載入所需套件

```julia
using StatsBase
using StatsFuns
using Plots
using Statistics
using Random
using HypothesisTests
```

Problem 1. Goodness-of-Fit and Bootstrap(20%)

> **IMPORTANT:** For this problem, you can directly use the density function of binomial distribution. You CANNOT use existing Kolmogorov-Smirnov or bootstrap package.

You are given the following data:

$$6, 7, 3, 4, 7, 3, 7, 2, 6, 3, 7, 8, 2, 1, 3, 5, 8, 7.$$

You want to know whether this data is coming from a binomial distribution with parameters $(8, p)$, where $p \in [0, 1]$ is unknown.

(a) (5%): Compute the corresponding Kolmogorov-Smirnov statistics.

(b) (10%): Write down the algorithm of approximating the p-value of this Kolmogorov-Smirnov statistics based on bootstrap with 10^4 sample.

(c) (5%): Write a program based on your algorithm in (b). Determine the result of the hypothesis testing.

1(a) & 1(c)

```julia
function FromBin(x ; n = 8, nBootstrap = 1e4)
    function max_dif(x, p)
        f = ecdf(x)
        sx = sort(x)
        a = abs.(binomcdf.(n, p, sx) .- f.(sx))
        return max(a...)
    end
    function SimBin(nsim, n, p)
        u = rand(nsim)
        tmp = map(x -> x .>= binomcdf.(n, p, -1:n), u)
        return findlast.(tmp) .- 1
    end
    l = length(x)
    p = sum(x) / (n * l)
    d = max_dif(x, p)
    Bdata = [SimBin(length(x), n, p) for _ in 1:nBootstrap]
    ds = max_dif.(Bdata, map(x -> sum(x) / (n * l), Bdata))
    p_value = mean(ds .> d)
    return (KS_statistic = d, p_value = p_value)
end
```
> FromBin

```julia
Ans = FromBin([6, 7, 3, 4, 7, 3, 7, 2, 6, 3, 7, 8, 2, 1, 3, 5, 8, 7]);
Ans    (KS_statistic = 0.2623320024369238, p_value = 0.0001)
```

這組樣本對應 Binomial(8,p)分配的 Kolmogorov-Smirnov statistic 為 0.262332
用 Bootstrap 求出近似的 p-value 為 0.0001
在$\alpha = 0.05$的顯著水準下，p-value $\leq$ α，因此拒絕虛無假設，即我們有足夠的證據說明資料顯著不來自 Binomial(8,p)分配

1(b)

Step1. 令 k = 1
Step2. 由樣本$X_1, X_2, \dots, X_{18}$計算 MLE $\hat{p}$
Step3. 計算 $D = \max\ |\text{empirical cdf} - F_{\hat{p}})|$
Step4. 生成$\text{Bin}(8, \hat{p})$的隨機值$X_{k,1}, X_{k,2}, \dots, X_{k,18}$
Step5. 由生成的假樣本$X_{k,1}, X_{k,2}, \dots, X_{k,18}$計算 MLE $\hat{p}_k$
Step6. 計算$d_k = \max\ |\text{empirical cdf} - F_{\hat{p}_k})|$
Step7. 如果$k < 10000$，$k = k + 1$，回到 Step4.，否則進到 Step8.
Step8. approximated P-value $= \sum_{k=1}^{10000} I(d_k \geq D)\,/10000$

Problem 2. Hastings-Metropolis Algorithm(20%)

> **IMPORTANT:** For this problem, you can directly use sampling packages that samples from a specific distribution, such as $rnorm()$ or so. You CANNOT use existing MCMC package.

The following data is coming from a normal distribution $N(\mu, 1)$ with unknown μ:

0.8605	1.2175	−0.9772	−0.0378	2.9478
−0.2710	0.0380	0.6765	2.7070	0.5617
1.1110	2.4136	1.0066	2.3637	1.4502
0.2516	0.3485	1.6041	1.2023	1.6049.

Suppose you want to do an Bayesian inference on μ by putting a truncated normal prior $N(0,1)$ on $[0,2]$ for μ. Equivalently, you are consider sampling μ from the following posterior density:

$$f(\mu) := \frac{\prod_{i=1}^{20} \phi(x_i - \mu)\phi(\mu)}{\int_{\nu=0}^{2} \prod_{i=1}^{20} \phi(x_i - \nu)\phi(\nu)d\nu} \propto \prod_{i=1}^{20} \phi(x_i - \mu)\phi(\mu). \tag{1}$$

a) (5%): Write down the MCMC algorithm to sample μ from (1).

b) (10%): Based on your algorithm in a), sample 10^4 μ from (1). Report the histogram, the mean and the standard error of your sampled μ.

c) (5%): Check the dependency of your sampled μ to make sure that they are i.i.d. samples.

2(a)

Step1. 令$x = x_0, i = 0, \text{output} = \text{NULL vector}$

Step2. 生成 y 為 Uniform$\left(x - \frac{2}{3}, x + \frac{2}{3}\right)$的隨機值

Step3. $\alpha = \frac{f(y) * \frac{3}{4}}{f(x) * \frac{3}{4}} = \frac{\prod_{i=1}^{20} \phi(x_i - y)\phi(y)}{\prod_{i=1}^{20} \phi(x_i - x)\phi(x)}$

Step4. 生成$u \sim \text{Uniform}(0,1)$，如果$u > \alpha$，回到 Step2.，否則令$x = y$

Step5. $i = i + 1$，若 $i \in \{1000, 2000, 3000, \ldots\}$，append! (output, x)

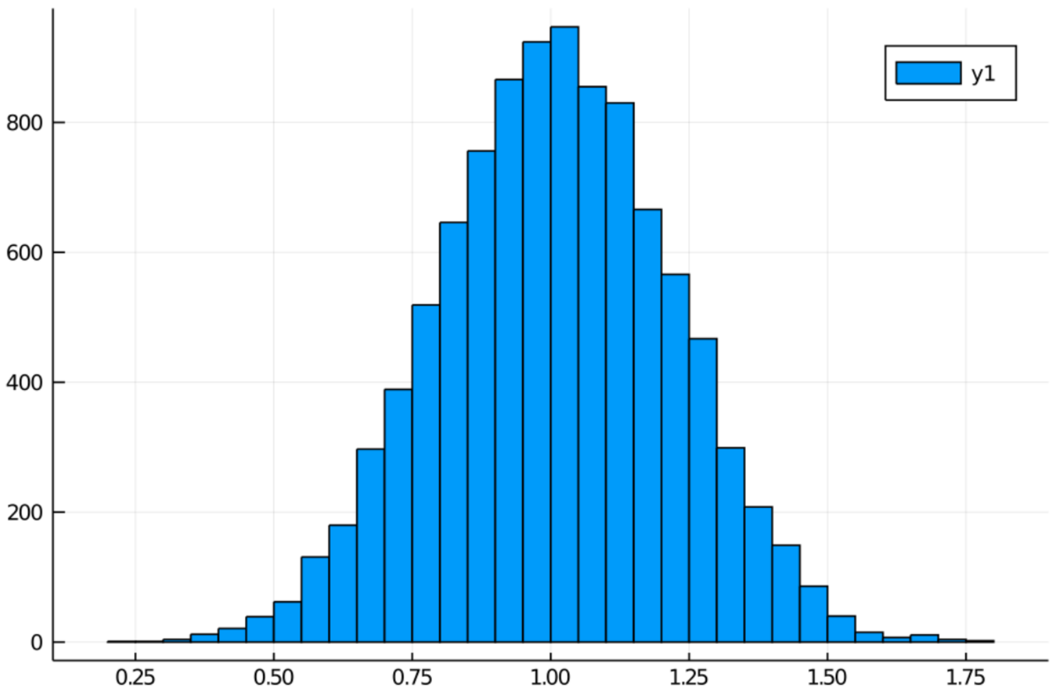
Step6. 若$i < 10^4 * 10^3$，回到 Step2.，否則回傳 output

2(b)

```
function HM_MCMC(d, lower, upper ; nsim = 1e4, iter = 1e3, init = 0)
    output = Array{Float64}(undef, Int(nsim))
    x = init
    l = upper - lower
    i = 0
    while i < iter * nsim
        y = rand() * 2l/3 + x - l/3
        if (y < 0) || (2 < y) ; continue ; end
        α = (prod(normpdf.(d.-y))*normpdf(y)) / (prod(normpdf.(d.-x))*normpdf(x))
        if rand() <= min(α, 1)
            x = y
        end
        i += 1
        if isinteger(i/iter)
            output[Int(i/iter)] = x
        end
    end
    return output
end  > HM_MCMC
```

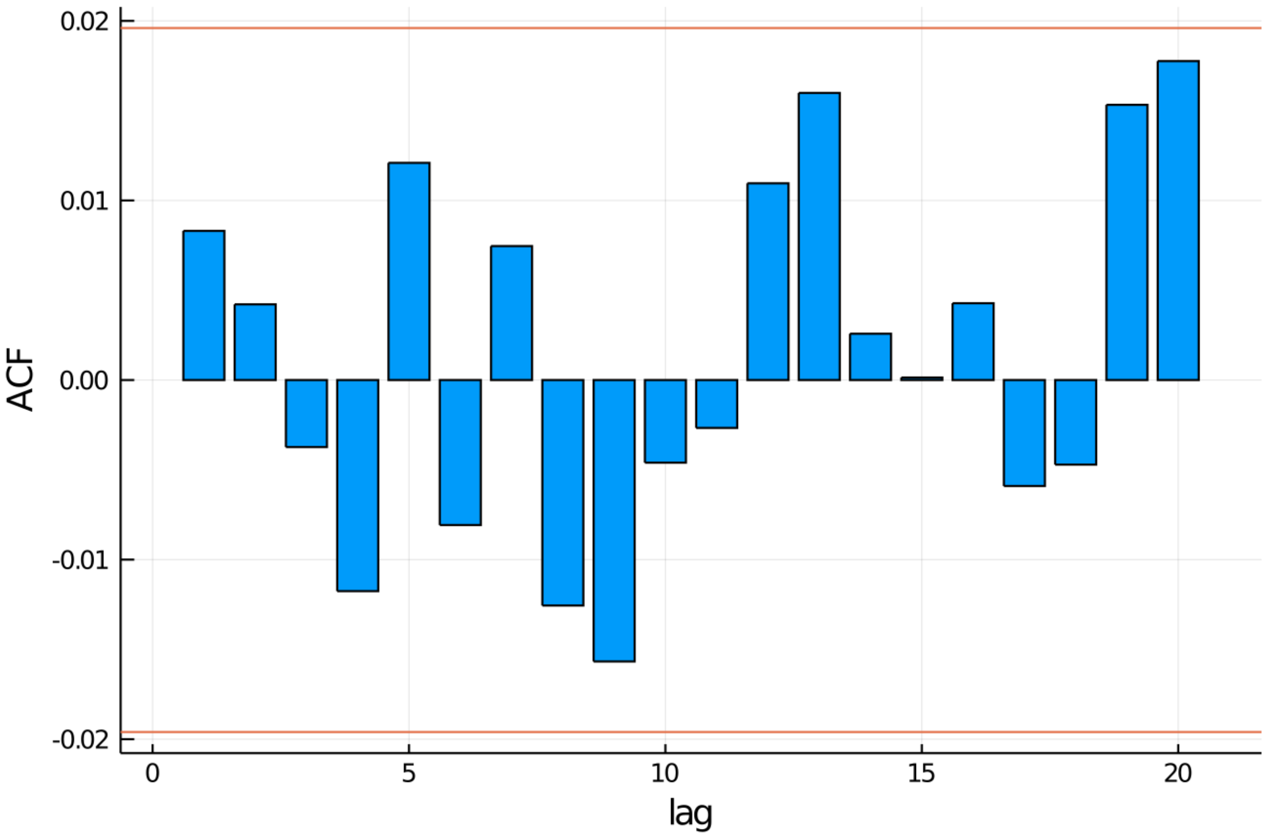
```
d2 = [0.8605, 1.2175, -0.9772, -0.0378, 2.9478,
        -0.2710, 0.0380, 0.6765, 2.7070, 0.5617,
        1.1110, 2.4136, 1.0066, 2.3637, 1.4502,
        0.2516, 0.3485, 1.6041, 1.2023, 1.6049]  > Vector{Float64} with 20 elements


Ans = HM_MCMC(d2, 0, 2, nsim = 1e4)  > Vector{Float64} with 10000 elements
histogram(Ans)  Plot{Plots.GRBackend() n=1}
mean(Ans)  1.0076878334334227
std(Ans)  0.2114998305251261
```

Sample mean 跟 standard error 分別為 1.0077 與 0.2215



2(c)

```
plot(bar(autocor(Ans, 1:20)), legend = nothing)    Plot{Plots.GRBackend() n=1}
plot!([-1.96 * sqrt(1/1e4), 1.96 * sqrt(1/1e4)], seriestype = "hline", legned = nothing)
xaxis!("lag")    Plot{Plots.GRBackend() n=2}
yaxis!("ACF")    Plot{Plots.GRBackend() n=2}
```



先從 ACF 看依序產生的模擬值的自相關程度，上下兩條是漸進的 95%拒絕域

```
print(LjungBoxTest(Ans, 20))
```

```
Ljung-Box autocorrelation test
------------------------------
Population details:
    parameter of interest:   autocorrelations up to lag k
    value under h_0:         "all zero"
    point estimate:          NaN

Test summary:
    outcome with 95% confidence: fail to reject h_0
    one-sided p-value:           0.4908

Details:
    number of observations:         10000
    number of lags:                 20
    degrees of freedom correction:  0
    Q statistic:                    19.481540513720955
```

再用 LB test，p-value = 0.49 > 0.05，因此在 0.05 的顯著水準下，不拒絕虛無假設，即模擬的資料在 lag=20 以內沒有顯著自相關

Problem 3. Gibbs Sampler(20%)

IMPORTANT: For this problem, you can directly use sampling packages that samples from a specific distribution, such as $rnorm()$ or so. You CANNOT use existing MCMC package.

Suppose that for random variables X, Y, N,

$$P\{X = i, y \le Y \le y + dy, N = n\} \propto C_i^n y^{i+\alpha-1}(1-y)^{n-i+\beta-1} e^{-\lambda} \frac{\lambda^n}{n!} dy$$

where $n \in \mathbb{N}, i \in \{0, \cdots, n\}, y \ge 0$, and α, β, γ are constants.

a) (10%): Derive the conditional distribution of X given (Y, N), Y given (X, N), and N given (X, Y).

b) (5%): Based on a), write down the Gibbs sampler algorithm to sample (X, Y, N).

c) (5%): Use the algorithm in b) to simulated 10^4 pairs of (X, Y, N). Report EX, EY and EN.

3(a)

由題目可直觀看出

$Y \sim \mathrm{Beta}(\alpha, \beta)$

$N \sim \mathrm{Poisson}(\lambda)$

$X|Y = y, N = n \ \sim \ \mathrm{Binomial}(n, y) \ \cdots \ (1)$

從而，

$$f_{Y|X,N}(y|i,n) = \frac{\binom{n}{i} y^{i+\alpha-1}(1-y)^{n-i+\beta-1} e^{-\lambda}\frac{\lambda^n}{n!}}{\int_0^1 \binom{n}{i} y^{i+\alpha-1}(1-y)^{n-i+\beta-1} e^{-\lambda}\frac{\lambda^n}{n!} dy} = \frac{y^{i+\alpha-1}(1-y)^{n-i+\beta-1}}{\int_0^1 y^{i+\alpha-1}(1-y)^{n-i+\beta-1} dy}$$

$$= \frac{y^{i+\alpha-1}(1-y)^{n-i+\beta-1}}{B(i+\alpha, n-i+\beta)}$$

$\Rightarrow Y|X = i, N = n \ \sim \ \mathrm{Beta}(i+\alpha, n-i+\beta) \ \cdots \ (2)$

$$P(N = n|X = i, Y = y) = \frac{\binom{n}{i} y^{i+\alpha-1}(1-y)^{n-i+\beta-1} e^{-\lambda}\frac{\lambda^n}{n!}}{\sum_{n=i}^{\infty} \binom{n}{i} y^{i+\alpha-1}(1-y)^{n-i+\beta-1} e^{-\lambda}\frac{\lambda^n}{n!}} = \frac{\binom{n}{i}(1-y)^n \frac{\lambda^n}{n!}}{\sum_{i=n}^{\infty} \binom{n}{i}(1-y)^n \frac{\lambda^n}{n!}} =$$

$$= \frac{\binom{n}{i}(1-y)^n \frac{\lambda^n}{n!}}{\sum_{i=n}^{\infty} \frac{1}{i!(n-i)!}(\lambda(1-y))^n} = \frac{\binom{n}{i}(1-y)^n \frac{\lambda^n}{n!}}{\frac{(\lambda(i-y))^i}{i!} \sum_{i=n}^{\infty} \frac{(\lambda(1-y))^{n-i}}{(n-i)!}} = \frac{\binom{n}{i}(1-y)^n \frac{\lambda^n}{n!}}{\frac{(\lambda(i-y))^i}{i!} e^{\lambda(1-y)}}$$

$$= e^{-\lambda(1-y)} \frac{(\lambda(i-y))^{n-i}}{(n-i)!}, \qquad n = i, i+1, i+2, \ldots$$

$\Rightarrow N - i|X = i, Y = y \ \sim \ \mathrm{Poisson}(\lambda(1-y)) \ \cdots \ (3)$

3(b)

Step1. 令$x = x_0, y = y_0, n = n_0, i = 0$, output = NULL vector

Step2. 生成 j 為 Discrete Uniform(1,2,3)的隨機值

Step3. 如果$j = 1$，令 $x = \mathrm{Binomial}(n, y)$的隨機值，並跳到 Step6.

Step4. 如果$j = 2$，令 $y = \mathrm{Beta}(x+\alpha, n-x+\beta)$的隨機值，並跳到 Step6.

Step5. 如果$j = 3$，令 $n = \mathrm{Poisson}(\lambda(1-y))$的隨機值再加 x

Step6. $i = i + 1$，若 $i \in \{3000, 6000, 9000, \ldots\}$，append! (output, x)

Step7. 若$i < 10^4 * 3 * 10^3$，回到 Step2.，否則回傳 output

3(c)

```
function Gibbs_MCMC(α, β, λ ; nsim = 1e4, iter = 3e3, init = [3, 1/2, 5])
    output = Array{Float64}(undef, Int(nsim), 3)
    x = copy(init)
    i = 0
    while i < nsim * iter
        j = sample([1, 2, 3])
        if j == 1
            global x[1] = binominvcdf(x[3], x[2], rand())
        elseif j == 2
            global x[2] = betainvcdf(α + x[1], β + x[3] - x[1], rand())
        else
            global x[3] = poisinvcdf(λ*(1 - x[2]), rand()) + x[1]
        end
        i += 1
        if isinteger(i/iter)
            output[Int(i/iter), :] = [x...]
        end
    end
    return output
end
```

```
Ans = Gibbs_MCMC(5, 5, 7)      > 10000×3 Array{Float64,2}:
mean(Ans[:,1])     3.5164
mean(Ans[:,2])     0.5006932178760732
mean(Ans[:,3])     7.0135
```

EX=3.5164

EY=0.5

EN=7.0135

Problem 4. Simulated Annealing(20%)

IMPORTANT: For this problem, you can directly use the random permutation function. You CANNOT use existing simulated annealing package.

Consider a traveling salesman problem in which the salesman starts at city 0 and must travel in turn to each of the 10 cities $1, \cdots, 10$ according to some permutation of $1, \cdots, 10$. Let the reward earned by the salesman when he goes directly from city i to city j be $U_{i,j}$.

a) (5%): Write down the simulated annealing algorithm for finding the maximum of the salesman's reward.

b) (5%): Generate 100 random numbers $U_{0,k}, k = 1, \cdots, 10, U_{i,j}, i \neq j, i, j = 1, \cdots, 10$.

c) (5%): Based on your algorithm in a) and the sampled $U_{i,j}$ in b), do a simulated annealing. Report the maximum reward and the corresponding order of cities that the salesman should travel.

d) (5%): Repeat b) and c) for 10^6 times, and report the mean and variance for the maximum reward.

4(a)

Step1. 設定初始路線$x = [0,1,2,3,4,5,6,7,8,9,10], k = 1, C = 1$

Step2. $r = \text{reward}(x), \text{result} = x$, 分別代表目前為止最大的 reward 與其路徑

Step3. 除了第一個位置之外，在 x 中隨機選兩個元素對調
(包含自己與自己對調)，並賦值給 y

Step4. 生成 $u \sim \text{Uniform}(0,1)$, 令 $\lambda = C * \log(k+1)$

Step5. $\alpha = \exp\left(\lambda\left(\text{reward}(y) - \text{reward}(x)\right)\right)$，如果 $u > \alpha$，回到 Step3.，否則令$x = y$

Step6. 如果 $\text{reward}(x) \geq r$，則令 $r = \text{reward}(x), \text{result} = x$

Step7. $k = k + 1$，如果 $k > 10000$，回傳 result 與 r，否則回到 Step3.

4(b)

```
u = Array{Float64}(undef, 11, 10)    > 11×10 Array{Float64,2}:
for i in 1:11, j in setdiff(1:10, i)
    u[i,j] = rand()
end    ✓
u    > 11×10 Array{Float64,2}:
```

4(c)

```
function Simulated_Annealing2(m ; iter = 60, init = collect(0:10), C = 1)
    function reward(v)
        r = m[11, v[2]]
        r += sum([m[v[i], v[i+1]] for i in 2:10])
        return r
    end
    function find_neighbor(v)
        t = copy(v)
        (i, j) = rand(2:11, 2)
        t[i], t[j] = t[j], t[i]
        return t
    end
    x = copy(init)
    result = copy(x)
    a = reward(result)
    k = 1
```

```
    while k <= iter
        y = find_neighbor(x)
        λ = C * log(k + 1)
        if rand() <= exp(λ*(reward(y) - reward(x)))
            x = copy(y)
            if reward(y) >= a
                a = reward(y)
                result = copy(y)
            end
            k += 1
        end
    end
    return (path = result, reward = a)
end    > Simulated_Annealing2
```

```
a4 = Simulated_Annealing2(u, iter = 10000, C = 1);
a4    (path = [0, 9, 8, 4, 2, 5, 3, 10, 1, 6, 7], reward = 8.635371745536425)
```

Maximum reward = 8.63537, 路徑為$0 \rightarrow 9 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 10 \rightarrow 1 \rightarrow 6 \rightarrow 7$

4(d)

```
u = reshape(rand(11*10*Int(1e6)), (11,10,Int(1e6)))    > 11×10×1000000 Array{Float64,3}:
@time p4 = [Simulated_Annealing2(u[:,:,i], iter = 10000, C = 1)[2] for i in 1:Int(1e6)]
mean(p4)    8.641091720133895
var(p4)    0.09398018035864722
```

Mean = 8.641

Variance = 0.094

Problem 5. EM Algorithm(20%)

> **IMPORTANT:** For this problem, you can directly use the density function of binomial and Poison distribution. You CANNOT use existing EM algorithm package.

Consider the binomial/Poison mixture problem in the slide of Week 12-1, page 26-38. The data is given in page 38.

a) (10%): Write down the EM algorithm for this problem.

b) (5%): Reconstruct the table in page 38 by setting the initial as $\xi^0 = 0.75$ and $\lambda^0 = 0.4$.

c) (5%): Repeat b), but with $\xi^0 = 0.5$ and $\lambda^0 = 0.6$.

5(a)

Step1. $\xi = \xi_0, \lambda = \lambda_0, \epsilon = 10^{\wedge} - 16$

Step2. $n_A = \frac{n_0 \xi}{\xi + (1-\xi)e^{-\lambda}}, n_B = n_0 - n_A$

Step3. $\xi^* = \frac{n_A}{N}, \lambda^* = \frac{\sum_{x=1}^{6} x * n_x}{N - n_A}$

Step4. 如果$\sqrt{(\xi-\xi^*)^2 + (\lambda-\lambda^*)^2} > \epsilon$，令 $\xi = \xi^*, \lambda = \lambda^*$，並回到 Step2.

Step5. 回傳 ξ, λ, n_A, n_B

5(b)

```
function EM_algorithm(v, ξ₀, λ₀ ; ϵ = 1e-16)
    ξ = copy(ξ₀)
    λ = copy(λ₀)
    N = sum(v)
    t = 0
    d = Inf
    table = [Text.(["t" "ξ" "λ" "n_A" "n_B"]);Array{Any}(nothing, 6, 5)]
    while d > ϵ
        n_A = v[1]*ξ / (ξ+(1-ξ)*exp(-λ))
        n_B = v[1] - n_A
        if t <= 5
            table[t+2,:] = Any[Int8(t) ξ λ n_A n_B]
        end
        t += 1
        D = [ξ, λ] - [n_A/N, sum(collect(1:6).*v[2:end])/(N-n_A)]
        d = sqrt(sum(D.^2))
        (ξ, λ) = [ξ, λ] - D
    end
    table
end
```

```
data = [3062, 587, 284, 103, 33, 4, 2]
EM_algorithm(data, 0.75, 0.4)
```

> Vector{Int64} with 7 elements

7×5 Array{Any,2}:

t	ξ	λ	n_A	n_B
0	0.75	0.4	2502.78	559.221
1	0.614179	1.03548	2503.59	558.409
2	0.614378	1.03601	2504.22	557.781
3	0.614532	1.03643	2504.71	557.295
4	0.614652	1.03675	2505.08	556.919
5	0.614744	1.037	2505.37	556.629

5(c)

```
EM_algorithm(data, 0.5, 0.6)
```

7×5 Array{Any,2}:

t	ξ	λ	n_A	n_B
0	0.5	0.6	1977.0	1085.0
1	0.485153	0.775977	2057.21	1004.79
2	0.504837	0.806823	2129.76	932.242
3	0.52264	0.836914	2194.15	867.847
4	0.538442	0.865568	2250.27	811.728
5	0.552214	0.892188	2298.33	763.666