DA5030 Project - Student ID: 001426343 - Mohammad Nabizadehmashhadtoroghi

Problem Definition

Colloidal gels are found in many applications like food processing, cosmetics, cell biology etc. Such materials usually have a very non-linear behavior. In this work, the gelation process of a colloidal suspension is studied over time. There are two important factors that control the amount of time it takes for a gel to form, volume fraction (ϕ) and interparticle attraction (D_0). In an edible gel, take volume fraction as a measure of how much gel powder you add to the water. The more powder you add, the higher your volume fraction is. Interparticle attraction, as the name says, is a measure of how attractive those powder particles are when suspended in the fluid. There are several questions we are going to answer in this work. We're going to observe the data and answer the following questions accordingly:

- Under what conditions the mixture finally becomes a gel? (what ϕ and D_0)
- Can we predict the phase space (Given a ϕ and D_0, predict if a gel is eventually formed)?
- Can we train models to predict the gelation process with time?
- Present a generic correlation between the mixture state (gel, not gel and the transition state in between) with volume fraction, attraction and time.

Step 0:

Remove previously defined variables and clear Console and set a seed for our results to be reproducable.

```
rm(list = ls())
cat("\014")
```

```r
set.seed(12345)
```

Part 1 - Systems with low volume fraction

Data Acquisition

The data used in this project is obtained from some of my simulations of the colloidal gels. **HOOMD-blue** package is used for the simulations and the `Networkx` module in `python` is used in order to generate the particles' structure at each time of the simulation.

```r
gel.data = read.csv('N_Clusters.csv', header = TRUE)
```

Data Exploration

Let's take a look at the attributes of our data, using the `names()` function. (summary will be looked at a bit later, this is to avoid messy outputs)

```r
names(gel.data)
```

```
##   [1] "phi0.05D00"  "phi0.05D02"  "phi0.05D04"  "phi0.05D06"  "phi0.05D08"
##   [6] "phi0.05D010" "phi0.05D012" "phi0.05D014" "phi0.05D016" "phi0.05D018"
##  [11] "phi0.05D020" "phi0.05D022" "phi0.06D00"  "phi0.06D02"  "phi0.06D04"
##  [16] "phi0.06D06"  "phi0.06D08"  "phi0.06D010" "phi0.06D012" "phi0.06D014"
##  [21] "phi0.06D016" "phi0.06D018" "phi0.06D020" "phi0.06D022" "phi0.07D00"
##  [26] "phi0.07D02"  "phi0.07D04"  "phi0.07D06"  "phi0.07D08"  "phi0.07D010"
##  [31] "phi0.07D012" "phi0.07D014" "phi0.07D016" "phi0.07D018" "phi0.07D020"
##  [36] "phi0.07D022" "phi0.08D00"  "phi0.08D02"  "phi0.08D04"  "phi0.08D06"
##  [41] "phi0.08D08"  "phi0.08D010" "phi0.08D012" "phi0.08D014" "phi0.08D016"
##  [46] "phi0.08D018" "phi0.08D020" "phi0.08D022" "phi0.09D00"  "phi0.09D02"
##  [51] "phi0.09D04"  "phi0.09D06"  "phi0.09D08"  "phi0.09D010" "phi0.09D012"
##  [56] "phi0.09D014" "phi0.09D016" "phi0.09D018" "phi0.09D020" "phi0.09D022"
##  [61] "phi0.1D00"   "phi0.1D02"   "phi0.1D04"   "phi0.1D06"   "phi0.1D08"
##  [66] "phi0.1D010"  "phi0.1D012"  "phi0.1D014"  "phi0.1D016"  "phi0.1D018"
##  [71] "phi0.1D020"  "phi0.1D022"  "phi0.11D00"  "phi0.11D02"  "phi0.11D04"
##  [76] "phi0.11D06"  "phi0.11D08"  "phi0.11D010" "phi0.11D012" "phi0.11D014"
##  [81] "phi0.11D016" "phi0.11D018" "phi0.11D020" "phi0.11D022" "phi0.12D00"
##  [86] "phi0.12D02"  "phi0.12D04"  "phi0.12D06"  "phi0.12D08"  "phi0.12D010"
##  [91] "phi0.12D012" "phi0.12D014" "phi0.12D016" "phi0.12D018" "phi0.12D020"
##  [96] "phi0.12D022" "phi0.13D00"  "phi0.13D02"  "phi0.13D04"  "phi0.13D06"
## [101] "phi0.13D08"  "phi0.13D010" "phi0.13D012" "phi0.13D014" "phi0.13D016"
## [106] "phi0.13D018" "phi0.13D020" "phi0.13D022" "phi0.14D00"  "phi0.14D02"
## [111] "phi0.14D04"  "phi0.14D06"  "phi0.14D08"  "phi0.14D010" "phi0.14D012"
## [116] "phi0.14D014" "phi0.14D016" "phi0.14D018" "phi0.14D020" "phi0.14D022"
## [121] "phi0.15D00"  "phi0.15D02"  "phi0.15D04"  "phi0.15D06"  "phi0.15D08"
## [126] "phi0.15D010" "phi0.15D012" "phi0.15D014" "phi0.15D016" "phi0.15D018"
## [131] "phi0.15D020" "phi0.15D022" "timestep"
```

There are `133` attributes here. In this data, each column corresponds to a unique simulation where the value at each row corresponds to the number of interconnected clusters in the system. The name of the attribute shows the simulation configureation, for instance, "phi0.12D02" means $\phi = 0.1$ and $D_0 = 2$. A function is written here to extract these values from the names of the attributes.

```
extract_values <- function(att_name) {
  options(digits=3)
  sp.name = unlist(strsplit(att_name, ""))
  if (sp.name[7] != "D"){
      phi = as.double(paste(sp.name[4],sp.name[5],sp.name[6],sp.name[7],sep=""))
      name.length = length(sp.name)
      D0 = ""
      for (j in c(9:name.length)){
        D0 = paste(D0,sp.name[j],sep="")
      }
      D0 = as.double(D0)
  }
  if (sp.name[7] == "D"){
      phi = as.double(paste(sp.name[4],sp.name[5],sp.name[6],sep=""))
      name.length = length(sp.name)
      D0 = ""
      for (j in c(8:name.length)){
        D0 = paste(D0,sp.name[j],sep="")
      }
      D0 = as.double(D0)
  }
  return(c(phi,D0))
}
```

Let's see if the function works properly:

```
print(paste("phi and D0 for ", names(gel.data)[20], " are: ",extract_values(names(gel.data)[20])[1], " a
```

```
## [1] "phi and D0 for  phi0.06D014  are:  0.06  and  14"
```

Make a data frame to store the final state of the simulation for each configuration.

```
phi_list = c(5:15) * 0.01
D0_list = c(1:11) * 2
phase_space = matrix(0, nrow = length(phi_list), ncol = (length(D0_list)+1))
phase_space[,1] = phi_list
```

Now, let's take a look at some some basic details of the data. Use `summary()` function to observe 10 random attributes:

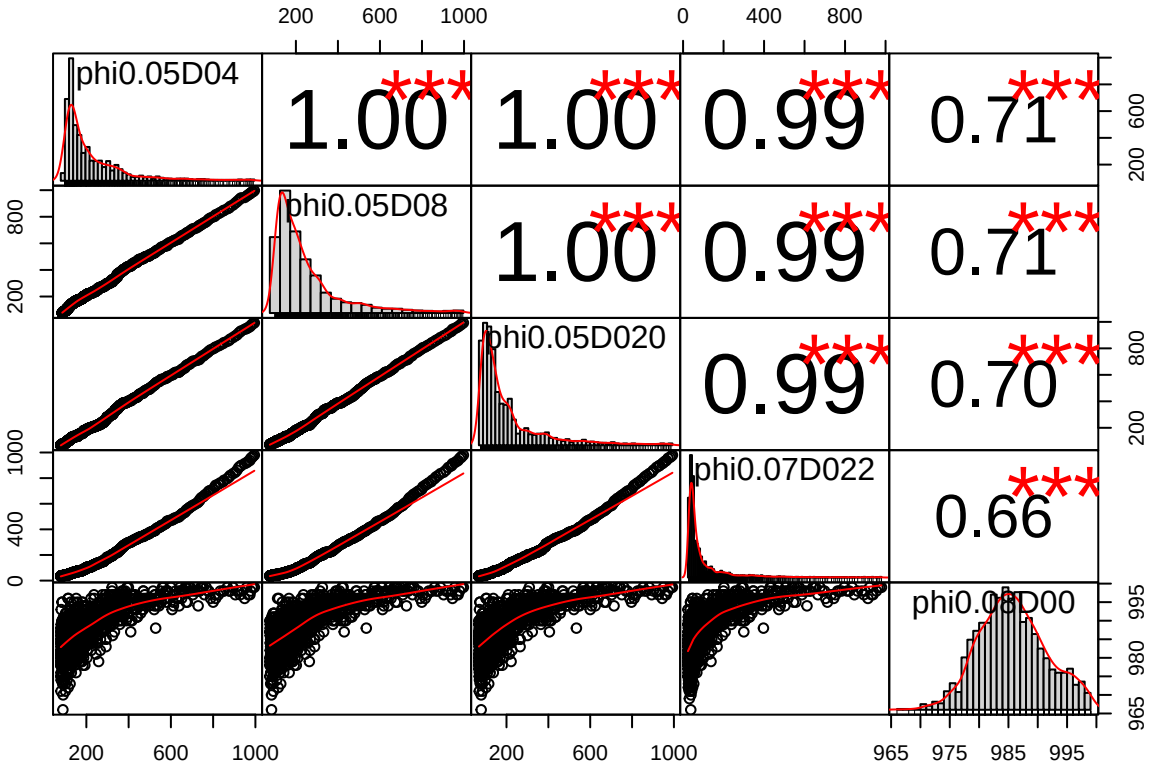
```
summary(gel.data[,sample(ncol(gel.data),10)])
```

```
##    phi0.09D04    phi0.09D018    phi0.12D016     phi0.11D04    phi0.12D022
##  Min.   :  4    Min.   :  1    Min.   :  1    Min.   :  1    Min.   :  2
##  1st Qu.:  5    1st Qu.:  2    1st Qu.:  1    1st Qu.:  2    1st Qu.:  4
##  Median :  7    Median :  8    Median :  1    Median :  2    Median :  6
##  Mean   : 40    Mean   : 34    Mean   : 15    Mean   : 20    Mean   : 19
##  3rd Qu.: 19    3rd Qu.: 14    3rd Qu.:  3    3rd Qu.:  8    3rd Qu.:  8
##  Max.   :991    Max.   :984    Max.   :979    Max.   :984    Max.   :968
##  NA's   :1953   NA's   :1841   NA's   :1089   NA's   :1492   NA's   :1408
##    phi0.12D06     phi0.05D02     phi0.12D02   phi0.15D018    phi0.08D02
##  Min.   :  1    Min.   : 72    Min.   :  2   Min.   :  1   Min.   :  8
##  1st Qu.:  1    1st Qu.:111    1st Qu.:  3   1st Qu.:  1   1st Qu.: 12
##  Median :  2    Median :153    Median :  3   Median :  1   Median : 21
##  Mean   : 15    Mean   :217    Mean   : 17   Mean   :  9   Mean   : 67
##  3rd Qu.:  4    3rd Qu.:256    3rd Qu.:  4   3rd Qu.:  1   3rd Qu.: 56
```

```
##  Max.   :977    Max.   :992    Max.   :988    Max.   :950    Max.   :998
##  NA's   :1415   NA's   :3121   NA's   :971    NA's   :798    NA's   :2272
```

We can see that there are large number of NA values for some attributes. The reason is that each simulation has been running for some unique time step, therefore, all attributes, except for the one with the largest number of time steps, have `NA` values. Before going further in dealing with the `NA` values, let's take a look at the correlation between the attributes using the `chart.Correlation()` function from `PerformanceAnalytics` library. Here, few random attributes are selected.

```
set.seed(123245)
suppressMessages(suppressWarnings(library(PerformanceAnalytics)))
chart.Correlation((gel.data[c(sample(50,5))]), histogram=TRUE, pch=19)
```

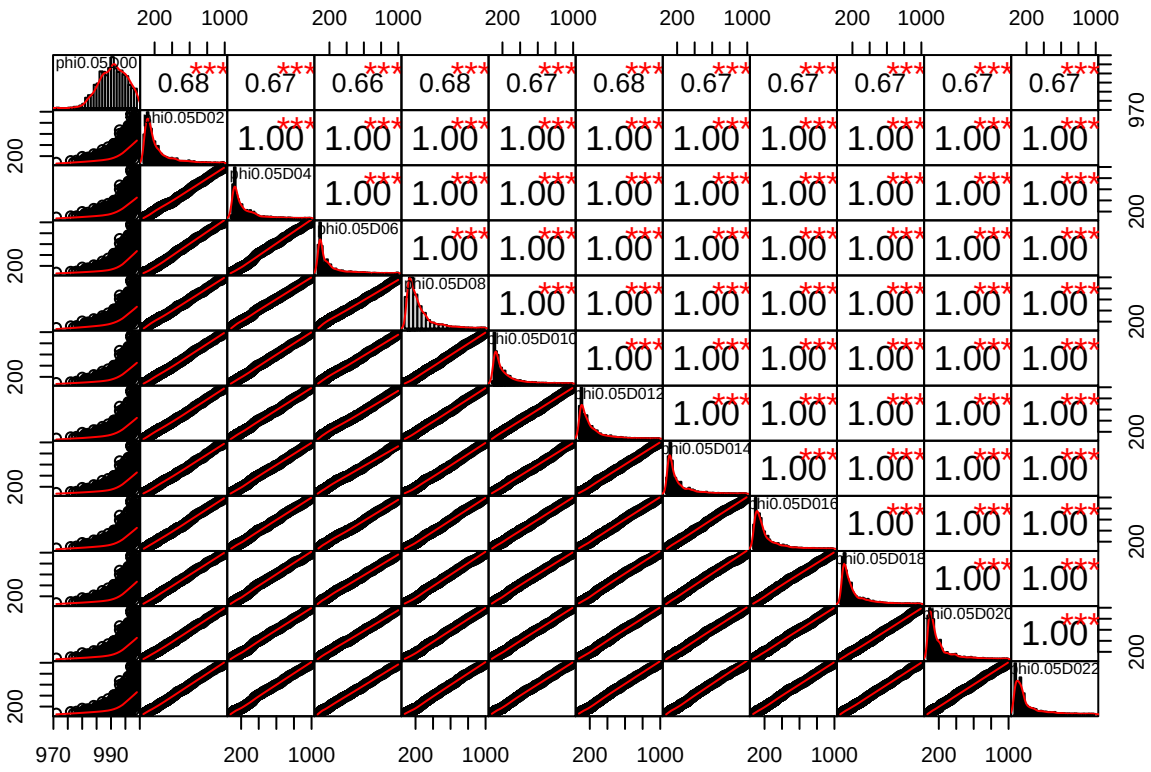


There are two important things seen in the data:

1. There are cases where show a more gussian distribution and are less correlated with the others. (It's happening at $D_0 = 0$)
2. Some of them are exponentially decreasing and are highly correlated. Although, we shouldn't let these correlations fool us, because for some simulations, the plots are mostly in a plateau state. From this point on, this is something we must have an eye on.

Let's plot simulations of the same ϕ and see if it gives us anything:

```
set.seed(12345)
suppressMessages(suppressWarnings(library(PerformanceAnalytics)))
chart.Correlation((gel.data[c(1:12)]), histogram=TRUE, pch=19)
```

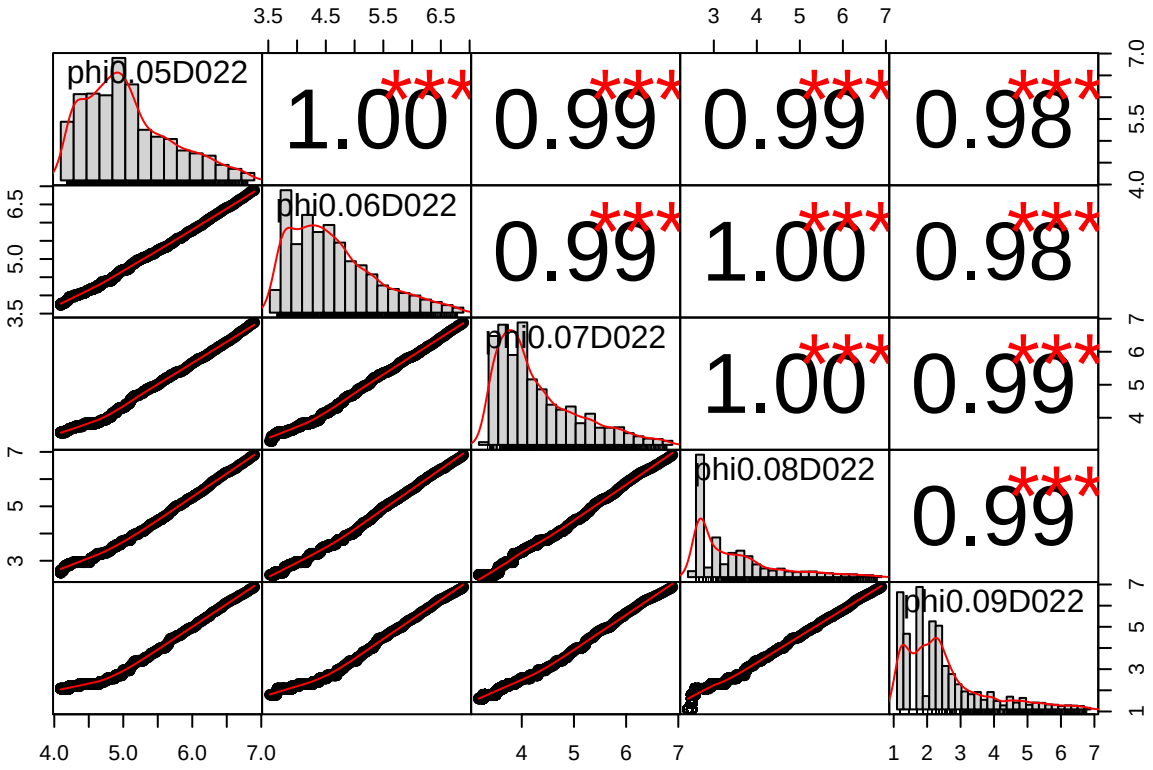


Again, these high correlations are very misleading, what we care about, is the exponential decay that happens to happen at the first steps Interestingly, the case with $D_0 = 0$ has a completely different behavior than the others. Dynamically speaking, it makes sense because the attraction changes everything and we expect no gelation without it. So, we don't really need this, let's remove the simulations with no attraction from data.

```
D0_cols = c()
counter = 0
for (i in c(1:(ncol(gel.data)-1))){
  temp.D0 = extract_values(names(gel.data)[i])[2]
    if (temp.D0==0){
      counter = counter + 1
      D0_cols[counter] = i
    }
}
gel.data = gel.data[,-D0_cols]
```

Now, let's take one attraction, and look at the correlations of different volume fractions. Also take the logarithm of the data to make them less skewed.

```
chart.Correlation(log(gel.data[(c(0:4)*11 + 11)]), histogram=TRUE, pch=19)
```



Take a closer look at those correlation lines, in fact, what we care about, are those initial times where the correlations are not linear. The linear parts of the correlations actually correspond to the steady state and do not count because we care about the decaying process. To avoid a waste of computational size and misleading correlations, for each simulation, the data is trimmed where the number of clusters becomes smaller than 5 (using $0.005N_{total}$ as the exponential relaxation cut-off).

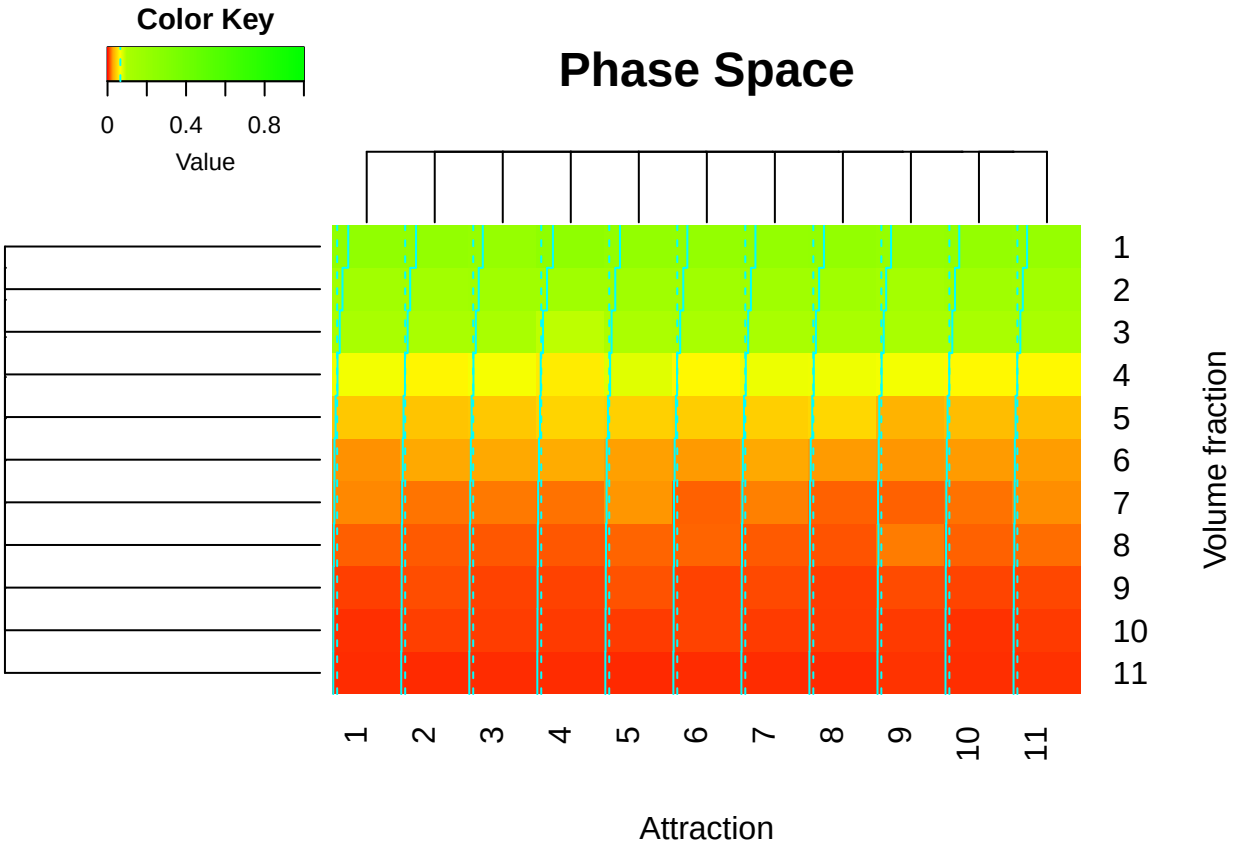
- The data we need for the phase space is extracted before trimming.

```
# loop over all attributes
max.timestep = 0; #
for (sim in c(1:(ncol(gel.data) -1 ) )){
  cut_off_vals = which(gel.data[,sim] > 5)
  tmp.phi = extract_values(names(gel.data)[sim])[1]
  tmp.D0 = extract_values(names(gel.data)[sim])[2]
  last.num = max(which(is.na(gel.data[,sim]) == FALSE))
  phase_space[which(phi_list==tmp.phi), (which(D0_list==tmp.D0)+1)] =
    mean(gel.data[c(last.num-100:last.num),sim],na.rm = TRUE)
  if (length(cut_off_vals) > 0){
      if(max(cut_off_vals)>max.timestep) {max.timestep =  max(cut_off_vals)}
    gel.data[-cut_off_vals,sim] = NA
  }
}
phase_space[,-1] = phase_space[,-1]/1000
```

Phase Space

Before going further, plot the phase space. We need to plot the final state of the mixture with respect to the system configuration.

```
suppressMessages(suppressWarnings(library(gplots)))
my_palette <- colorRampPalette(c("red", "yellow", "green"))(n = 299)
col_breaks = c(seq(0,.03,length=100),  # for red
               seq(0.031,0.1,length=100),           # for yellow
               seq(0.11,1,length=100))              # for green
heatmap.2(phase_space[,-1],main = "Phase Space",
          notecol=NULL,density.info="none",  col=my_palette,
          breaks=col_breaks,Colv="NA", Rowv="NA",
          xlab = "Attraction", ylab= "Volume fraction")
```



Data Cleaning & Shaping

data imputation

The `NA` values are untouched for these reasons:

- We don't want to substitute them with anything because we need to predict them correctly to do so, which defeats the purpose here.
- We don't want to remove them from data frame because other attributes will lose precious values.
- These are benign `NA` values that start existing at a point of time for each attribute and are not assigned up to the end of that simulation. It's easier to work with them if they're just there.

Normalization of feature values

Let's normalize the number of clusters with the possible maximum number of clusters $N = 1000$. Don't normalize the `timestep` attribute.

```
#gel.data[,-ncol(gel.data)] = gel.data[,-ncol(gel.data)] / 1000
```

Model Construction and Evaluation

Training and Validation subsets

80% of the simulations are selected as the training sets and the other are used for validation.

```
tr.indc = sample(ncol(gel.data[-1]), floor(0.8 * ncol(gel.data)))
gel.tr1 = gel.data[tr.indc]
gel.vl1 = gel.data[-tr.indc]
```

Linear regression

Make a linear regression to model the gel relaxation over time for all simulations. Each gel relaxes exponentially: $N(t) = N_0 e^{(-t/\tau)} \therefore log(N(t)) = log(N_0) - t/\tau$ Therefore, what we feed to the linear regression is: $y = ax + b$ where $y \equiv log(N(t))$, $a \equiv log(N_0)$ and $b \equiv -1/\tau$.

```
relaxation_matrix = matrix(0, nrow = length(phi_list), ncol = length(D0_list))
relaxation_matrix2 = matrix(0, nrow = length(phi_list), ncol = length(D0_list))
R_sq = matrix(0, nrow = length(phi_list), ncol = length(D0_list))
#gel.data = gel.data[-c(1:10),]
set.seed(12345)
for (i in c(1:(ncol(gel.data)-1))){
  # y = a * exp(- t / tau) = a * exp( b * t), where b = 1/tau
  # log(y) = log(a) + b * t
  train_rows = which(gel.data[,i] > 200)
  lm1 = lm(I(log(gel.data[train_rows,i]) - 0) ~  I(timestep[train_rows] - log(1000)), data = gel.data)#
  tmp.phi = extract_values(names(gel.data)[i])[1]
  tmp.D0 = extract_values(names(gel.data)[i])[2]
  row.ind = which(phi_list==tmp.phi); col.ind = which(D0_list==tmp.D0)
  relaxation_matrix[row.ind, col.ind] = -1/lm1$coefficients[2]
  relaxation_matrix2[row.ind, col.ind] = lm1$coefficients[1]
  R_sq[row.ind, col.ind] = summary(lm1)$r.squared
}
```

first, let's see what is the minimu R-squared value for all the models;

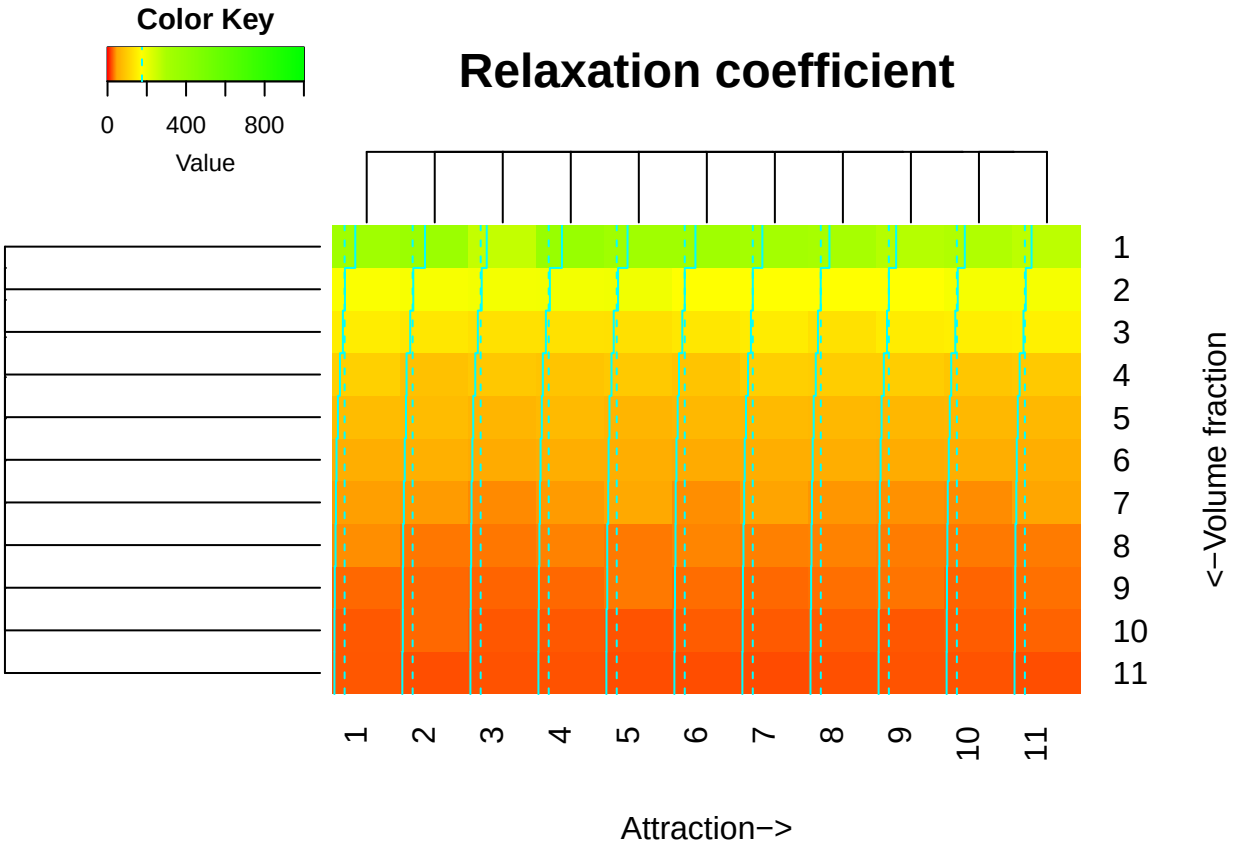
```
min(R_sq)
```

```
## [1] 0.923
```

Well, this means the model with least accuracy, still is a very good approximation of the actual data. Although, we must still be aware that this might be becuase of the skewed nature of our data, some tuning will be discusssed later.

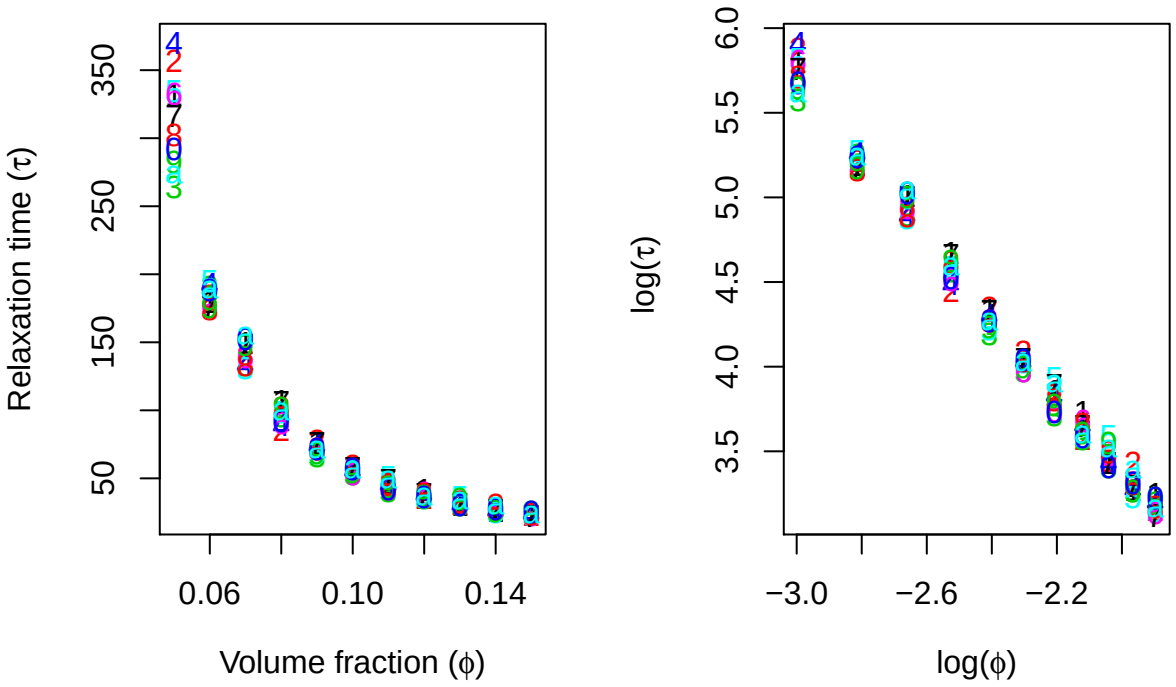
Now, let's take a look at the heatmap of the calculated relaxation times,

```
my_palette <- colorRampPalette(c("red", "yellow", "green"))(n = 299)
col_breaks = c(seq(0,50,length=100),  # for red
               seq(51,300,length=100),            # for yellow
               seq(301,1000,length=100))              # for green
heatmap.2(relaxation_matrix,main = "Relaxation coefficient",
          notecol=NULL,density.info="none",  col=my_palette,
          breaks=col_breaks,Colv="NA", Rowv="NA",
          xlab = "Attraction->", ylab= "<-Volume fraction")
```



We can clearly see that the relaxation time decreases from top to bottom, this implies that the gels with higher volume fraction, happen to evolve faster. This is consistent with what we already observed. Yet there is no specific pattern observed in "attraction" direction, seems like in this specific regime, attraction is not playing a major role in gelation process. Let's see how volume fraction is affecting the relaxation time for different simulations, for this purpose, group the simulations with same attraction together, now plot each group's relaxation time w.r.t. volume fraction.

```
par(mfrow=c(1,2))
matplot(x=phi_list, y=(relaxation_matrix), type = "p", lty = 1:5, lwd = 1, pch = NULL,
        col = 1:6, cex = NULL, bg = NA,
        xlab = expression(paste("Volume fraction (",phi,")")),
        ylab =expression(paste("Relaxation time (",tau,")")),
        xlim = NULL, ylim = NULL)
matplot(x=log(phi_list), y=log(relaxation_matrix), type = "p", lty = 1:5, lwd = 1, pch = NULL,
        col = 1:6, cex = NULL, bg = NA,
        xlab = expression(paste("log(",phi,")")),
        ylab =expression(paste(" log(",tau,")")),
        xlim = NULL, ylim = NULL)
```



Interestingly, all the curves collapse on each other almost perfectly, showing that simulations of different attractions are dynamically same. You can also see in the second figure that, the log of relaxation time changes linearly with time. Now, lets use linear regression to find the correlation between volume fraction and relaxation time. Here's the mathematical grooming of the equations: $\tau = a_0 e^{-\phi/\phi_0} \therefore log(\tau) = log(a_0) - \phi/\phi_0$ Therefore, this is what we feed to the linear regression model: $y = ax + b$, where $y \equiv log(\tau)$, $a = log(a_0)$ and $b \equiv -1/\phi_0$.

```
tau_phi = matrix(0, nrow = length(phi_list), ncol = 5)
tau_phi[,1] = phi_list
set.seed(12345)
for (i in c(1:(ncol(relaxation_matrix)))){
  lm2 = lm(log(relaxation_matrix[,i])~I(phi_list - 0))
  tau_phi[i,2] = -1/lm2$coefficients[2]
  tau_phi[i,3] = exp(lm2$coefficients[1])
  tau_phi[,4] = mean(relaxation_matrix[i,])
}
tau_phi[,5] = tau_phi[,3] * exp(-tau_phi[,1] / tau_phi[,2])
```

So, till now, we learned that $N(t) = N_0 e^{-t/\tau}$ where $N_0 = 1000$, i.e the total number of particles and τ is a function of ϕ such that $\tau(\phi) = \tau_0 e^{-phi/\phi_0}$ where :

```
N_0 = 1000
tau_0 = mean(tau_phi[i,3])
tau_0
```

```
## [1] 756
```

and :

```
phi_0 = mean(tau_phi[i,2])
phi_0
```

```
## [1] 0.0414
```

Therefore, we can write the relaxation as : $N(t) = N_0 e^{-t/(\tau_0 e^{-phi/phi_0})}$.

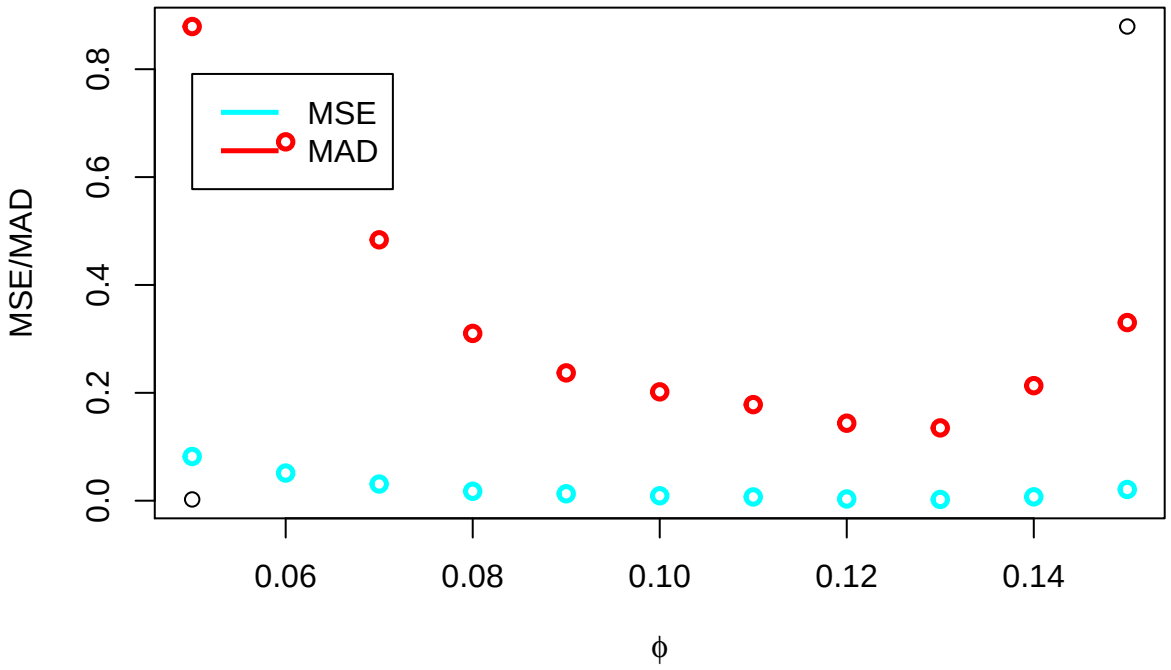
Model Evaluation

In this part, **MSE** and **MAD** are used to calculate the accuracy of our model, for different simulations.

```
MSE = rep(0, length(phi_list))
MAD = rep(0, length(phi_list))
for (sim in c(1:ncol(gel.data[-1]))){
  tmp.phi = extract_values(names(gel.data)[sim])[1]
  n.pred = length(which(is.na(gel.data[,sim]) == FALSE))
  actual = gel.data[c(1:n.pred),sim]
  i = which(phi_list == tmp.phi)
  Time = gel.data$timestep[c(1:n.pred)]
  pred = N_0 * exp(-Time / (tau_0 * exp(-tmp.phi / phi_0))) + 1
  MSE[i] = MSE[i] + mean((pred/1000 - actual/1000)^2,na.rm = TRUE)
  MAD[i] = MAD[i] + mean(abs(pred/1000 - actual/1000),na.rm = TRUE)
}
```

After calculating the errors, plot them for different volume fractions:

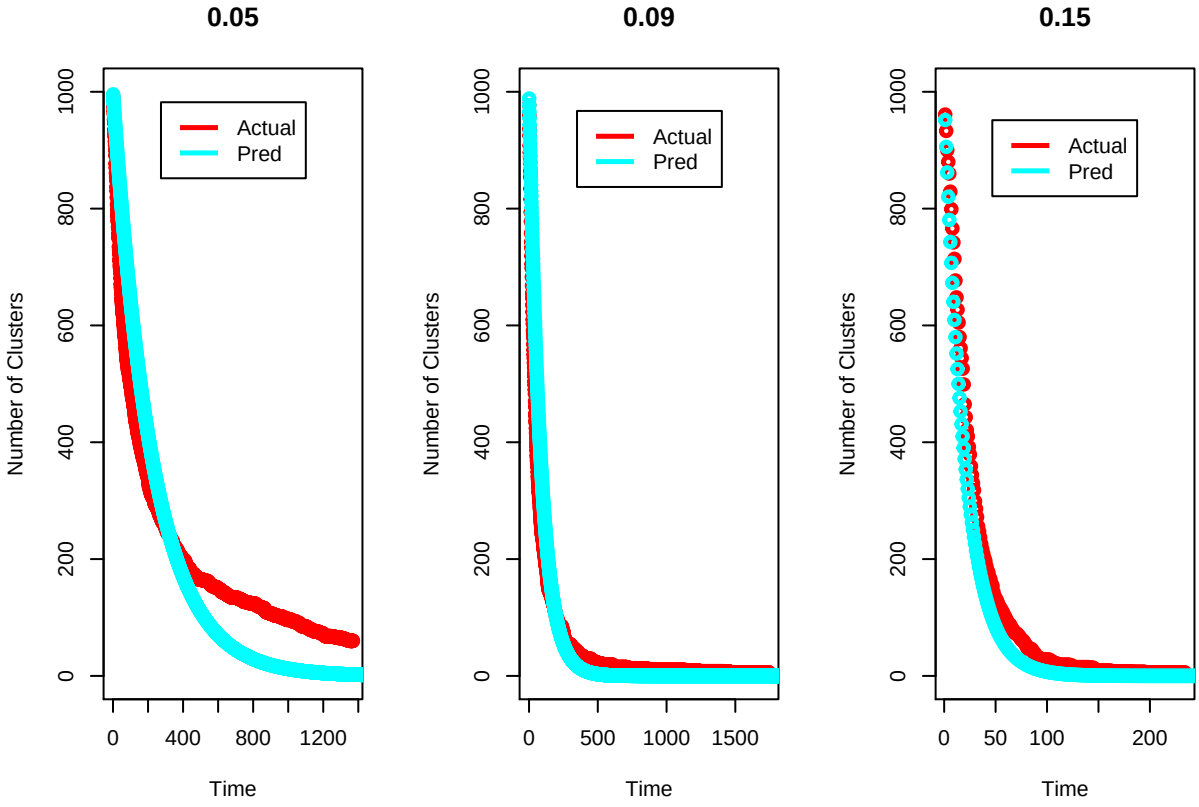
```
min_x = min(phi_list);max_x = max(phi_list)
min_y = min(MAD,MSE);max_y = max(MAD,MSE)
plot(c(min_x,max_x),c(min_y,max_y),xlab = expression(phi),ylab = "MSE/MAD")
points(phi_list,MSE,col="cyan",lwd=2.5)
points(phi_list,MAD,col="red",lwd=2.5)
legend(min_x,0.9*max_y, c("MSE","MAD"),
                          lty=c(1,1), lwd=c(2.5,2.5),col=c("cyan","red"))
```



This plot suggests that **MSE** is always in a good range of error but **MAD** increases at the start and end of the volume fraction window, Let's plot some of our actual data versus predictions to see why this is happening:

```
par(mfrow=c(1,3)); Time = gel.data$timestep
for (i in c(1,5,11)){
  target = gel.data[,(i*11)]; n.m = length(which(is.na(target)==FALSE))
```

```
  phi.t = extract_values(names(gel.data)[i*11])[1]
  pred = N_0 * exp(-Time / (tau_0 * exp(-phi.t / phi_0)))
  min_x = min(Time); max_x = n.m;
  min_y = min(target, na.rm = TRUE);max_y = max(target, na.rm = TRUE);
  plot(c(min_x,n.m),c(0,1000),type="n"
      , xlab = "Time",ylab = "Number of Clusters",
     main = phi.t)
  points(Time,target,col="red",lwd=2.5)
  points(Time,pred,col="cyan",lwd=2.5)
  legend(0.2*max_x,0.99*max_y, c("Actual","Pred"),
                           lty=c(1,1), lwd=c(2.5,2.5),col=c("red","cyan"))
}
```



This figure shows why the error is so large for the low volume fractions, the reason is that the regression model tries to fit on as many data points as possible and in this problem, relaxation is quick which makes the model converge to zero faster than it shoud. Therefore, we need to further tune the data frame and model, in order to get better results. Lets train the model again, this time, start the learning after the very sharp initial steps. (Start training set when the number of clusters drops below 100)

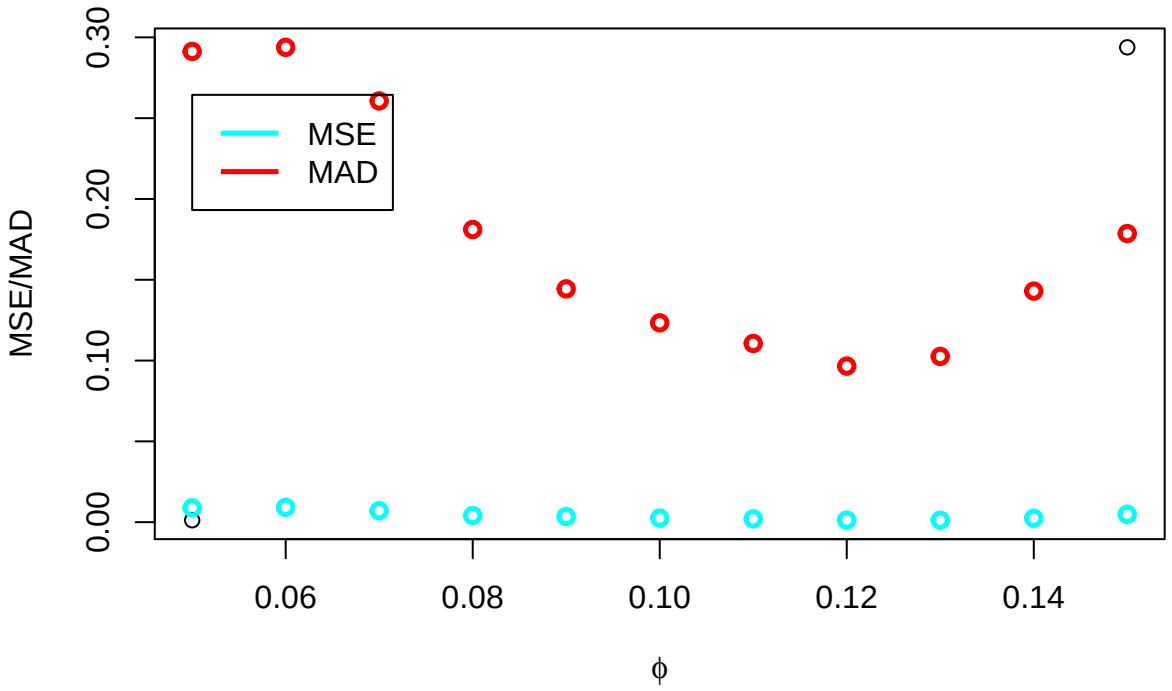
```
new.tr = 100
min.tr = 5
relaxation_matrix = matrix(0, nrow = length(phi_list), ncol = length(D0_list))
relaxation_matrix2 = matrix(0, nrow = length(phi_list), ncol = length(D0_list))
R_sq = matrix(0, nrow = length(phi_list), ncol = length(D0_list))
#gel.data = gel.data[-c(1:10),]
set.seed(12345)
for (i in c(1:(ncol(gel.data)-1))){
  train_rows = which(gel.data[,i] > new.tr);
  train_rows = train_rows[-c(1:min.tr)]
```

```
  lm1 = lm(I(log(gel.data[train_rows,i]) - 0) ~  I(timestep[train_rows] - log(1000)), data = gel.data)#
  tmp.phi = extract_values(names(gel.data)[i])[1]
  tmp.D0 = extract_values(names(gel.data)[i])[2]
  row.ind = which(phi_list==tmp.phi); col.ind = which(D0_list==tmp.D0)
  relaxation_matrix[row.ind, col.ind] = -1/lm1$coefficients[2]
  relaxation_matrix2[row.ind, col.ind] = lm1$coefficients[1]
  R_sq[row.ind, col.ind] = summary(lm1)$r.squared
}

tau_phi = matrix(0, nrow = length(phi_list), ncol = 5)
tau_phi[,1] = phi_list
set.seed(12345)
for (i in c(1:(ncol(relaxation_matrix)))){
  lm2 = lm(log(relaxation_matrix[,i])~I(phi_list - 0))
  tau_phi[i,2] = -1/lm2$coefficients[2]
  tau_phi[i,3] = exp(lm2$coefficients[1])
  tau_phi[,4] = mean(relaxation_matrix[i,])
}
tau_phi[,5] = tau_phi[,3] * exp(-tau_phi[,1] / tau_phi[,2])
N_0 = 1000
tau_0 = mean(tau_phi[i,3])
phi_0 = mean(tau_phi[i,2])

MSE = rep(0, length(phi_list))
MAD = rep(0, length(phi_list))
for (sim in c(1:ncol(gel.data[-1]))){
  tmp.phi = extract_values(names(gel.data)[sim])[1]
  n.start = length(which(gel.data[,sim] > new.tr))
  n.pred = length(which(is.na(gel.data[,sim]) == FALSE))
  actual = gel.data[c(n.start:n.pred),sim]
  i = which(phi_list == tmp.phi)
  Time = gel.data$timestep[c(n.start:n.pred)]
  pred = N_0 * exp(-Time / (tau_0 * exp(-tmp.phi / phi_0))) + 1
  MSE[i] = MSE[i] + mean((pred/1000 - actual/1000)^2,na.rm = TRUE)
  MAD[i] = MAD[i] + mean(abs(pred/1000 - actual/1000),na.rm = TRUE)
}

min_x = min(phi_list);max_x = max(phi_list)
min_y = min(MAD,MSE);max_y = max(MAD,MSE)
plot(c(min_x,max_x),c(min_y,max_y),xlab = expression(phi),ylab = "MSE/MAD")
points(phi_list,MSE,col="cyan",lwd=2.5)
points(phi_list,MAD,col="red",lwd=2.5)
legend(min_x,0.9*max_y, c("MSE","MAD"),
                          lty=c(1,1), lwd=c(2.5,2.5),col=c("cyan","red"))
```



After tuning the model, we can see that the error values are significantly improved now. The reason is that with the tuning we did, there is a better fit at the initial steps as well as trivial/steady state later steps.

Conclustion of the first part: At this specific regime, there is no effect of attraction on the evolution process of gelation. To study the effects of volume fraction, the relaxation times were first calculated and the exponential decay of relaxation time with volume fraction was alsso modeled. Therefore, one can predict the time it takes for his gel to prepare if they know their mixture properties.

In an effort to expand the work on a dynamically more complicated system, a specific regime was simulated where the gelation evolution of the matter is a result of an interplay between volume fraction and interparticle attraction. Let's start the second part:

Second part, getting attraction involved

In the first part of the project, the interparticle attraction does not play an important role. In this part, new simulations are done to get attraction involved in the phase space. This time means, we need a more complicated model. Following steps are done:

- Using `svm` to classify simulations as gel/not gel using the ϕ and D_0 features.
- Using `nn` for the same purpose
- Using Multiple linear regression to decide the time evolution of the matter and the final state
- Compare results of different models

Data Acquisition

Read the new data:

```
gel.data2 = read.csv('/Users/nabi_137/DA5030/Project/RMD/N_Clusters2.csv', header = TRUE)
```

Initiallize list of simulation features for future use:

```
phi_list = c(0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6)
DO_list = c(1,4,7,10,13,16,19,22)
```

Data Exploration

Use `summary()` function to observe 10 random attributes:

```
summary(gel.data2[,sample(ncol(gel.data2),10)])
```

```
##    phi0.1D016    phi0.35D07    phi0.5D022      phi0.6D04      phi0.6D010
##  Min.   :178   Min.   : 29   Min.   : 1.00   Min.   : 1.0   Min.   :1.00
##  1st Qu.:229   1st Qu.:158   1st Qu.: 1.00   1st Qu.: 1.0   1st Qu.:1.00
##  Median :259   Median :168   Median : 1.00   Median : 1.0   Median :1.00
##  Mean   :275   Mean   :168   Mean   : 1.01   Mean   : 1.1   Mean   :1.26
##  3rd Qu.:315   3rd Qu.:178   3rd Qu.: 1.00   3rd Qu.: 1.0   3rd Qu.:1.00
##  Max.   :918   Max.   :221   Max.   :20.00   Max.   :11.0   Max.   :8.00
##   phi0.15D022    phi0.4D04     phi0.5D07      phi0.55D022   phi0.05D04
##  Min.   :  1   Min.   : 22   Min.   : 1.00   Min.   :1    Min.   :946
##  1st Qu.:  1   1st Qu.:174   1st Qu.: 5.00   1st Qu.:1    1st Qu.:963
##  Median :  3   Median :184   Median : 6.00   Median :1    Median :967
##  Mean   :  8   Mean   :184   Mean   : 6.39   Mean   :1    Mean   :966
##  3rd Qu.:  5   3rd Qu.:194   3rd Qu.: 8.00   3rd Qu.:1    3rd Qu.:970
##  Max.   :680   Max.   :234   Max.   :24.00   Max.   :9    Max.   :984
```

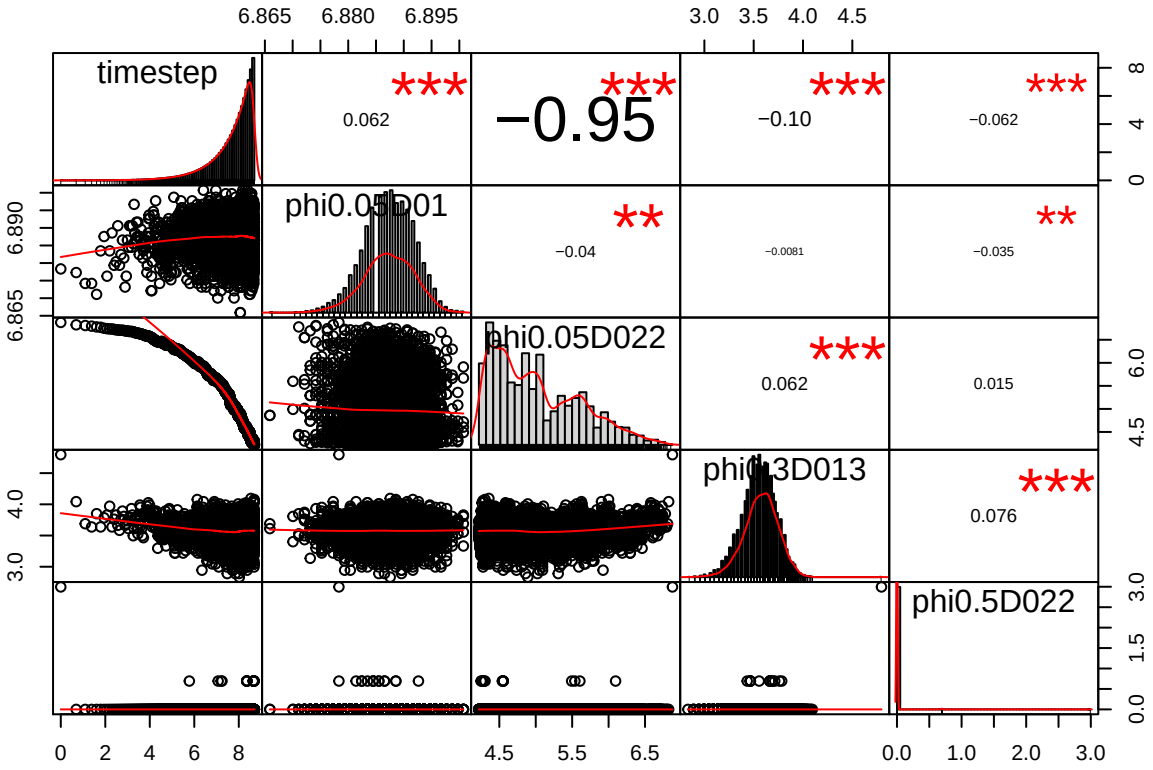
This time, the data set is already clean and there is no `NA` value, all attributes are of the same size. Let's make sure there is no `NA` value:

```
which(is.na(gel.data2) == TRUE)
```

```
## integer(0)
```

Great! Now, let's take a look at the correlatios:

```
set.seed(12345)
chart.Correlation(log(gel.data2[,c(97,1,8,45,80)]), histogram=TRUE, pch=19)
```



Clearly, now we're dealing with a more complicated system. There are several cases that must be causiosly treated. A simulation where no network is formed whatsoever ("phi0.05D01"), relaxation time is actually infinity here ($\tau = \infty$). A case where gradual relaxing happens ("phi0.05D022"), just like what we saw in part 1, and a case where relaxation happens so fast that it appears almost as a vertical line in the correlation plot ("phi0.05D022"). Make a data frame of the features and the class attribute for future use:

```
z = rep(0,(ncol(gel.data2)-1))
c.a = rep(FALSE,(ncol(gel.data2)-1))
phase.df = data.frame("phi"=z,"D_0"=z,"N_C"=z,"is.gel"=c.a,"svm"=c.a,"nn"=c.a,"regression"=c.a)
```

Phase of each simulation is calculated here, using the `mean()` of number of clusters at the last time steps.

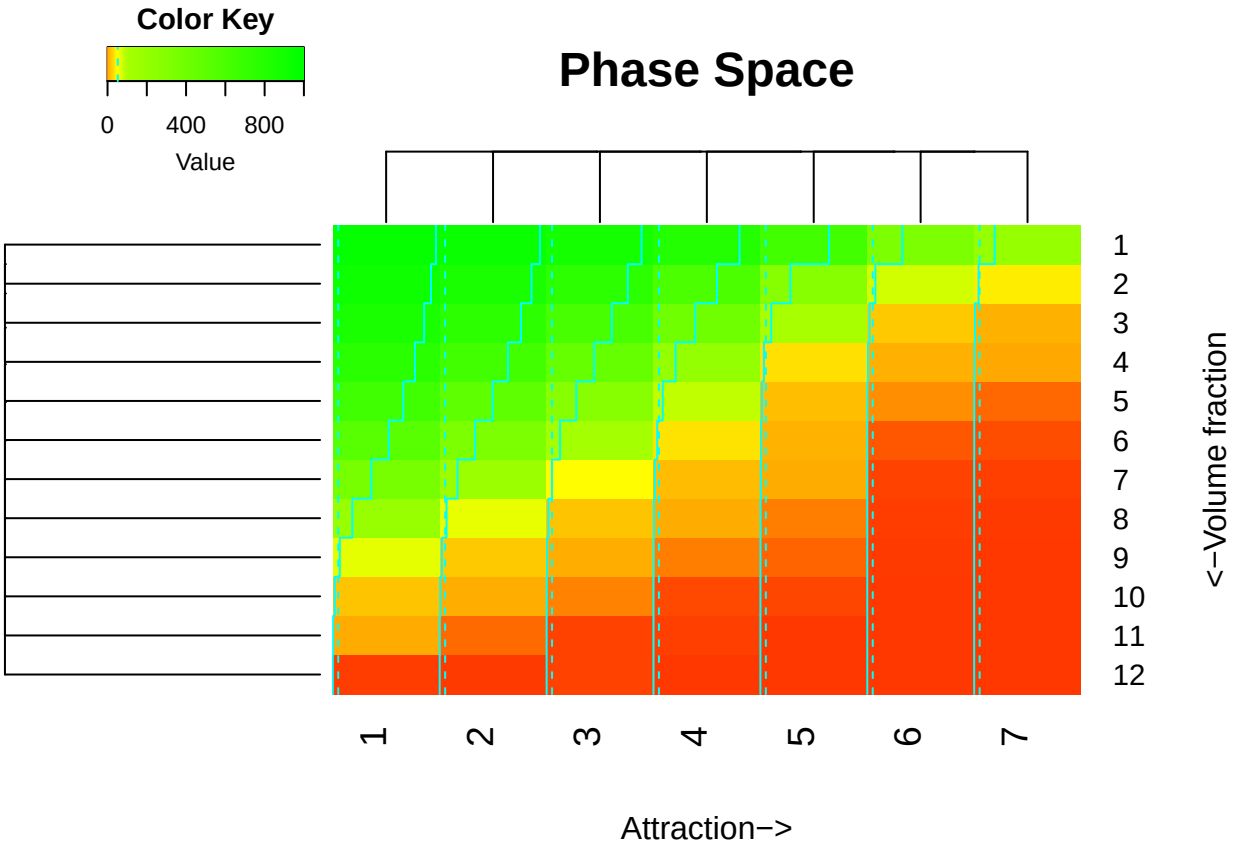
```
tr.indc = sample(ncol(gel.data2[-1]), floor(0.8 * ncol(gel.data2)))
gel.tr2 = gel.data2[tr.indc]
gel.vl2 = gel.data2[-tr.indc]
max.timestep = 0;
phase_space2 = matrix(0, nrow = length(phi_list), ncol = (length(D0_list)))
bool_phase_space = phase_space2
for (sim in c(1:(ncol(gel.data2) -1 ) )){
  cut_off_vals = which(gel.data2[,sim] > 0)
  tmp.phi = extract_values(names(gel.data2)[sim])[1]
  tmp.D0 = extract_values(names(gel.data2)[sim])[2]
  phase.df$phi[sim] = tmp.phi;phase.df$D_0[sim] = tmp.D0;
  last.num = max(which(is.na(gel.data2[,sim]) == FALSE))
  ps.x = which(phi_list==tmp.phi); ps.y=which(D0_list==tmp.D0)
  phase_space2[ps.x, ps.y] =
    mean(gel.data2[c(last.num-500:last.num),sim],na.rm = TRUE)
  phase.df$N_C[sim]=phase_space2[ps.x, ps.y]
  if (phase_space2[ps.x, ps.y] < 15){
    bool_phase_space[ps.x,ps.y] = 1
    phase.df$is.gel[sim]=TRUE
```

```
  }
  if (length(cut_off_vals) > 0){
      if(max(cut_off_vals)>max.timestep) {max.timestep =  max(cut_off_vals)}
    gel.data2[-cut_off_vals,sim] = NA
  }
}
```

Phase Space in presence of significant attractions

Before going further, plot the phase space. We need to plot the final state of the mixture with respect to the system configuration.

```
my_palette <- colorRampPalette(c("red", "yellow", "green"))(n = 299)
col_breaks = c(seq(0,3,length=100),  # for red
               seq(4,100,length=100),            # for yellow
               seq(101,1000,length=100))              # for green
heatmap.2(phase_space2[,-1],main = "Phase Space",
          notecol=NULL,density.info="none",  col=my_palette,
          breaks=col_breaks,Colv="NA", Rowv="NA",
          xlab = "Attraction->", ylab= "<-Volume fraction")
```



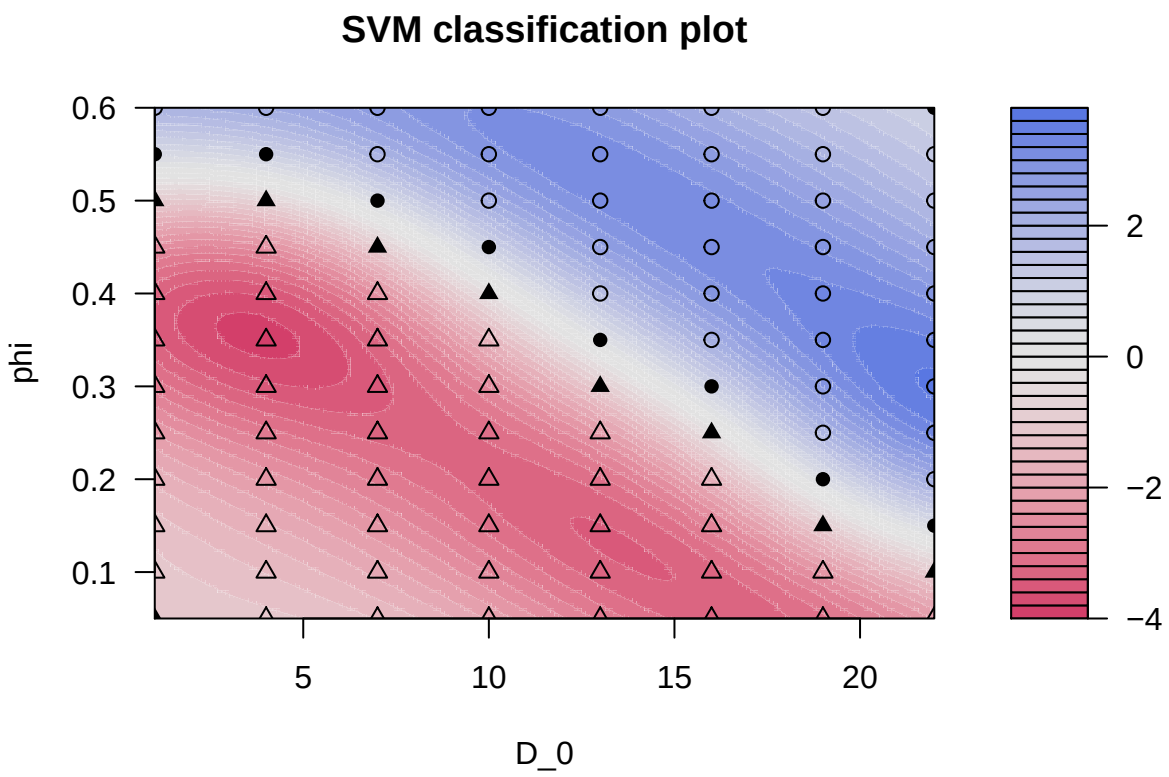
In this heatmap, red blocks represent a completely formed gel and green represents a liquild-like mixture. We can see that now both volume fraction and attraction are determining the phase space, actually, there are cases of the same volume fraction, where increase of attraction, solely, leads to gelation.

Model cunstruction

Training svm to phase space

Make a `svm` model to find the gelation boundary and plot it. Store the results of svm predictor so that we can compare the models.

```
set.seed(12345)
suppressMessages(suppressWarnings(library(kernlab)))
svm.m <- ksvm(is.gel ~ phi + D_0, data = phase.df, type="C-svc",C=10, kernel = "rbfdot")
plot(svm.m, data = phase.df, grid = 200 ,C = 10, kernel = "rbfdot")
```




```
phase.df$svm = ifelse(predict(svm.m,phase.df[,c(1,2)])==2, TRUE,FALSE)
```

Here, svm better shows us how non-linear the phase space becomes in higher volume fractions. Now, train a neural network on the data. In this case, the `rbfdot` kernel shows to be a good fit and a cost `10` is selected, although, changing the cost does not affect the results significantly. Models are evaluated and compared at the end of the work. Let's see how accurate our model is:

```
agreement_svm <- phase.df$svm == phase.df$is.gel
prop.table(table(agreement_svm))
```

```
## agreement_svm
## TRUE
##    1
```

100% accuracy! This is not much weird because of two reasons:

- The phase boundary is not much complicated
- Small data set (because of the huge trimming before even starting the work and high computational cost of the new simulations)

Training Neural Network to phase space

```
set.seed(12345)
suppressMessages(suppressWarnings(library(neuralnet)))
nn.m <- neuralnet(is.gel ~ phi + D_0, data = phase.df)
nn.result <- neuralnet::compute(nn.m, phase.df[,c(1,2)])
phase.df$nn <- ifelse(nn.result$net.result>0.5,TRUE,FALSE)
agreement_nn <- phase.df$nn == phase.df$is.gel
prop.table(table(agreement_nn))
```

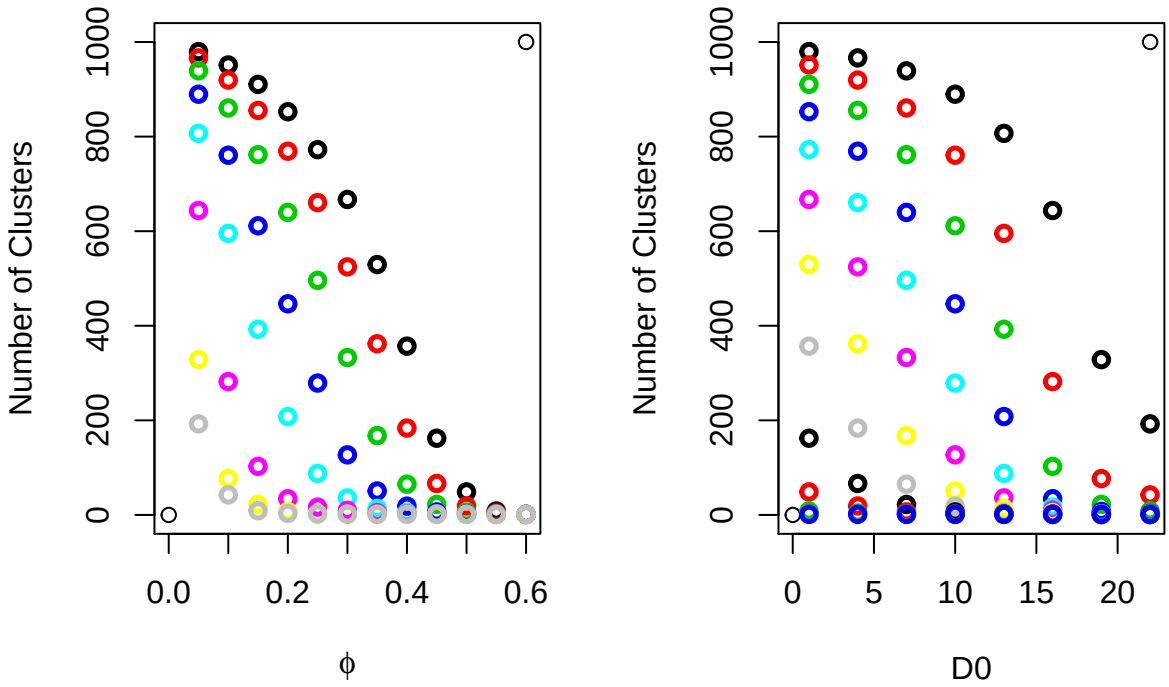
```
## agreement_nn
##  FALSE   TRUE
## 0.0208 0.9792
```

Neural network also does a great job predicting the boundary, it's not fair to say svm worked better because of the very small difference!

Linear regression

Here, we tend to make a model to find a correlation that relates number of clusters at each time, to ϕ and D_0. The desired equation will be in the form of: $N_C(t) = f(t, \phi, D_0)$, If we manage to pull this off, we can calculate the number of clusters at an arbitrary time and say if an interconnected network of particles (AKA gel) is formed. First, let's take a look at the correlation between final number of clusters and volume fraction/attraction seperately.

```
par(mfrow=c(1,2)); Time = gel.data2$timestep
min_x = 0;max_x = max(phi_list);min_y = 0;max_y = 1000
plot(c(min_x,max_x),c(min_y,max_y),xlab = expression(phi),ylab = "Number of Clusters")
for (i in c(1:length(D0_list))){
  indcs = c(1:length(phi_list)) * 8 - 8
  points(phi_list,phase.df$N_C[(indcs+i)],col=i,lwd=2.5)
}
par(new = FALSE)
plot(c(min_x,22),c(min_y,max_y),xlab = expression(D0),ylab = "Number of Clusters")
for (i in c(1:length(phi_list))){
  indcs = c(1:length(D0_list)) + (i - 1) * 8
  points(D0_list,phase.df$N_C[indcs],col=i,lwd=2.5)
}
```

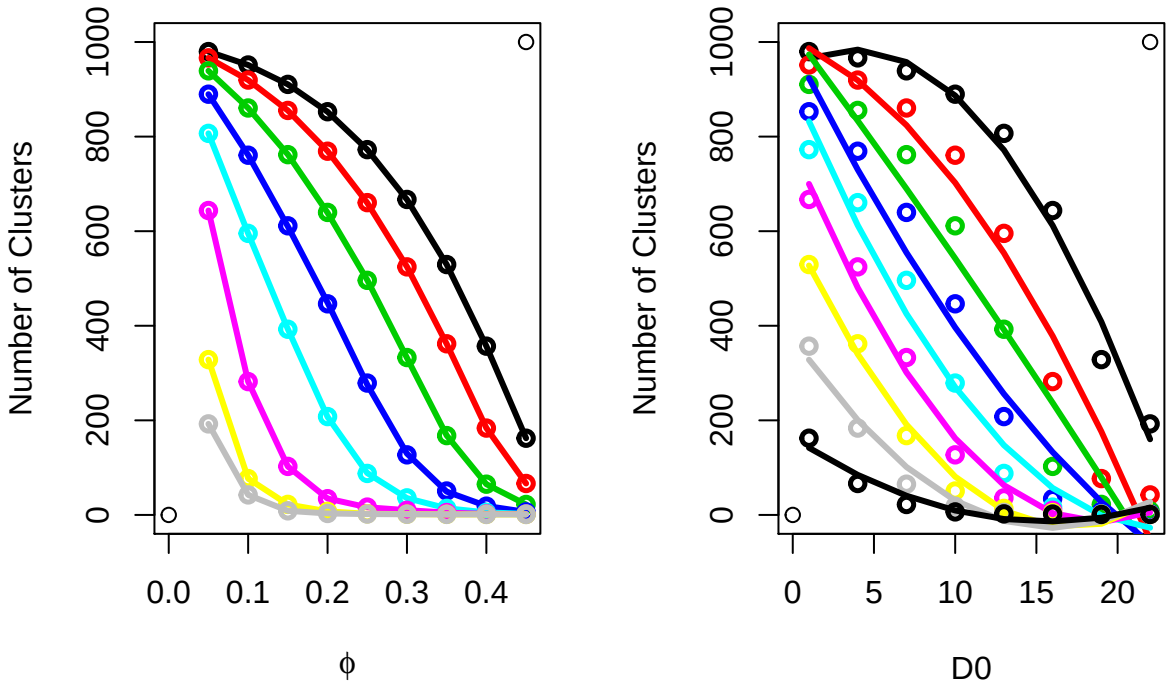


Before training the model, we must note that when $\phi > 0.45$ gelation always happens regardless of attraction; Therefore, we must ignore these in our regression model, to prevent errors.

```
to.rm = which(phase.df$phi > 0.5)
phase.df.2 = phase.df[-to.rm,]
```

In this part, we try to predict the final number of clusters using the given features.

```
phi_list = phi_list[-c(12,11,10)]
par(mfrow=c(1,2)); Time = gel.data2$timestep
min_x = 0;max_x = max(phi_list);min_y = 0;max_y = 1000
plot(c(min_x,max_x),c(min_y,max_y),xlab = expression(phi),ylab = "Number of Clusters")
for (i in c(1:length(D0_list))){
  indcs = c(1:length(phi_list)) * 8 - 8 + i
  points(phi_list,(phase.df.2$N_C[indcs]),col=i,lwd=2.5)
  model <- lm(phase.df$N_C[indcs] ~ poly(phi_list,7))
  new_data = phase.df$phi[indcs]
  predicted.intervals <- predict(model,data = new_data,interval='confidence',
                                 level=0.99)
  lines(phi_list,(predicted.intervals[,1]),col=i,lwd=3)
}
par(new = FALSE)
plot(c(min_x,22),c(min_y,max_y),xlab = expression(D0),ylab = "Number of Clusters")
for (i in c(1:length(phi_list))){
  indcs = c(1:length(D0_list)) + (i - 1) * 8
  points(D0_list,(phase.df.2$N_C[indcs]),col=i,lwd=2.5)
  model <- lm(phase.df$N_C[indcs] ~ poly(D0_list,2))
  new_data = phase.df$D_0[indcs]
  predicted.intervals <- predict(model,data = new_data,interval='confidence',
                                 level=0.9)
  lines(D0_list,(predicted.intervals[,1]),col=i,lwd=3)
}
```

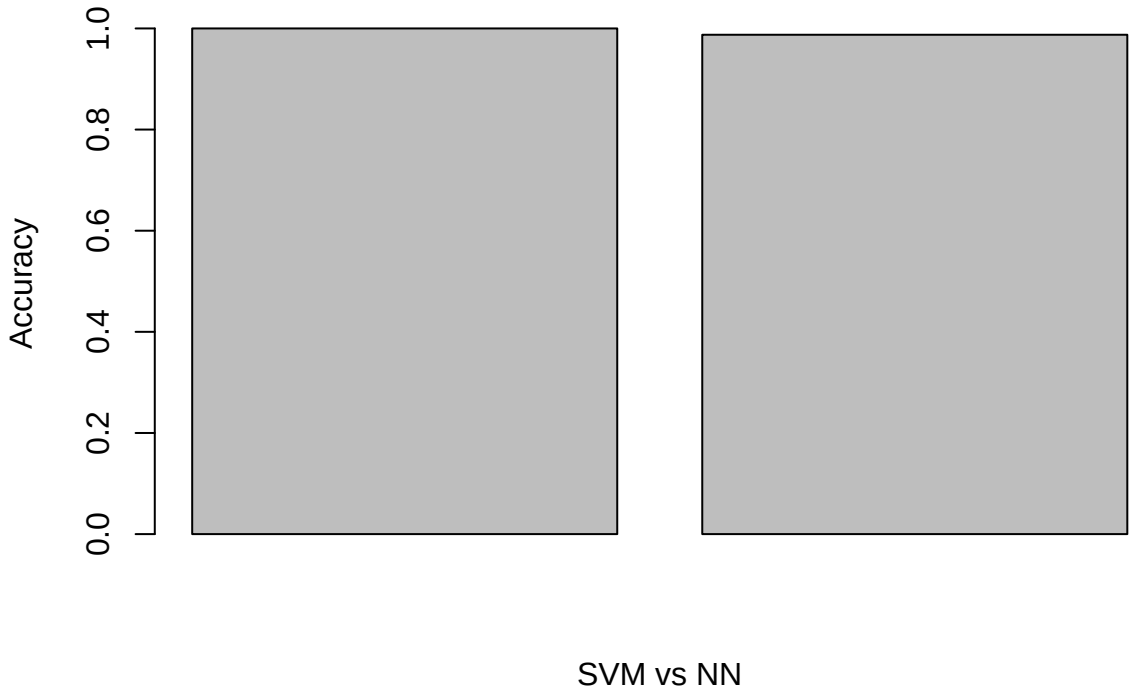


After several trial error processses, unfortunately, in this physical regime, our meaningful simulations are limited to a very narrow range of features, this makes our useful data much smaller than it already is, we need much more simulation data to make this work. This problem is left off at this point.

Compare the models

After failing to make the regression model to work for the second part, here, resuls of the **svm** and **neural network** are compared. The comparison is not of high significance as both models worked great for this problem. Although, svm always worked almost perfectly for this specific data while neural network was more sensitive to changes in the algorithm/data.

```
accuracy.df = data.frame("svm"=0,"nn"=0)
n.tot = nrow(phase.df.2)
accuracy.df$svm = length(which(phase.df.2$svm == phase.df.2$is.gel))/n.tot
accuracy.df$nn = length(which(phase.df.2$nn == phase.df.2$is.gel))/n.tot
barplot(c(accuracy.df$svm,accuracy.df$n),xlab = "SVM vs NN",ylab = "Accuracy")
```



Conclusion:

- Data mining and machine learning are great tools to dig deeper in science, find correlations, and predic the complicated behaviors we observe in nature
- High quality data is a very hard thing to get! As the system gets more complicated, the useful range of data gets narrower and narrower.
- smv and regression are such great models in an all numerical data set, combining the two can help build better models.