# Problem Set 2

```
import pandas as pd
import numpy as np
from scipy import stats
from scipy.stats import norm
from numpy.linalg import inv
import matplotlib.pyplot as plt
import os
from pathlib import Path
import re
import statistics as stat
import seaborn as sns

os.chdir('e:/MIT4/statistics-Computation/pset2')
```

## 2.1 Hi-C data analysis

```
def read_data(x, y):
        '''
        Read data file "chrX_chrY.txt" given X and Y,
        and return the pandas dataframe with column names
        '''
        file = 'data/hic'+'/chr'+str(x)+'_chr'+str(y)+'.txt'
        data = pd.read_csv(file, sep="  ", header=None)
        data.columns = ['Xloc','Yloc','IntFreq']
        data[['Xloc','Yloc']] = data[['Xloc','Yloc']]/250000
        return data
```

## (a)

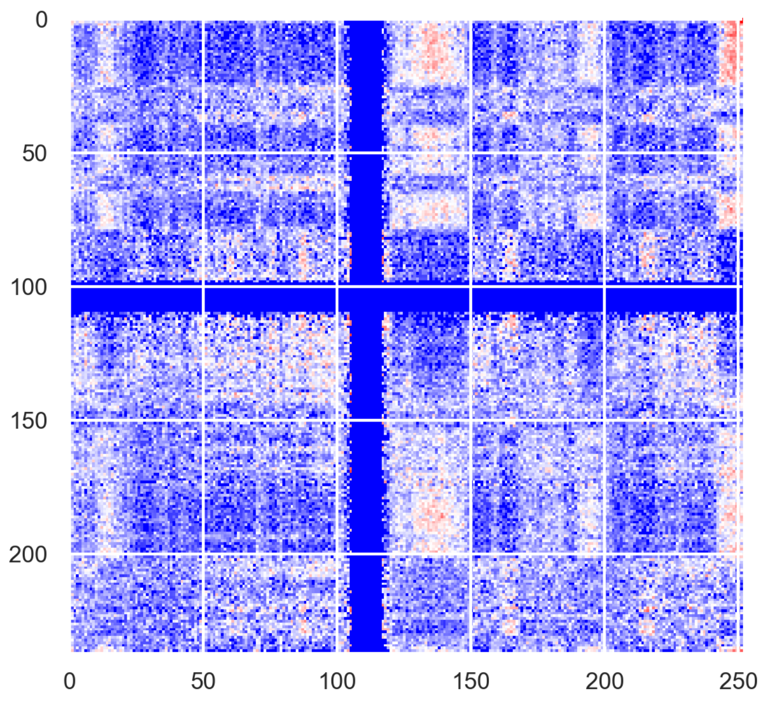
Compute the mean and standard deviation of log(1 + interaction frequency) across all inter-chromosome sites

```
destdir = Path('e:/MIT4/statistics-Computation/pset2/data/hic/')
# files = [p for p in destdir.iterdir() if p.is_file()]
# for f in files:
#       data = pd.read_csv(f, sep="     ", header=None).fillna(0)
# inters = []
n = 22 # the total number of chromosomes

def inter(x, y):
        '''
        Read data file "chrX_chrY.txt" given X and Y,
        and return the value IntFreq column
        '''
        file = 'data/hic'+'/chr'+str(x)+'_chr'+str(y)+'.txt'
        data = pd.read_csv(file, sep="  ", header=None).fillna(0)
        return data.iloc[:,2]

def inter_log(x, y):
        '''
        Read data file "chrX_chrY.txt" given X and Y,
        and return the log transformed IntFreq column
        '''
        file = 'data/hic'+'/chr'+str(x)+'_chr'+str(y)+'.txt'
        data = pd.read_csv(file, sep="  ", header=None).fillna(0)
        return np.log(1 + data.iloc[:,2])



sum_l = []
Nij_l = []
sd_l = []
for i in range(n):
        for j in range(n):
                if i != j:
                        try:
                                interaction = inter_log(i+1,j+1)
                                sum_l.append (interaction.sum())
                                Nij_l.append (interaction.shape[0])
                                sd_l.append (np.std(interaction))
                        except:
                                continue
sum(sum_l)/sum(Nij_l)

for i in range(n):
        for j in range(n):
                if i < j:
                        inters += ( np.log(inter(i+1,j+1) + 1)).tolist()

inters_array = np.array(inters)
inters_array.shape
mu_inters = np.nanmean(inters_array)

sigma_inters = np.nanstd(inters_array)

# mu_inters = 1.1311341591955062
# sigma_inters = 0.4019085024059505
```

## (b)

Next, we will look at interactions between chromosomes 19 and 20.To begin, plot a heat map of the interaction matrix (using the log(1+x) transformation). What do you see? Do you see any interacting regions?

```
df1920 = read_data(19, 20)
df1920.shape
df1920.head()

intmat_x = sorted(df1920['Xloc'].unique())
intmat_y = sorted(df1920['Yloc'].unique())

max(intmat_x)
max(intmat_y)

intmat = np.zeros((max(intmat_x)+1,max(intmat_y)+1))

for index, row in df1920.iterrows():
        intmat[int(row['Xloc'])][int(row['Yloc'])] = float(np.log(row['IntFreq']+1))

intmat.shape
np.savetxt("data/intmat.csv", intmat, delimiter=",")

intmat = pd.read_csv("data/intmat.csv", header = None)
intmat = np.matrix(intmat)
# Plot heatmap
plt.imshow(intmat, cmap='RdYlBu', interpolation='nearest')
plt.savefig('figure/heatmap.png')
```



## (c)

To identify regions with high interaction frequencies, we will perform a series of hypothesis tests...Explain where this formula comes from and compute the value of Nsubmatrices for the chromosome 19-20 interaction matrix.

# explanation of p-value

The null hypothesis is that each entry of the matrix is i.i.d Gaussian distributed (mu and sigma). Therefore, a k * l matrix, M, is a sample of size k * l, the test statistic is (m - mu)/SE. SE is for standard error, which equals to sigma/((k*l)^0.5). The p-value for M is 1 - norm.cdf((m - mu)/SE).

To correct the p-value for multiple tests on every submatrices, we use Holm-Bonferroni correction. For the sorted list of p-values for every test p_i, we reject null hypothesis when (N-i+1)p_i <= alpha, in which N equals to the total number of submatrices of M.

The min(p_i) of all tests of submatrices of M is the p_i for M itself because it has the minimum SE while the m and sigma stay the same according to the null hypothesis( norm.cdf() is an increasing function, so 1 - norm.cdf((m - mu)/SE) goes up when SE goes down). So (N-i+1)p_i = N * min(p_i)

# number of submatrices of intmat

```
def num_submat(matrix):
        # n_x, n_y = matrix.shape
        # return n_x * (1 + n_x) * n_y * (1 + n_y)/4
        return 899055234.0

num_submat(intmat)
np.nanmean(intmat)
```

## (d)

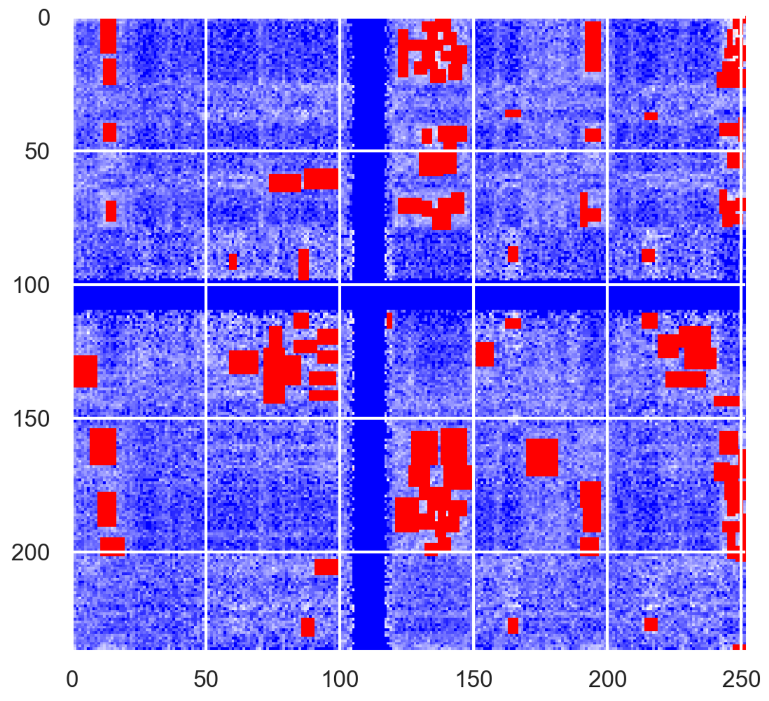
greedy search():

1. Randomly pick an entry in the interaction matrix. Compute the p-value of the 1 * 1 submatrix with this entry. We call this initial submatrix M.
2. Compute the p-value of the submatrix that consists of M joined with the column to the right of it. Repeat, but with M joined with the column to the left of M. Repeat again with M joined with the row above. Repeat once more with M joined with the row below.
3. If all four of these transformation led to increases in p-value, stop. Otherwise, proceed to step 4.
4. Choose which of the four transformations led to the smallest p-value and add the appropriate row/column to M. Return to step 2.

Our overall procedure to identify interaction regions in a given (transformed) interaction matrix, Z, is as follows.

1. Run greedy search() (multiple times) on Z to identify a submatrix with near-minimal p-value.
2. If the p-value of this submatrix is greater than 0:01, stop. Otherwise, proceed to step 3.
3. Store the identied submatrix as an interacting region. Subtract the mean of this submatrix from each entry of the submatrix in Z. Return to step 1 with this updated Z.

```
np.random.seed(0)

def compute_p(M, mu, sigma):
        m = np.nanmean(M)
        n_x, n_y = M.shape
        num = num_submat(M)
        return num * (1 - norm.cdf((m - mu) * (n_x * n_y)**0.5 /sigma))

# mat_test = intmat[:3,:3]
# compute_p(mat_test, mu, sigma)

class mat(object):
        def __init__(self, xmin, xmax, ymin, ymax, matrix):
                self.xmin, self.xmax, self.ymin, self.ymax = [xmin, xmax, ymin, ymax]
                self.matrix = matrix

        def getloc(self):
                xyloc = [self.xmin, self.xmax, self.ymin, self.ymax]
                return xyloc

        def getmat(self):
                return self.matrix[self.xmin:self.xmax+1 ,self.ymin:self.ymax+1]

        def getp(self, mu, sigma):
                return compute_p(self.getmat(), mu, sigma)

        def getmean(self):
                return np.nanmean(self.getmat())

def neighbor_mat(M, matrix, mu, sigma):
        N_X, N_Y = matrix.shape
        xmin, xmax, ymin, ymax = M.getloc()
        M_next = M
        p = M.getp(mu, sigma)
        if xmin >= 1:
                mat0 = mat(xmin - 1, xmax, ymin, ymax, matrix)
                p0 = mat0.getp(mu, sigma)
                if p0 < p:
                        M_next = mat0
                        p = p0

        if xmax <= N_X-2:
                mat0 = mat(xmin, xmax + 1, ymin, ymax, matrix)
                p0 = mat0.getp(mu, sigma)
                if p0 < p:
                        M_next = mat0
                        p = p0

        if ymin >= 1:
                mat0 = mat(xmin, xmax, ymin - 1, ymax, matrix)
                p0 = mat0.getp(mu, sigma)
                if p0 < p:
                        M_next = mat0
                        p = p0

        if ymax <= N_Y-2:
                mat0 = mat(xmin, xmax, ymin, ymax + 1, matrix)
                p0 = mat0.getp(mu, sigma)
                if p0 < p:
                        M_next = mat0
                        p = p0

        return M_next

def greedy_search(matrix, mu, sigma):

        N_X, N_Y = matrix.shape
        xmin = np.random.randint(N_X)
        ymin = np.random.randint(N_Y)
        xmax = xmin
        ymax = ymin

        M = mat(xmin, xmax, ymin, ymax, matrix)

        while neighbor_mat(M, matrix, mu, sigma) != M:
                M = neighbor_mat(M, matrix, mu, sigma)

        return M

def find_minmat(matrix, mu, sigma, num_iter):
        minmat = None
        p = np.inf
        for i in range(num_iter):
                M = greedy_search(matrix, mu, sigma)
                p_M = M.getp(mu, sigma)
                if p_M < p:
                        minmat = M
                        p = p_M

        return minmat

# minmat = find_minmat(intmat, mu_inters, sigma_inters, num_iter = 100)
#
# minmat.getmat()
# minmat.getp(mu_inters, sigma_inters)

def find_interaction(matrix, mu, sigma, num_iter):
        Z = matrix.copy()
        inters = []

        minmat = find_minmat(Z, mu, sigma, num_iter)

        while minmat.getp(mu, sigma) < 0.01:
                inters.append(minmat)
                xmin, xmax, ymin, ymax = minmat.getloc()
                Z[xmin:xmax+1, ymin:ymax+1] = Z[xmin:xmax+1, ymin:ymax+1] - minmat.getmean()
                minmat = find_minmat(Z, mu, sigma, num_iter)
        return inters

 interaction_zones = find_interaction(intmat, mu_inters, sigma_inters, num_iter = 500)

# len(interaction_zones)
# 500:121
# 100:86
# 50:59
intmat_highlight = intmat.copy()
k = 5
for zone in interaction_zones:
        xmin, xmax, ymin, ymax = zone.getloc()
        intmat_highlight[xmin:xmax+1, ymin:ymax+1] = np.ones(zone.getmat().shape)*k
# intmat.max()

heatmap = plt.imshow(intmat, cmap='bwr', interpolation='nearest')
fig_base = heatmap.get_figure()
fig_base.savefig('figure/heatmap1.png', dpi = 300)

highlight = plt.imshow(intmat_highlight, cmap='bwr', interpolation='nearest')

fig_hi = highlight.get_figure()
fig_hi.savefig('figure/heatmap2.png', dpi = 300)
```

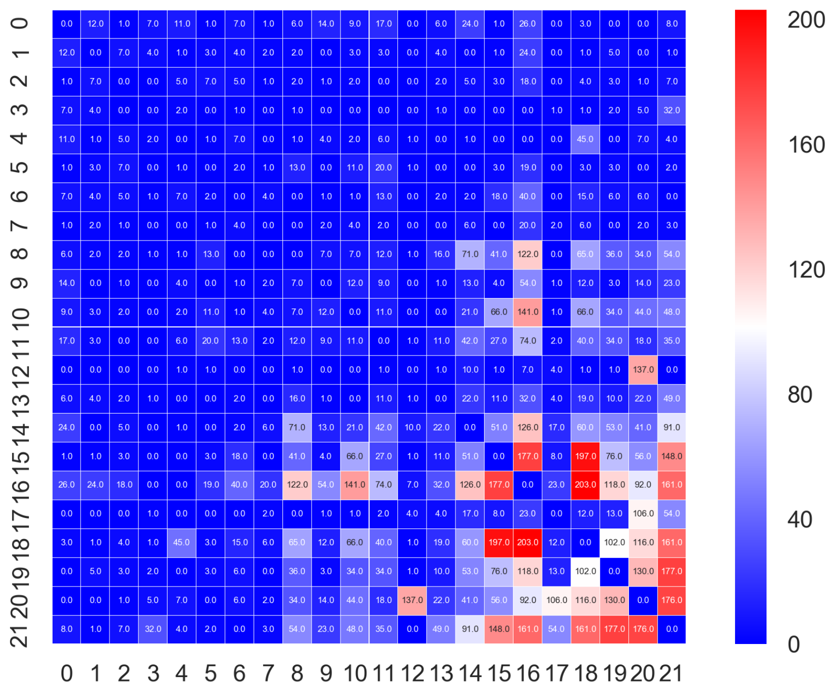


## (e)

Run the procedure you developed in part (d) on all pairs of chromosomes. Count the number of intermingling 250kb regions for each pair, i.e. the number of entries in the interaction matrix that are contained in any of the identied interaction regions. Plot a heat map of the inter-chromosome interaction counts (i.e. a 22 * 22 matrix with these interaction counts).

```
def build_intmat(x, y):
        df = read_data(x, y)
        intmat_x = sorted(df['Xloc'].unique())
        intmat_y = sorted(df['Yloc'].unique())
        intmat = np.zeros((max(intmat_x)+1,max(intmat_y)+1))
        for index, row in df.iterrows():
                intmat[int(row['Xloc'])][int(row['Yloc'])] = float(np.log(row['IntFreq']+1))
        return intmat


def intermingling(x, y, mu, sigma, num_iter = 500):
        intmat = build_intmat(x, y)
        interaction_zones = find_interaction(intmat, mu, sigma, num_iter)
        return len(interaction_zones)


n = 22
inter_count_mat = np.zeros([n, n])

for i in range(n):
        for j in range(n):
                if i < j:
                        try:
                                print(i+1 ,j+1)
                                inter_count_mat[i,j] = intermingling(i+1, j+1, mu_inters, sigma_inters)
                                inter_count_mat[j,i] = inter_count_mat[i,j]

                        except:
                                continue



inter_count_mat = inter_count_mat.astype(int)


sns.set(font_scale=0.8)
```

```
ax = sns.heatmap(inter_count_mat,cmap='bwr', annot=True, annot_kws={"size": 3},square=True, linewidths=0.01,fmt='.1f'
figure = ax.get_figure()
figure.savefig('figure/intermingling500.png', dpi=400)
```



## 2.2 Cell differentiation and gene expression

In this problem, we analyze single-cell RNA-seq data and determine structure in this high-dimensional data.

The data set consists of 272 cells, each row corresponds to the RNA-seq measurements of a particular gene.

Each entry corresponds to the normalized transcript compatibility count (TCC) of an equivalence class.

An equivalence class is a set of short RNA sequences. The TCC counts the number of reads of sequences which are compatible with each equivalence class for a given cell.

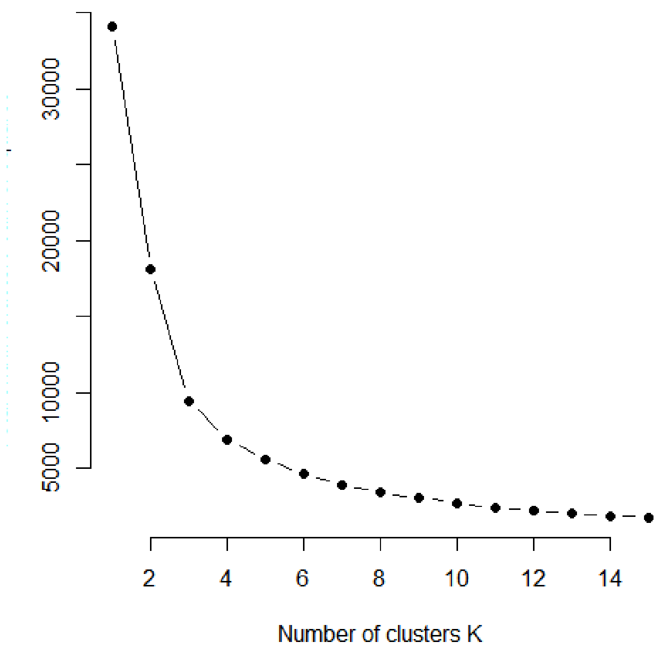
The entries have been normalized so that each row in the matrix sums to 1.

## (a)

Determine cell clusters by applying k-means clustering to the data. Hint: it may be helpful to first apply a dimension reduction method such as tSNE or PCA. This will help you determine the correct number of clusters to use and can speed up computations.

```
# The following part was done in R
library(data.table)
library(Rtsne)
library(plotly)
library(scatterplot3d)

setwd('e:/MIT4/6.439/pset2')
file = 'e:/MIT4/6.439/pset2/Trapnell/Trapnell.csv'
data = fread(file, sep = ",")

data_mat = as.matrix(data)
data_mat_t = t(data_mat)

# write.csv(data_mat_t, file = "Tra_t.csv")
```

```
tsne <- Rtsne(data_mat, dims = 3, perplexity=30, max_iter = 500)
# tsne$Y
# scatterplot3d(x=tsne$Y[,1],y=tsne$Y[,2],z=tsne$Y[,3], color = 'blue')

# clustering
data = data.frame(tsne$Y)

# decide the optimal number of clusters
#Elbow Method for finding the optimal number of clusters
set.seed(0)
# Compute and plot wss for k = 2 to k = 15.
k.max <- 15
wss <- sapply(1:k.max,
              function(k){kmeans(data, k, nstart=50,iter.max = 15 )$tot.withinss})
wss
plot(1:k.max, wss,
     type="b", pch = 19, frame = FALSE,
     xlab="Number of clusters K",
     ylab="Total within-clusters sum of squares")
```



```
# plot the clusters
clusters = kmeans(data, 10)
data$cluster10 = as.factor(clusters$cluster)
p <- plot_ly(data, x = ~X1, y = ~X2, z = ~X3, color = ~cluster10,
             marker = list(symbol = 'circle',
                           size = 3
                           #, colorscale = c('#FFE1A1', '#683531')
                          # , showscale = TRUE
                           )) %>%
  add_markers() %>%
  layout(scene = list(xaxis = list(title = 'X1'),
                      yaxis = list(title = 'X2'),
                      zaxis = list(title = 'X3'))
         )
# p
chart_link = api_create(p, filename="scatter3d-colorscale")
# write.csv(data, file = "tsne.csv")
# chart_link
```

10_clusters https://plot.ly/~theophilus_mit/1/#/

## (b)

Which genes are good markers for each cluster? Using the clusters calculated in the previous part as labels for each cell, train a classier on the original data. For example, you could use logistic regression (make sure you include a regularization term because the data is very high-dimensional!). Which genes have the largest coe�cient values for each cluster?

```
# factor to dummy
library(varhandle)
clusters = data.table(to.dummy(data$cluster10, prefix = 'cluster'))

data_reg = cbind(data0,clusters)
# for the first cluster
model1 <- glm(cluster.1 ~.,family=binomial(link='logit'),data=data_reg)
```