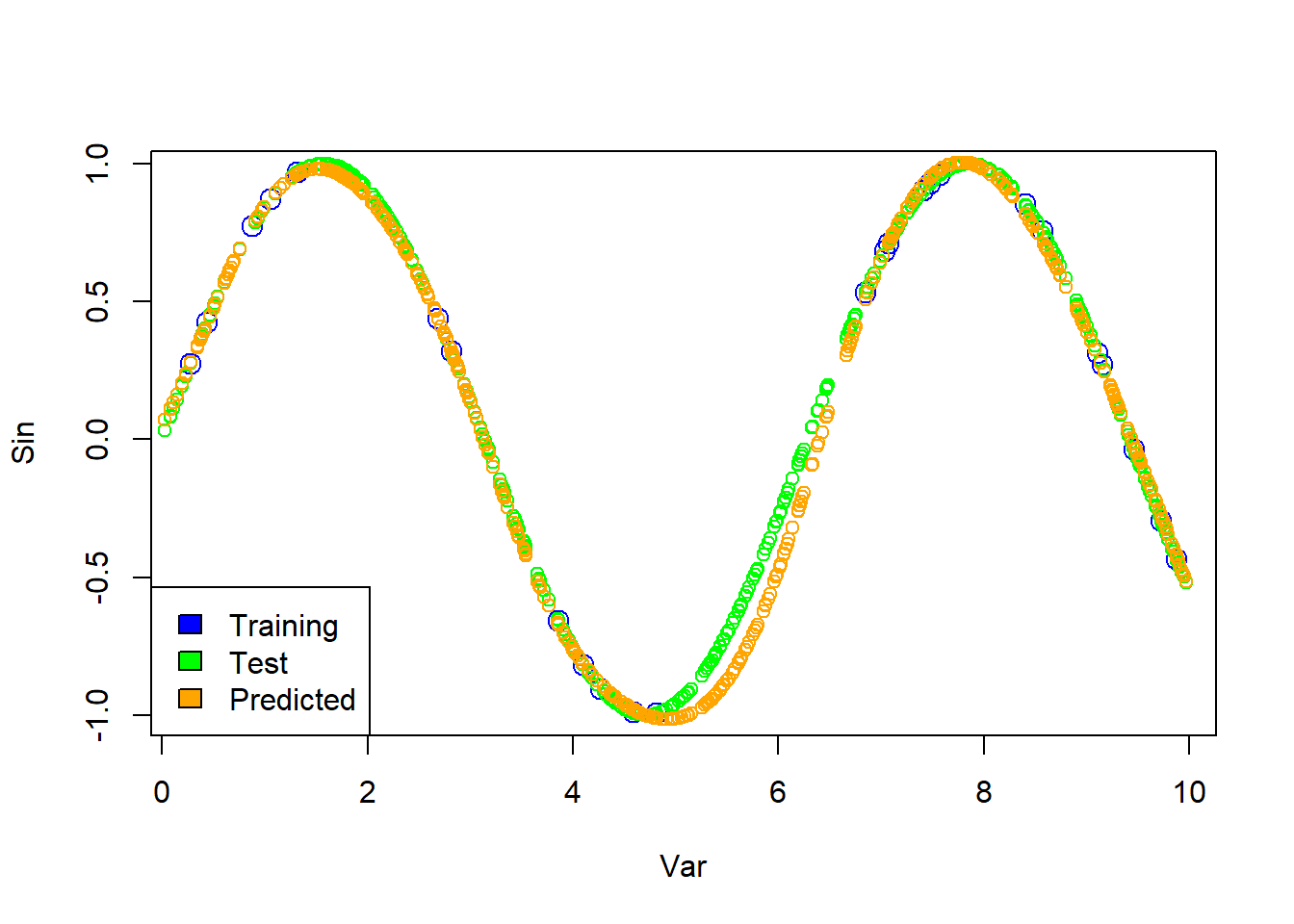
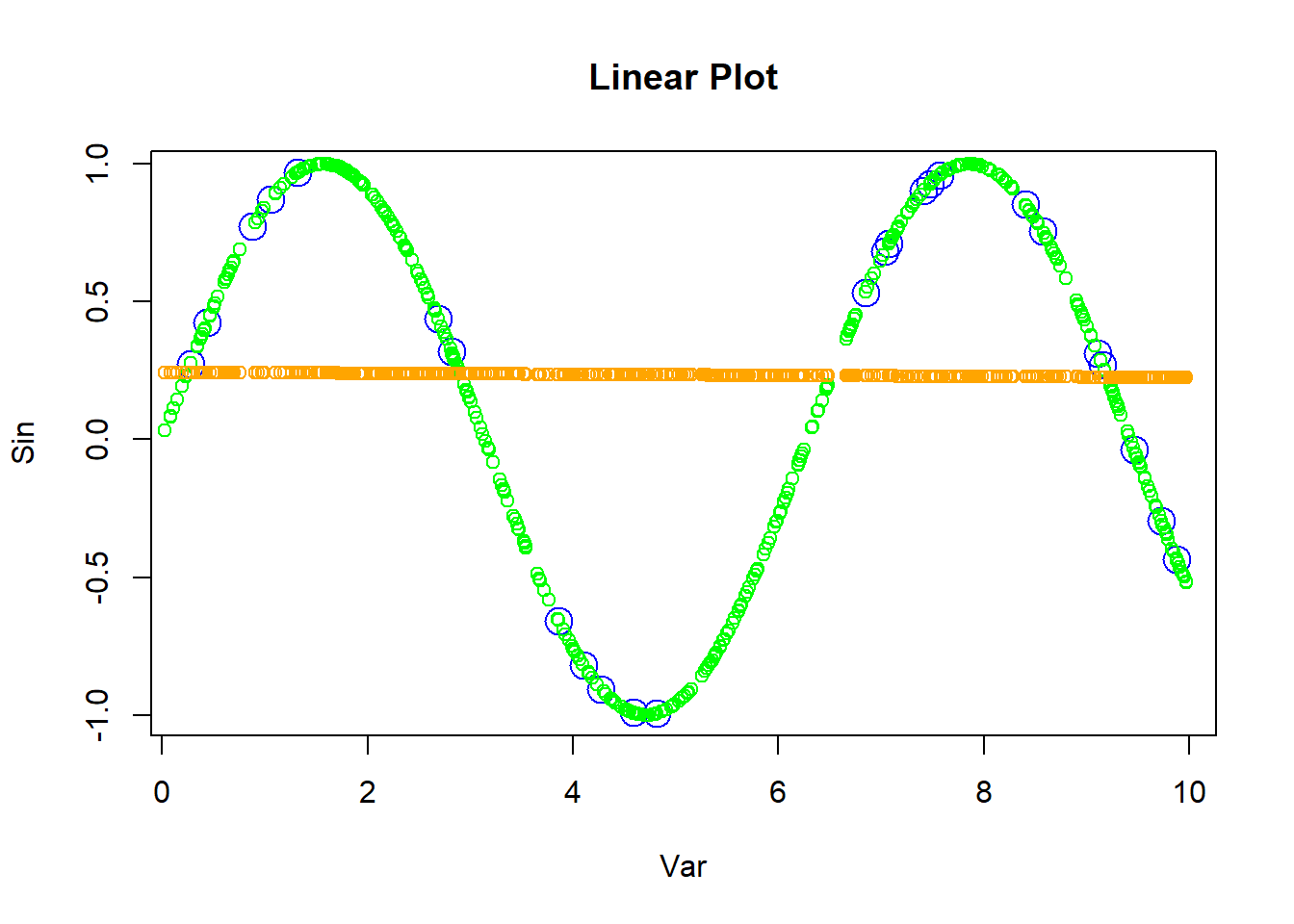
setwd("C:/Masters/TDDE01-Machine Learning/Lab/Lab3")  
  
library(neuralnet)  
library(ggplot2)  
set.seed(1234567890)  
  
####Task1####  
  
Var <- runif(500, 0 , 10)  
mydata <- data.frame(Var, Sin= sin(Var))  
  
tr <- mydata[1:25,] #Training Data  
te <- mydata[26:500,] #Test Data  
  
tr\_nn <- neuralnet(formula = Sin ~Var, data = tr, hidden = 10)  
  
predict\_te <- predict(tr\_nn, te)  
  
plot(tr, col= "blue",cex = 1.5)  
points(te, col = "green")  
points(te[,1],predict\_te, col="orange")  
legend("bottomleft", c("Training", "Test", "Predicted"), fill= c("blue","green", "orange"))



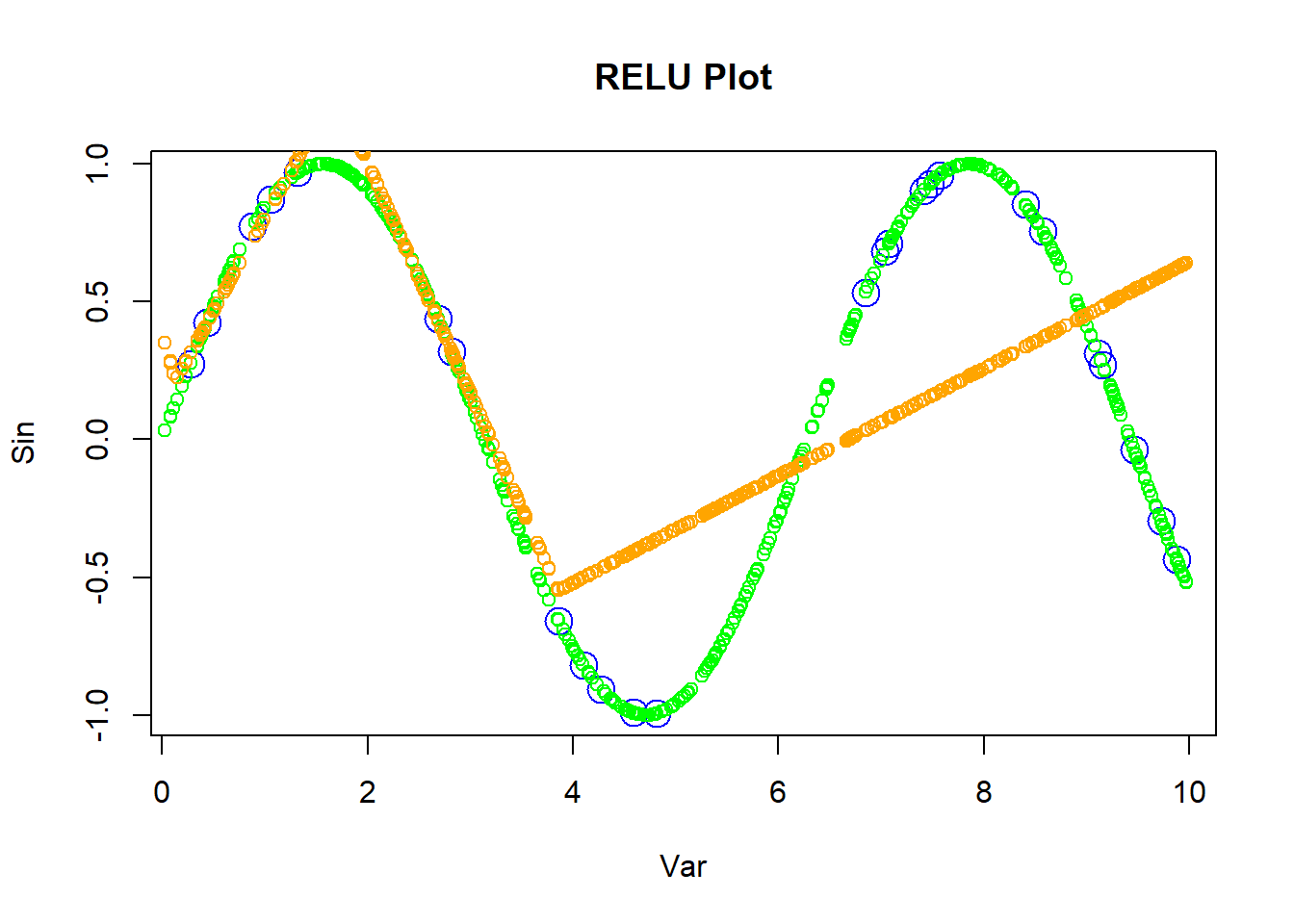
**Question 1: Train a neural network to learn the trigonometric sine function. To do so, sample 500 points uniformly at random in the interval [0,10]. Apply the sine function to each point. The resulting value pairs are the data points available to you. Use 25 of the 500 points for training and the rest for test. Use one hidden layer with 10 hidden units. You do not need to apply early stopping. Plot the training and test data, and the predictions of the learned NN on the test data. You should get good results. Comment your results**

From the above plot, we can see that predictions data point are nearly matching the training and test data points, as we can observe the difference between data points between range 4 and 6. So the predictions of Sin function is Good.

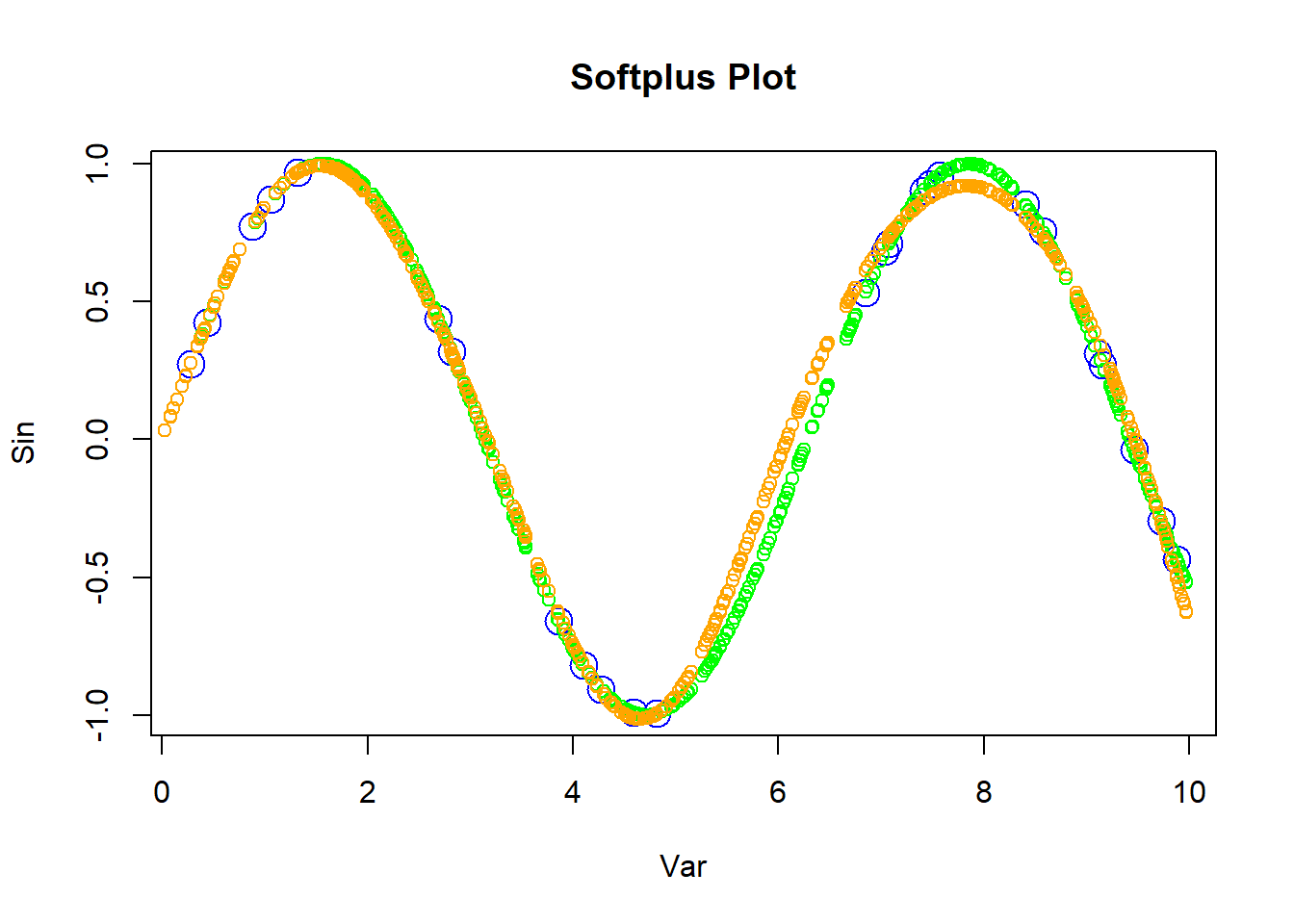
####Task2####  
  
linear\_activation <- function(x)  
{  
 y <- x  
}  
  
linear\_tr\_nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = 10, act.fct = linear\_activation)  
predict\_tr\_h1 <- predict(linear\_tr\_nn, te)  
  
plot(tr, col = "blue", cex= 2, main = "Linear Plot")  
points(te, col = "green", cex= 1)  
points(te[,1], predict\_tr\_h1, col = "orange" , cex= 1 )



relu\_activation <- function(x)  
{  
 ifelse(x>0, x,0)  
}  
  
relu\_nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = 10, act.fct = relu\_activation)  
predict\_te\_h2 <- predict(relu\_nn, te)  
  
plot(tr, col = "blue", cex= 2, main = "RELU Plot")  
points(te, col = "green", cex= 1)  
points(te[,1], predict\_te\_h2, col = "orange" , cex= 1 )



softplus\_activation <- function(x)  
{  
 y = log(1 + exp(x))  
}  
  
softplus\_nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = 10, act.fct = softplus\_activation)  
predict\_te\_h3 <- predict(softplus\_nn, te)  
  
plot(tr, col = "blue", cex= 2, main = "Softplus Plot")  
points(te, col = "green", cex= 1)  
points(te[,1], predict\_te\_h3, col = "orange" , cex= 1 )



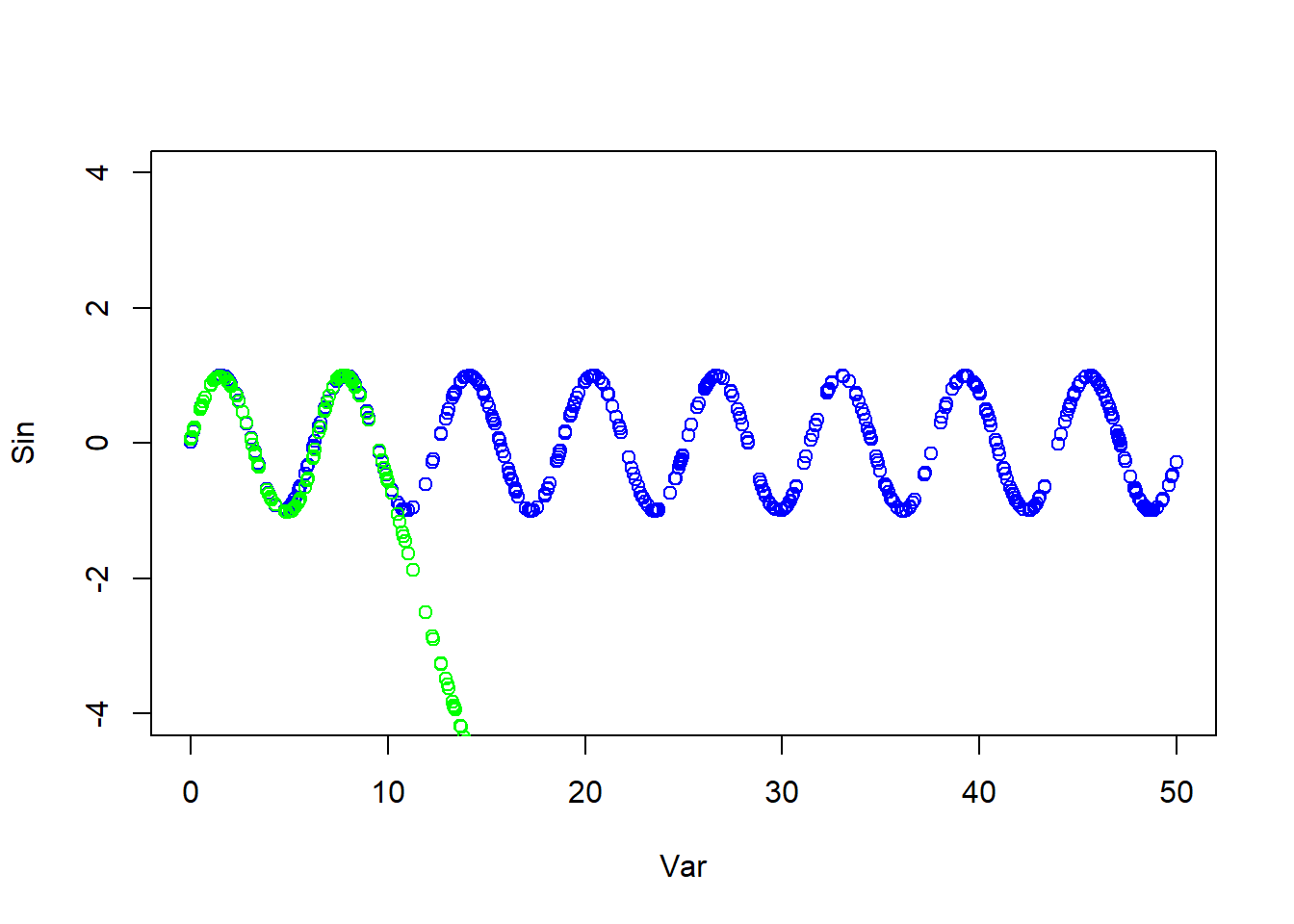
**Question 2 In question (1), you used the default logistic (a.k.a. sigmoid) activation function, i.e. act.fct = “logistic”. Repeat question (1) with the following custom activation functions: h1(x) = x, h2(x) = max{0, x} and h3(x) = ln(1 + exp x) (a.k.a. linear, ReLU and softplus). See the help file of the neuralnet package to learn how to use custom activation functions. Plot and comment your results**

According to Plot1, we can see a flat line which indicates the non complex learining patterns. Since we used Linear Activation, which does not takes non-linear features into account.

In Plot2, we can observe that RELU does capture some non-linear features.However data points deviates when compared to the sin plot, by forming a straight line when the data points are negative.

From Plot3, the data points are nearly matching the test data points, as the softmax activation function allows the model to learn complex features.

####Task3####  
  
Var <- runif(500, 0, 50)  
mydata\_1 <- data.frame(Var, Sin = sin(Var))  
  
predict\_tr\_1 <- predict(tr\_nn, mydata\_1 )  
  
plot(mydata\_1, col="blue", ylim= c(-4,4))  
points(mydata\_1[,1], predict\_tr\_1, col="green")



**Question 3: Sample 500 points uniformly at random in the interval [0,50], and apply the sine function to each point. Use the NN learned in question (1) to predict the sine function value for these new 500 points. You should get mixed results. Plot and comment your results.**

From the above plot, we can oberserve the prediced data points after the interval 10 drops and differs from the data points. That reason being data points with new interval has not been trained.

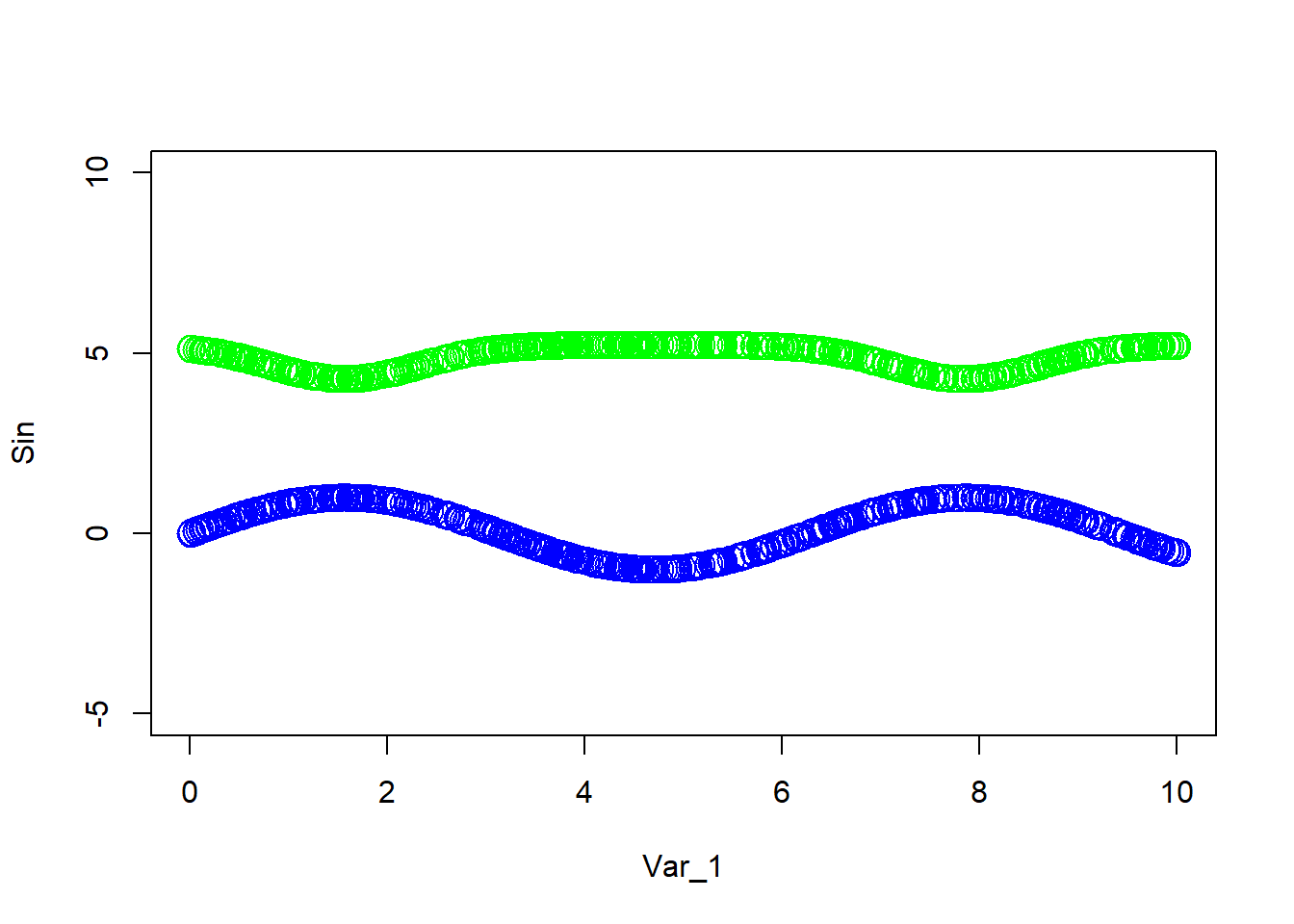
####Task4####  
  
tr\_nn$weights

## [[1]]  
## [[1]][[1]]  
## [,1] [,2] [,3] [,4] [,5] [,6] [,7]  
## [1,] 3.965406 -1.231339 -2.7255077 7.387216 -0.43923716 0.8143266 16.949728  
## [2,] -1.848497 -1.108160 0.2616825 -2.298154 -0.05618884 -1.9677321 -2.510252  
## [,8] [,9] [,10]  
## [1,] -0.8458926 2.4703284 -11.618491  
## [2,] -1.1041865 -0.7615488 1.755885  
##   
## [[1]][[2]]  
## [,1]  
## [1,] -0.8272835  
## [2,] 0.6261725  
## [3,] -1.3652234  
## [4,] -15.2546738  
## [5,] 1.4724743  
## [6,] 4.0194898  
## [7,] -2.3705710  
## [8,] 1.5225230  
## [9,] 1.3549643  
## [10,] -1.9815383  
## [11,] 6.4412888

**Question 4: In question (3), the predictions seem to converge to some value. Explain why this happens. To answer this question, you may need to get access to the weights of the NN learned. You can do it by running nn or nn$weights where nn is the NN learned.**

From the above result, as mentioned in question 3, test data has not been trained and the interval is huge[0,50] when compared to the train data which is less[0,10]. So the neurons in the hidden layer would be much larger for the test data than the training data. The relatively large biases in the hidden layer might influence the hidden layer’s neurons to prefer certain ranges of input values, leading to a bias in the overall predictions.

####Task5####  
  
Var\_1 <- runif(500, 0, 10)  
mydata\_2 <- data.frame(Var\_1, Sin = sin(Var\_1))  
tr\_nn\_new <- neuralnet(formula = Var\_1~Sin, data = mydata\_2, threshold = 0.1)   
  
predict\_new <- predict(tr\_nn\_new, mydata\_2)  
  
plot(mydata\_2, col = "blue", cex = 2, ylim = c(-5,10))  
points(mydata\_2[,1], predict\_new, col = "green", cex=2)



**Question 5: Sample 500 points uniformly at random in the interval [0,10], and apply the sine function to each point. Use all these points as training points for learning a NN that tries to predict x from sin(x), i.e. unlike before when the goal was to predict sin(x) from x. Use the learned NN to predict the training data. You should get bad results. Plot and comment your results Help: Some people get a convergence error in this question. It can be solved by stopping the training before reaching convergence by setting threshold = 0.1**

From the above plot, we can observe the predicted value does not closely follow the actual training data, as model does not capture the relationship features between the input data and the sin data.