat{f}(X_i))^2|X_I] = \\ =& E[(Y_i - \bar{Y}_i)^2|X_I] + E[(\bar{Y}_i - \hat{f}(X_i))^2] \end{aligned}$$

Of the two last summands, the first does not deppend on $\hat{f}$ and the seond is clearly minimized when $\hat{f}(X_i) = E[Y_i|X_i]$, as we wanted to prove. Note that we have used the property that for two intependent random variables $E[(X-Y)^2] = E[(\bar{X}-X)^2) + E[(\bar{X}-Y)^2]$

b. (10pt.) Show that the expected generalization error, $\mathcal{R} = E[(Y_i - \hat{f}(X_i))^2]$, can be decomposed into an irreducible error, a bias term, and a variance term. Furthermore, describe in plain English the meaning of variance and bias in this context. (Again, it suffices to do this pointwise, conditionally on X).

Using the previous result, we have:

$$\begin{aligned} E[(Y_i - f(X_i))^2] =& E[(Y_i - \bar{Y}_i)^2] + E[(\bar{Y}_i - \hat{f}(X_i))^2] = \\ =& E[(Y_i - \bar{Y}_i)^2] + E[(\bar{Y}_i - E[\hat{f}(X_i)] + E[\hat{f}(X_i)] - \hat{f}(X_i))^2] = \\ =& E[(Y_i - \bar{Y}_i)^2] + E[(\bar{Y}_i - E[\hat{f}(X_i)])^2 + (E[\hat{f}(X_i)] - \hat{f}(X_i))^2] = \\ =& E[(Y_i - \bar{Y}_i)^2] + E[(\bar{Y}_i - E[\hat{f}(X_i)])^2] + E[(E[\hat{f}(X_i)] - \hat{f}(X_i))^2] \end{aligned}$$

Where we have used (again) the same property of the expectations of squares of differences. THe first term is the irreducible error, the second therm is the bias term and the third term the variance term.

The variance term is the ineherent variance on the predictor function due to the fact that it is produced from random variables, the bias term is the perdiction error inherent to the learning algorithm, due to the fact that the real function cannot be totally understood by the predictor in some cases. It cannot be solved even by adding more data.

c (2pt.) In general how does a learning algorithm's flexibility relate to the bias and variance terms described above?

Assymptotically the flexibility decreases the bias, as since the function space is bigger, there is in some sense a higher chance that the space contains a function that is very similar to the real function underlying the process. On the other hand, higher flexibility means there are functions in the function space that charactherize the noise as well (in the sense that they overfit), and therefore we expect the variance to be higher. This can be formalized by saying that if we have two function spaces $E_1 \subseteq E_2$, then for any function we try to approximate, the bias error of E_2 will allways be smaller (or equal) to the one for E_2.

4 Question 4

Cross Validation for Polynomial Regression. (18 points) Consider the following four data generating processes:

- DGP 1: $Y = -2 * 1_{\{X<-3\}} + 2.55 * 1_{\{X>-2\}} - 2 * 1_{\{X>0\}} + 4 * 1_{\{X>2\}} - 1 * 1_{\{X>3\}} + \epsilon$
- DGP 2: $Y = 6 + 0.4X - 0.36X^2 + 0.005X^3 + \epsilon$
- DGP 3: $Y = 2.83 * $\sin(\frac{\pi}{2} \times X) + \epsilon \$ DGP4 : Y = 4 * \sin(3\pi \times X) * 1_{\{X>0\}} + \epsilon$

X is drawn from the uniform distribution in [-4,4] and ϵ is drawn from a standard normal ($\mu = 0$, σ^2 = 1).

```
In [24]: DGP1 = function(X){(
             -2   *(X < -3)
             +2.55*(X > -2)
             -2   *(X >  0)
             +4   *(X >  2)
             -1   *(X >  3))}
         DGP2 = function(X){
             6+0.4*X-0.36*X^2+0.005*X^3
         }
         DGP3 = function(X){
             2.83*sin(pi/2*X)
         }
         DGP4 = function(X){
             4*sin(3*pi*X)*(X>0)
         }
         ERR = rnorm
         DGP = c(DGP1,DGP2,DGP3,DGP4)
```

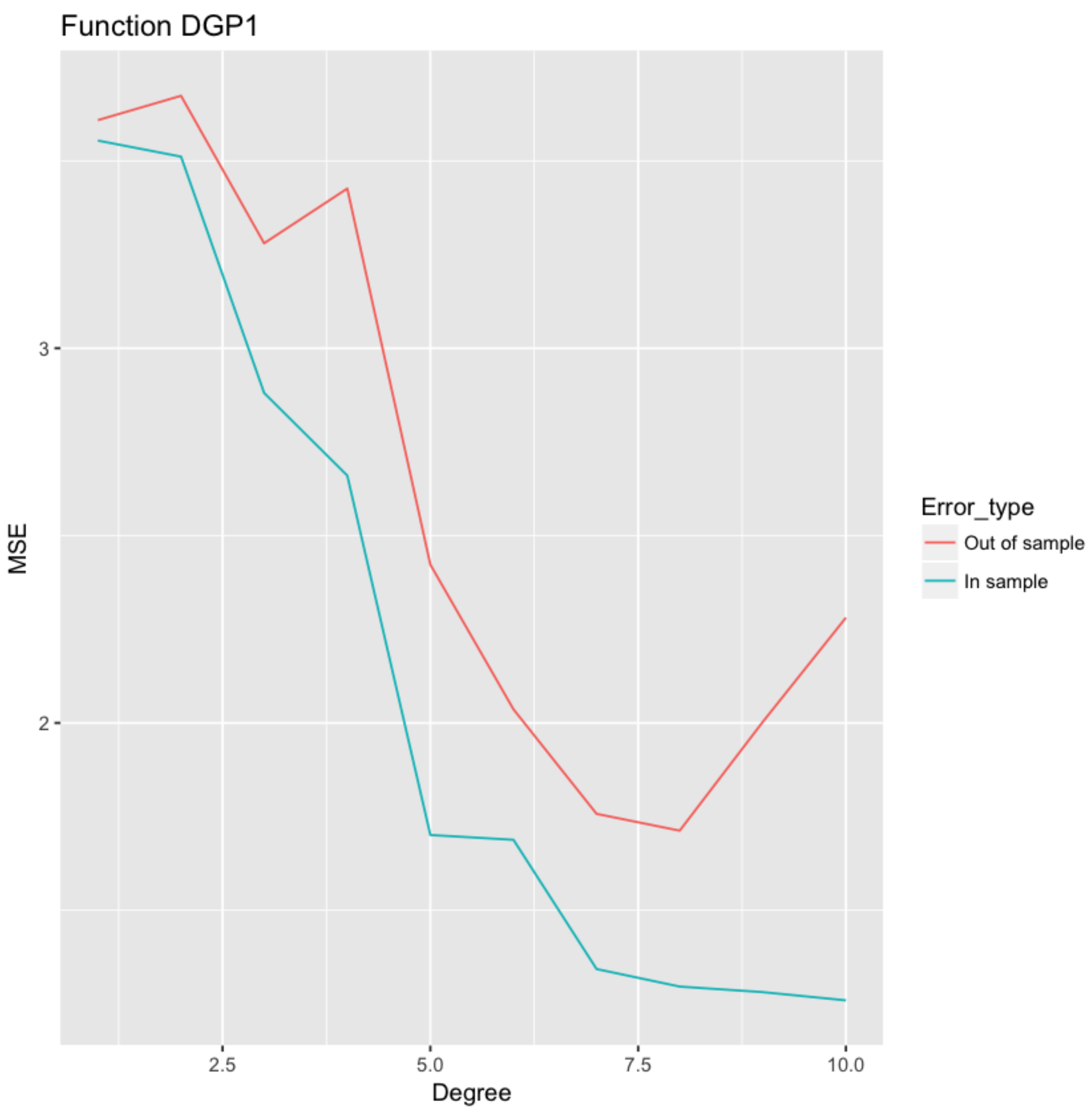
(5 pts.) Write a function to estimate the generalization error of a polynomial by k-fold cross-validation. It should take as arguments the data, the degree of the polynomial, and the number of folds k. It should return the cross-validation mean squared error.

```
In [25]: genError = function(df,deg,k){
             n = nrow(df);
             sampler = sample(n)
             df = df[sampler,]
             folds = cut(seq(1,n),breaks=k,labels=FALSE)
             sse = 0;
             ise = 0;
             for(i in 1:k)
             {
                 ind = which(folds==i,arr.ind=TRUE)
                 test = df[ind,]
                 train = df[-ind,]
                 model = lm(Y ~ poly(X,deg), data = train)

                 newY =  predict(model,test)
                 sse = sse + sum((newY-test$Y)**2)
                 newY =  predict(model,train)
                 ise = ise + sum((newY-train$Y)**2)
             }
             return(list(sse/n,ise/(n*(k-1)),model))
         }
```

Now a function to make some plots using this!

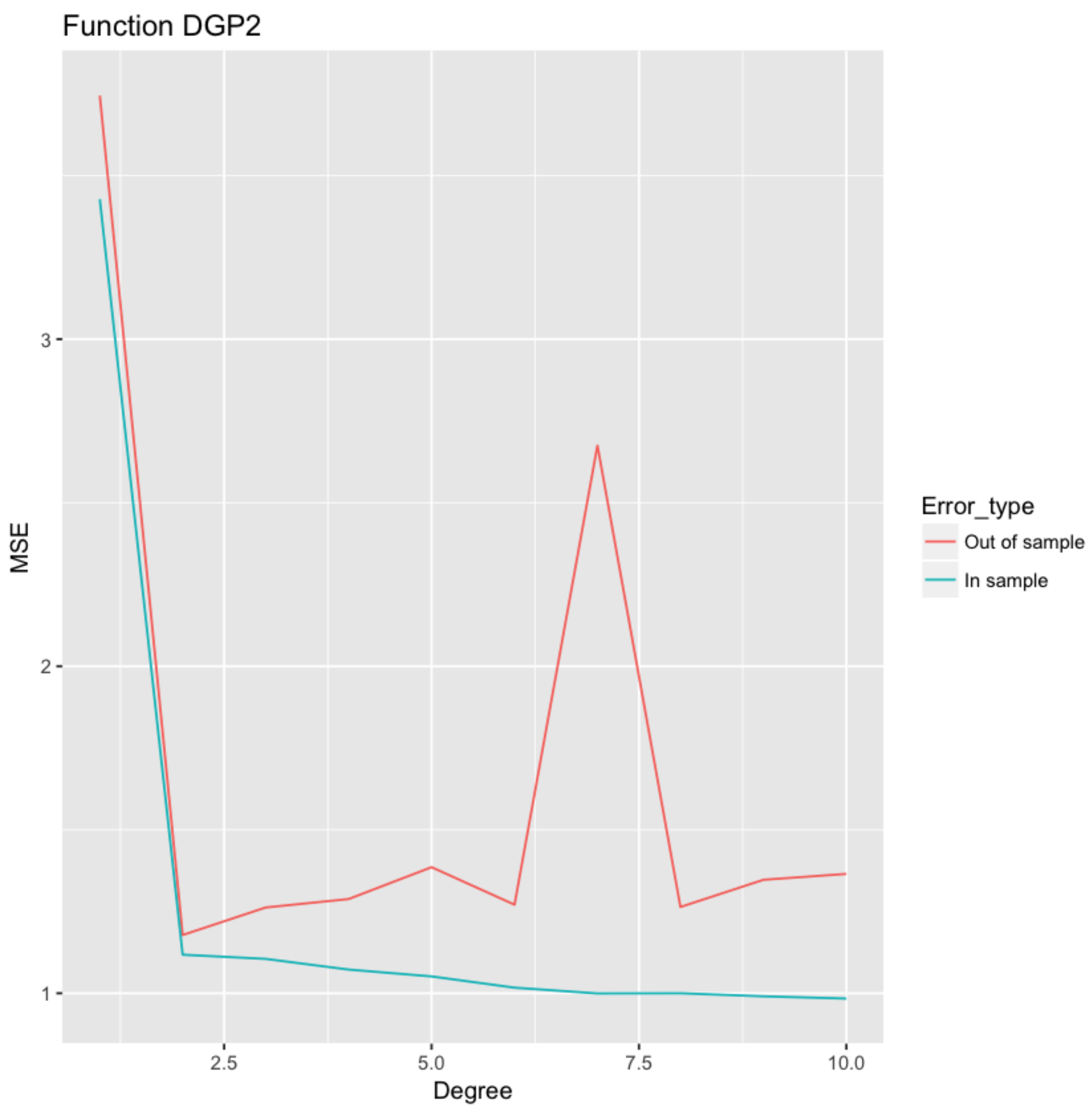
(5 pts.) Generate a dataset of size $n = 100$ from each of the four DGPs. For OLS fits of polynomials of orders $d = 0 \ldots 10$, calculate the in-sample MSE and the cross-validated MSE (with $k = 10$). For each dataset, plot the in-sample mean squared error and the cross-validated MSE as a function of d. For each dataset, what order polynomial has the best out of sample performance?

```
In [26]: N = 100
         X = 8*runif(N)-4
         R = sapply(DGP,function(y){y(X)+ERR(X)})
         df = data.frame(R)
         colnames(df) = 1:4
         df$x = X
         make_the_plot= function (i){
             helper = function(deg){
                 d = data.frame(df$x,df[i])
                 colnames(d) = c('X','Y')
                 genError(d,deg,4)
             }

             res = sapply(c(1:10),helper)
             mse = unlist(res[1,])
             ise = unlist(res[2,])
             deg = c(1:10)


             results = data.frame(deg,mse,ise)
             names(results) = c('degree','Out of sample','In sample')
             results2 = results

             df = melt(results ,  id.vars = 'degree')
             names(df) = c('Degree', 'Error_type','MSE')


             plot = ggplot(df, aes(x = Degree,y = MSE))
             plot = plot + geom_line(aes(colour = Error_type))
             plot = plot + ggtitle(paste("Function DGP",toString(i),sep=''))
             list(results,which.min(results[['Out of sample']]),plot)
         }

In [27]: x1 = make_the_plot(1)
         x1[[3]]
         x1[[2]]

         x2 = make_the_plot(2)
         x2[[3]]
         x2[[2]]

         x3 = make_the_plot(3)
         x3[[3]]
         x3[[2]]

         x4 = make_the_plot(4)
         x4[[3]]
         x4[[2]]

   8
```

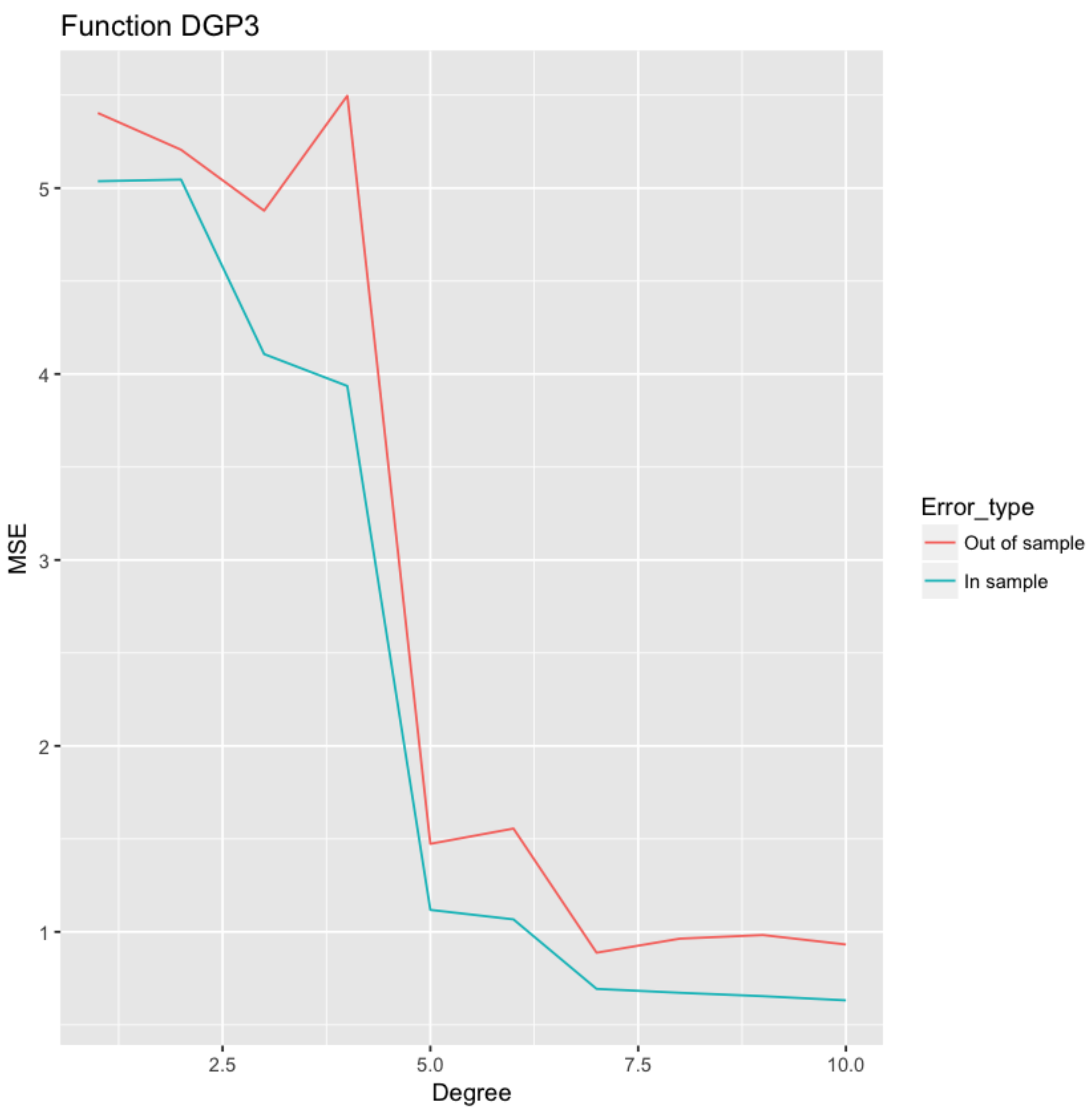

Function DGP1
MSE
3
2
Degree
2.5
5.0
7.5
10.0
Error_type
Out of sample
In sample

-

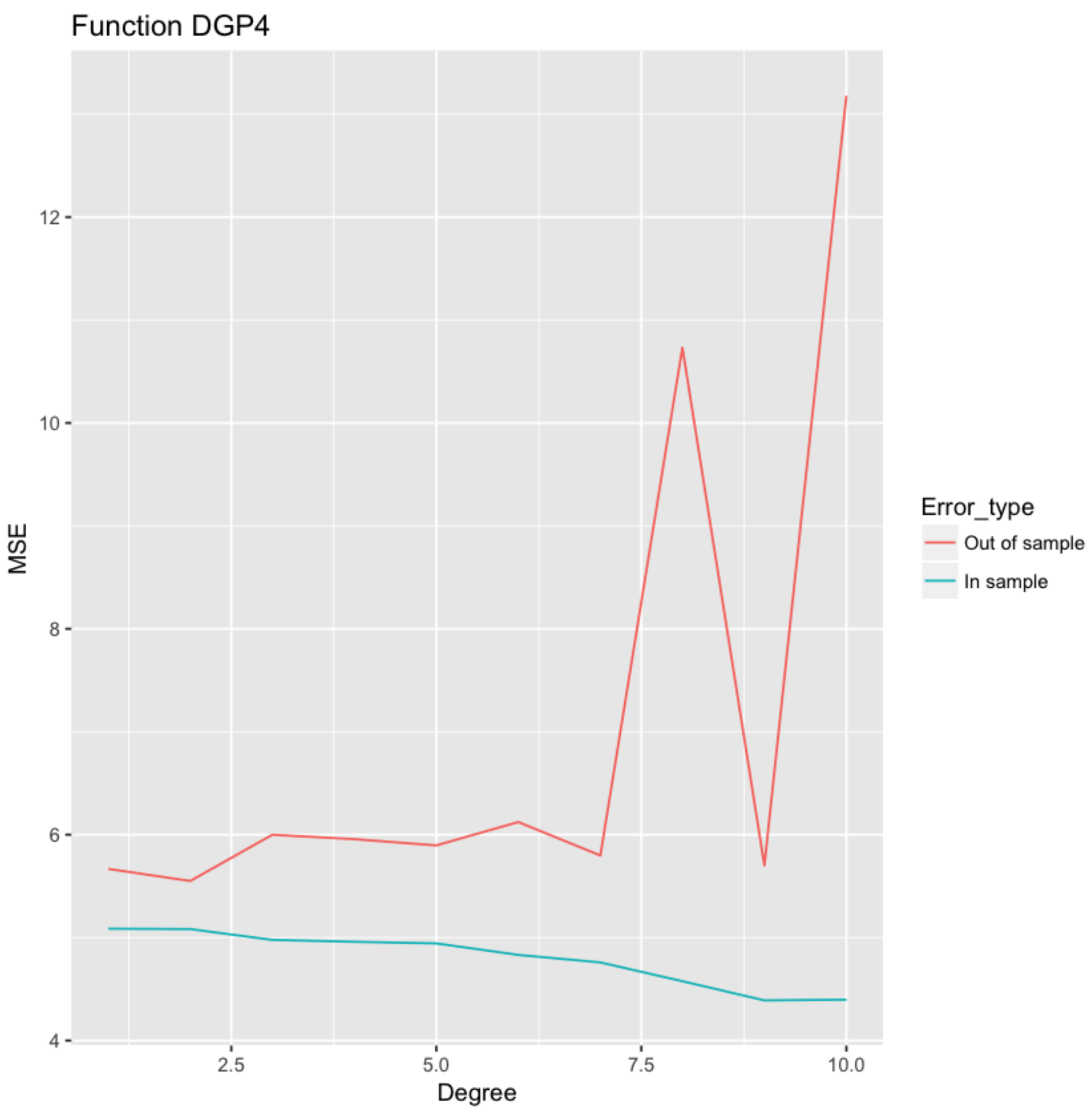
2


Function DGP2
MSE
3
2
1
2.5
5.0
7.5
10.0
Degree
Error_type
Out of sample
In sample

7


Function DGP3
MSE
5
4
3
2
1
2.5
5.0
7.5
10.0
Degree
Error_type
Out of sample
In sample

2

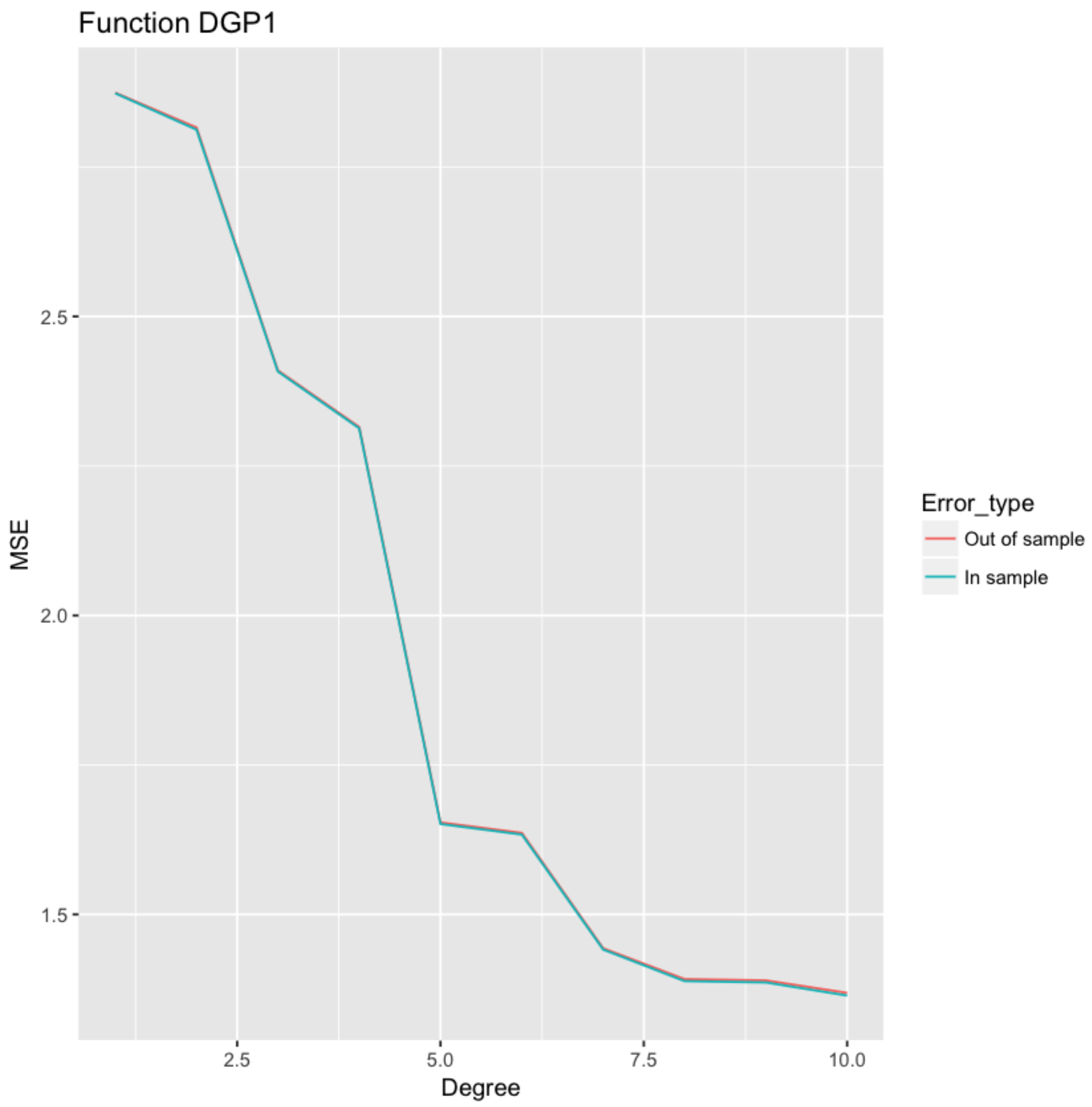


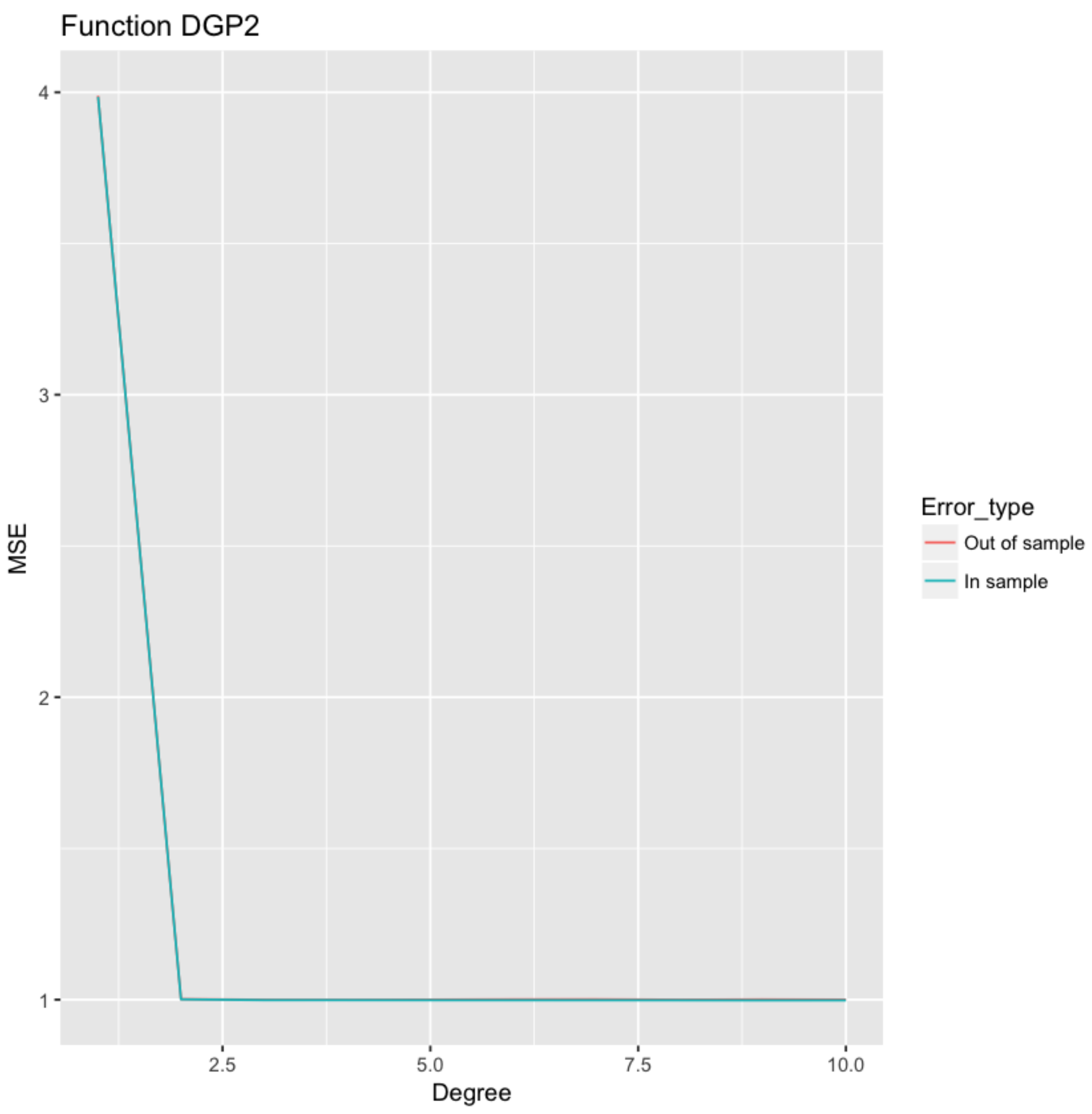
```
In [ ]: c(x1[[2]],x2[[2]],x3[[2]],x4[[2]])
```

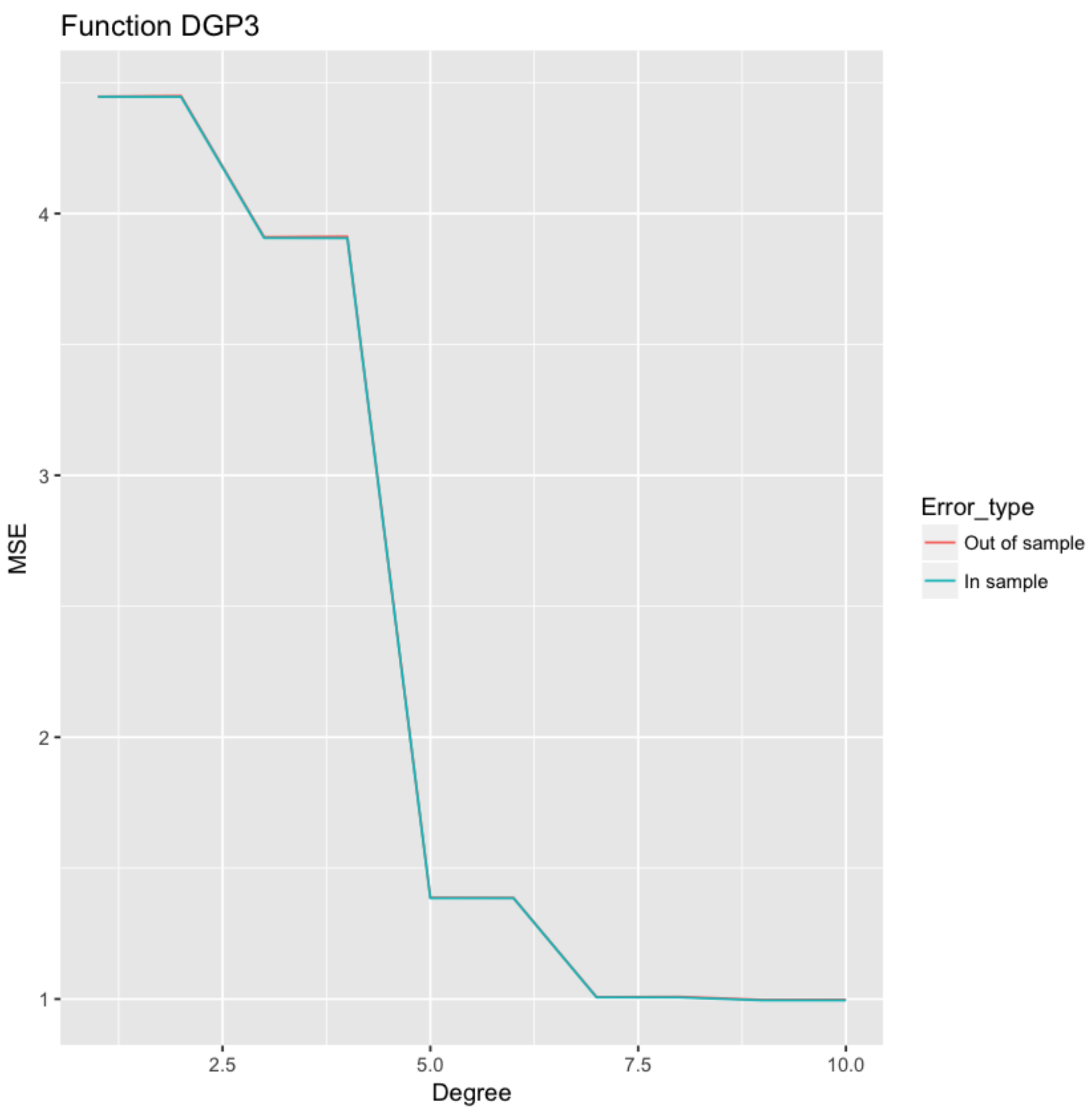
Now if we repeat again for the case of 10^4 we obtain:

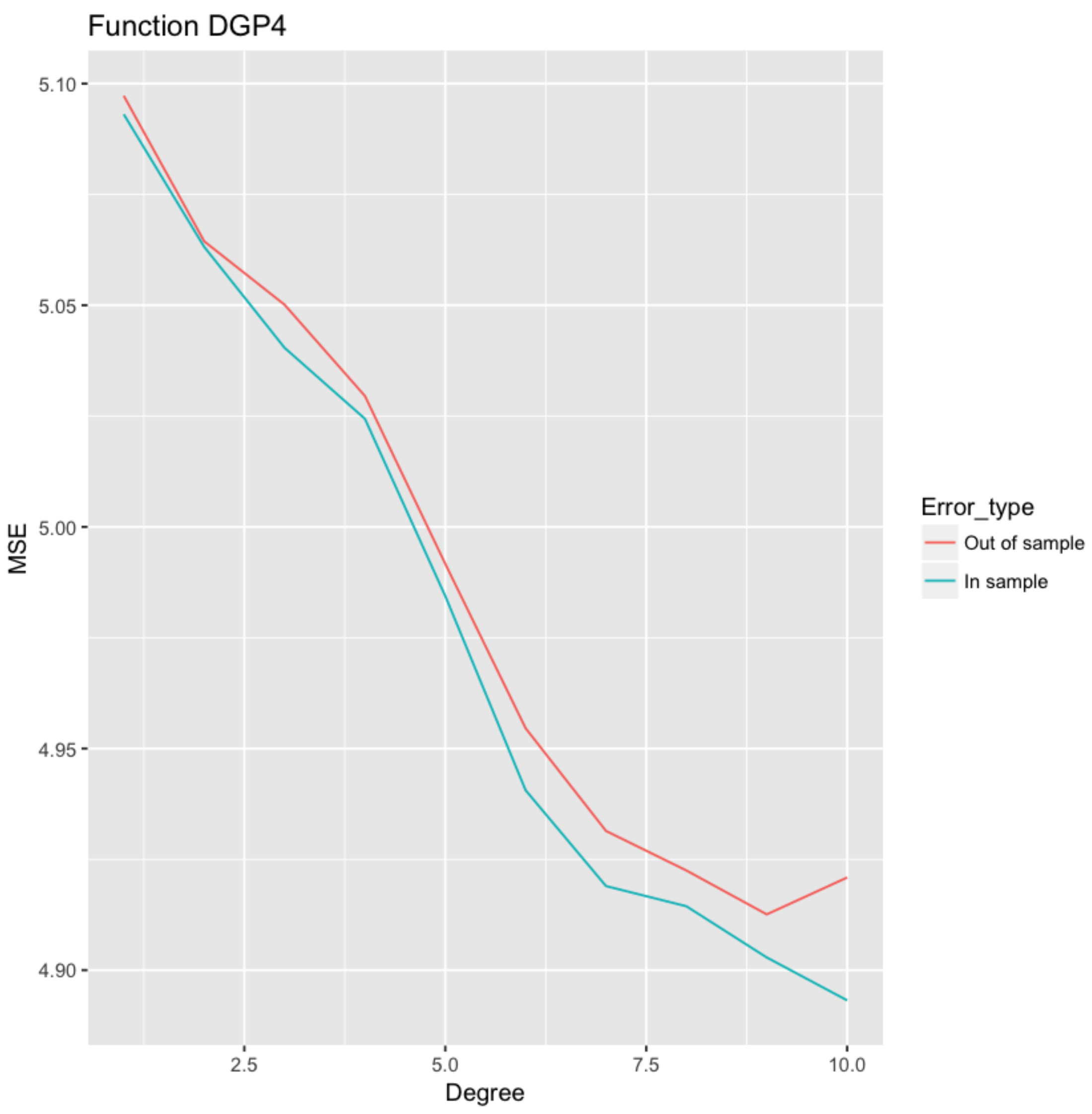
```
In [29]: N = 10000
         X = 8*runif(N)-4
         R = sapply(DGP,function(y){y(X)+ERR(X)})
         df = data.frame(R)
         colnames(df) = 1:4
         df$x = X
         x1 = make_the_plot(1)
         x1[[3]]

         x2 = make_the_plot(2)
         x2[[3]]
```

```
x3 = make_the_plot(3)
x3[[3]]

x4 = make_the_plot(4)
x4[[3]]
```




Function DGP2
MSE
4
3
2
1
2.5
5.0
7.5
10.0
Degree
Error_type
Out of sample
In sample


Function DGP3
MSE
4
3
2
1
2.5
5.0
7.5
10.0
Degree
Error_type
Out of sample
In sample



```
In [30]: c(x1[[2]],x2[[2]],x3[[2]],x4[[2]])
```

```
10
4
9
9
```

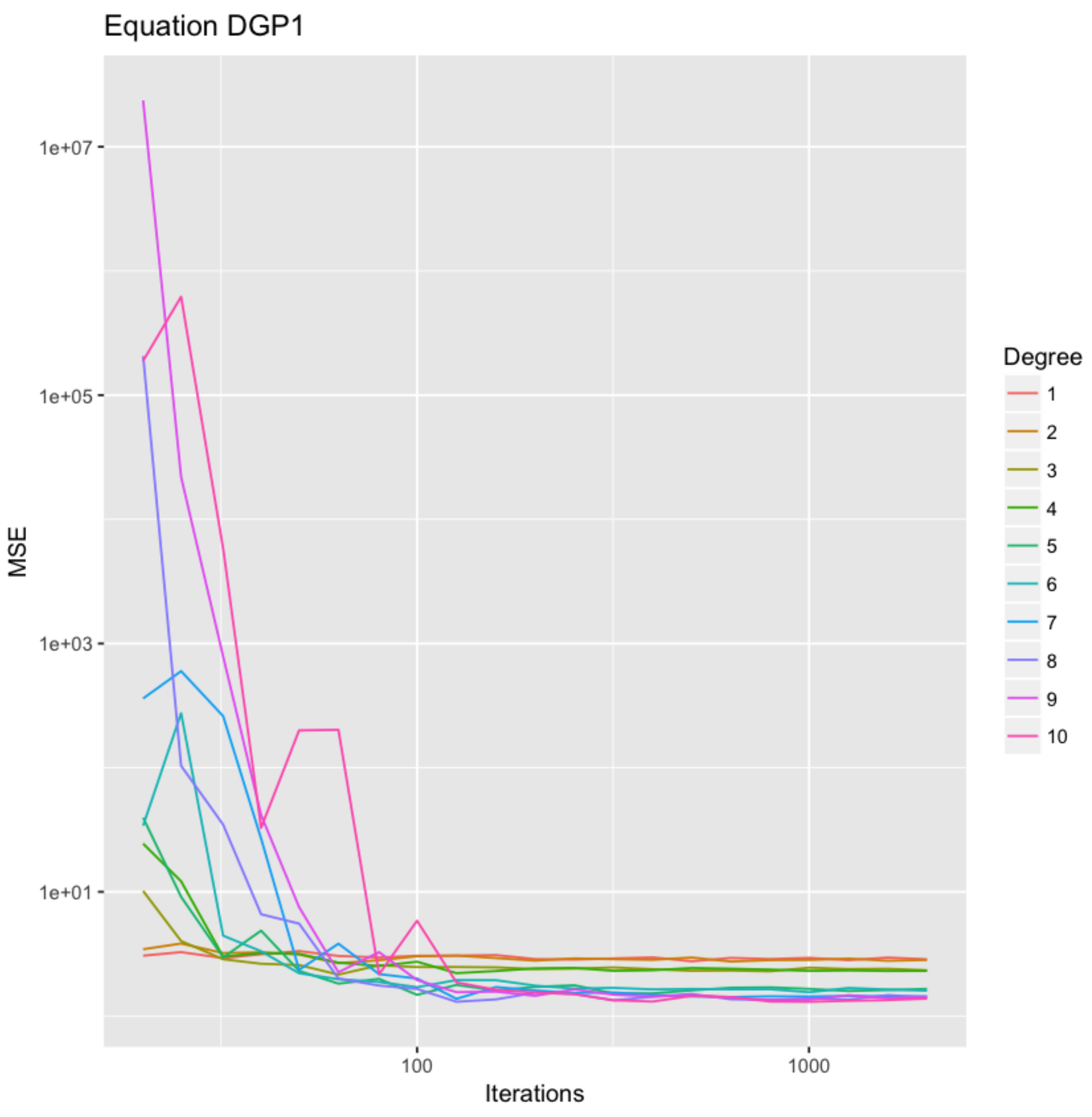
Extra: How does the out of sample error improve with the data for every degree?

```
In [31]: testCompare= function (interval_d,f,interval_i,n_times=3){
             h = function(N,D){
                 r = 0;
                 n = round(max(n_times,(n_times*1E2)/N+1))
                 for(i in 1:n){
                     X = 8*runif(N)-4
                     Y = f(X)+ERR(X)
```

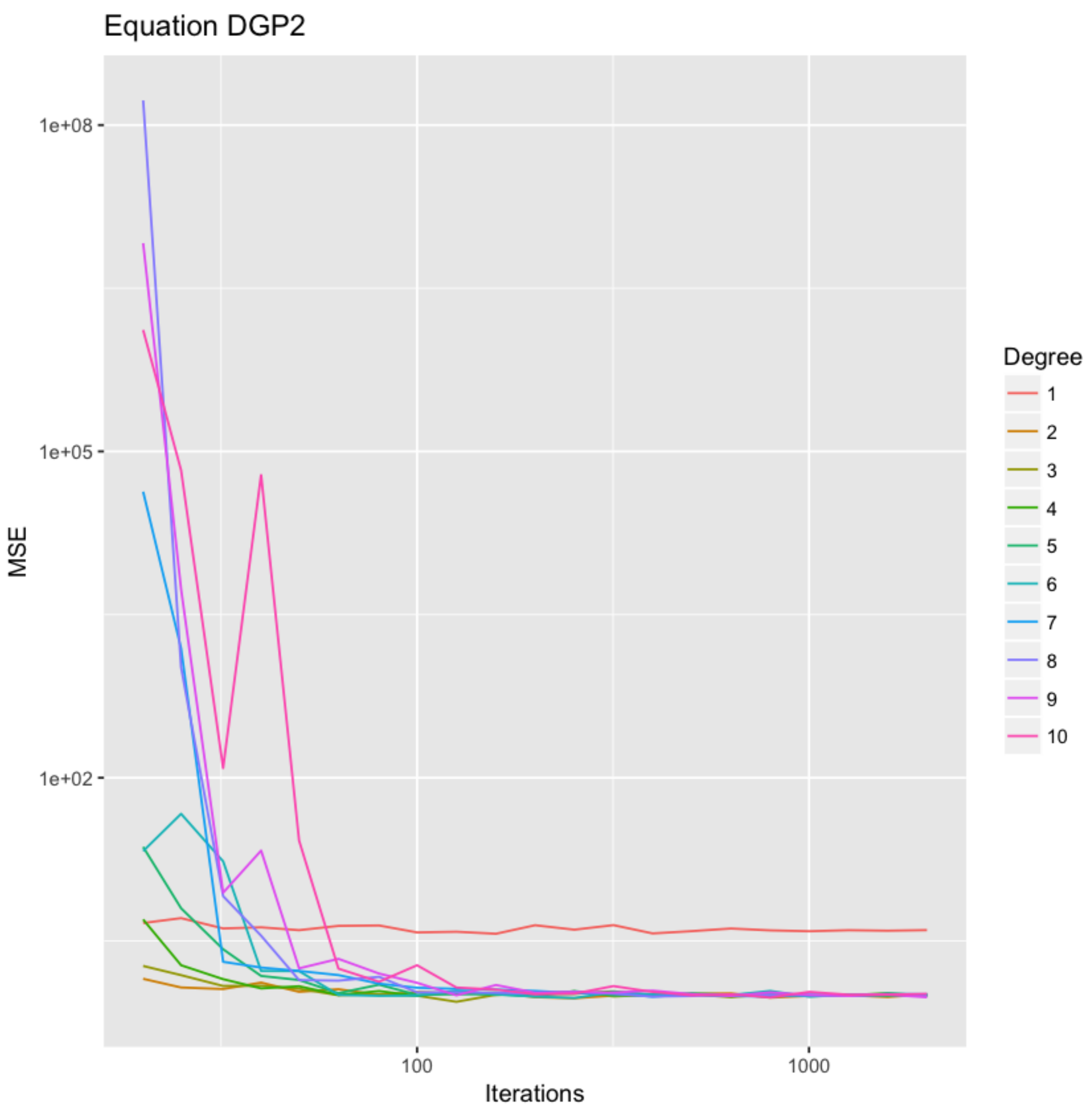
```
                df = data.frame(X,Y)
                r=r+genError(df,D,4)[[1]]
            }
            r/n
        }
        r = sapply(interval_d,
            function(D){
                h2 = function(N){h(N,D)}
                sapply(interval_i,
                       h2)
            }
           )
        r = data.frame(r)
        colnames(r) = lapply(interval_d,
                             toString)
        r$x = interval_i
        r
    }
```

```
In [32]: make_the_plot=function(i){
             degrees = c(1:10)
             x = round(c(20*100^((0:20)/20)))
             r = testCompare(degrees,DGP[[i]],x)
             df = melt(r ,  id.vars = 'x')
             names(df) = c('Iterations', 'Degree','MSE')
             plot = ggplot(df, aes(Iterations,MSE))
             plot = plot + geom_line(aes(colour = Degree))
             plot = plot + scale_y_log10() + scale_x_log10()
             plot = plot + ggtitle(paste("Equation DGP",toString(i),sep=''))
             plot
         }
```

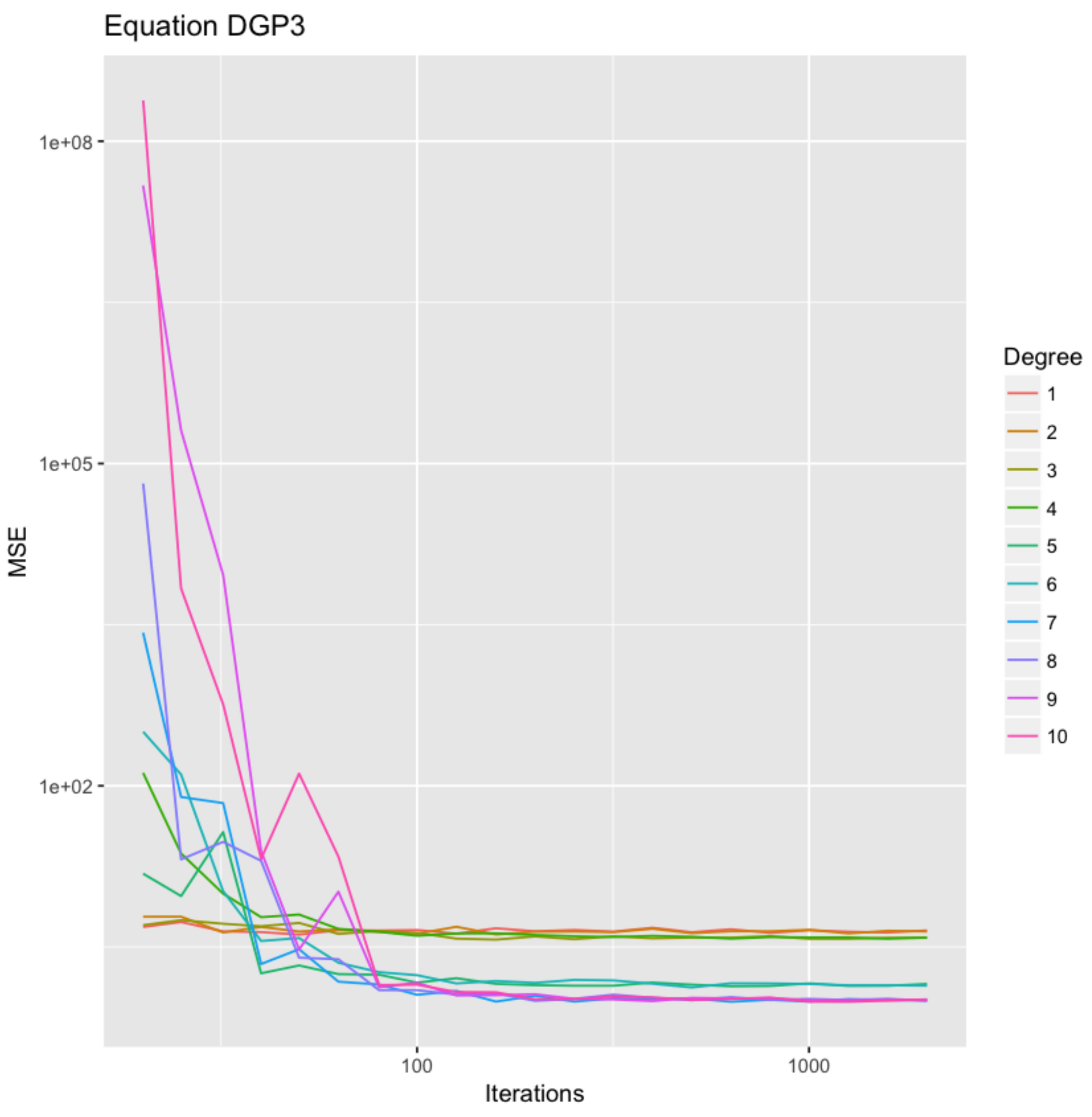
```
In [33]: make_the_plot(1)
```



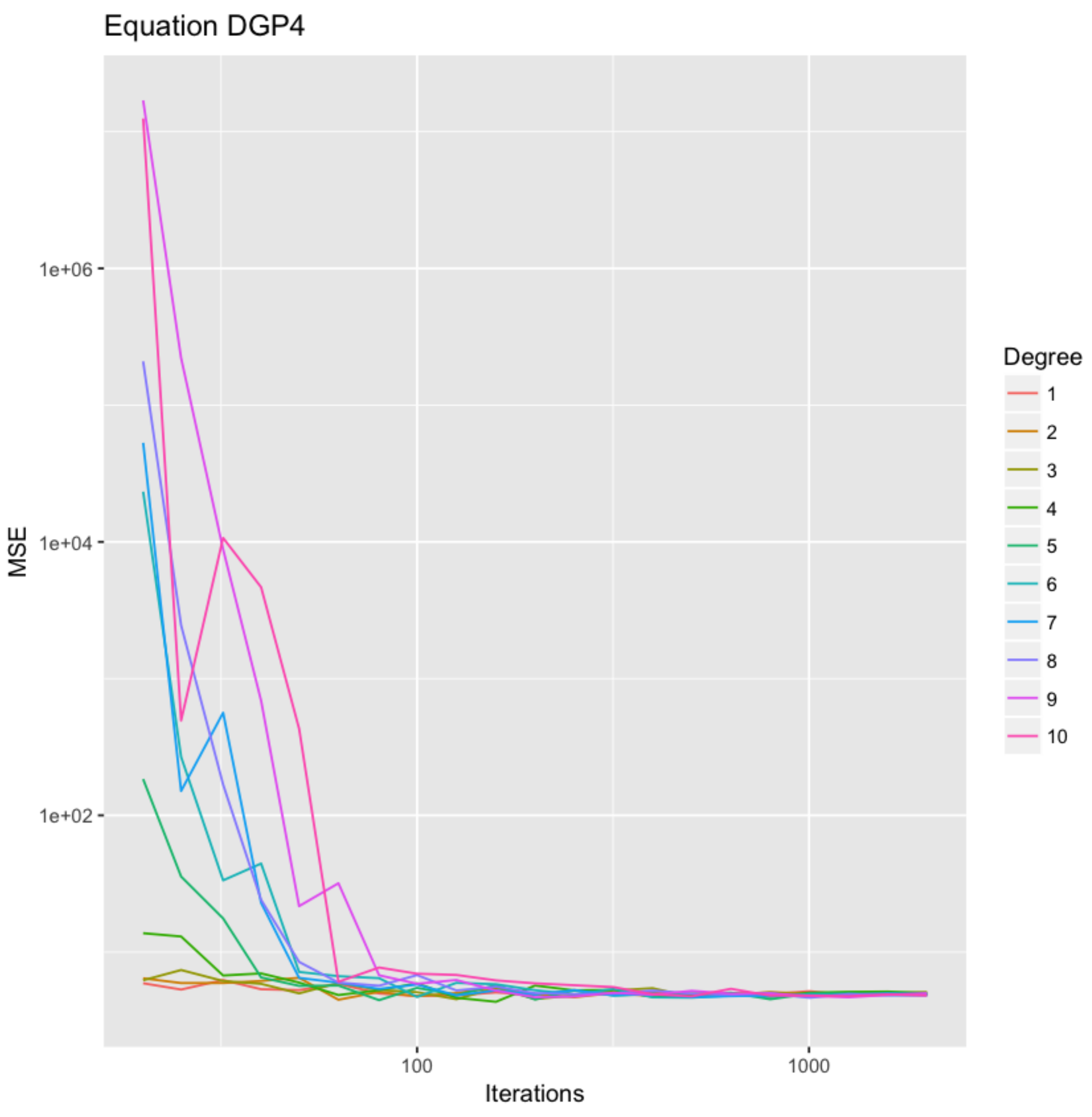
```
In [34]: make_the_plot(2)
```



```
In [35]: make_the_plot(3)
```



```
In [36]: make_the_plot(4)
```


Equation DGP4
MSE
1e+06
1e+04
1e+02
Iterations
100
1000
Degree
1
2
3
4
5
6
7
8
9
10