載入所須套件

```
import SpecialFunctions.gamma
using StatsFuns
using Statistics
using Plots
using JuMP
using Ipopt
using PoissonRandom
using Distributions
```

Problem 1. Dirichlet Distribution(35%)

IMPORTANT: For this problem, the computer only knows how to generate Unif$(0,1)$. The random functions such as rnorm() doesn't exist for you.

A Dirichlet distribution is a continuous distribution on $A = \left\{x = (x_1, \cdots, x_d) : x_i \geq 0, \sum_{i=1}^d x_i = 1\right\}$ with pdf

$$f(x) = \frac{\Gamma(\sum_{j=1}^d \alpha_j)}{\prod_{j=1}^d \Gamma(\alpha_j)} x_1^{\alpha_1 - 1} x_2^{\alpha_2 - 2} \cdots x_d^{\alpha_d - 1}.$$

We want to sample from this distribution.

(a) (5%): Write down the algorithm to sample using acceptance-rejection method with a proposal $g(x)$.

(b) Write a code to sample 10^4 data points through your algorithm in (a) with $d = 3$, $(\alpha_1, \alpha_2, \alpha_3) = (2, 3, 4)$, and g being the uniform distribution on $[0, 1]^3$. Report the following:

- (5%) histogram for x_1, x_2 and x_3, individually;
- (5%) the sample mean and covariance matrix;
- (5%) EN and $VarN$, where N is the number of iterations to accept.

(c) (5%) There's a way to sample Dirichlet distribution through Gamma distribution. Write down the corresponding algorithm (Note. Check the slides for the relationship between Dirichlet and Gamma. Note that your computer still only know how to sample Unif$(0,1)$.)

(d) Write a code to sample 10^4 data points through your algorithm in (c) with $d = 3$, $(\alpha_1, \alpha_2, \alpha_3) = (2, 3, 4)$, and g being the uniform distribution on $[0, 1]^3$. Report the following:

- (5%) Histogram for x_1, x_2 and x_3;
- (5%) the sample mean and covariance matrix.

(a)

Algorithm:

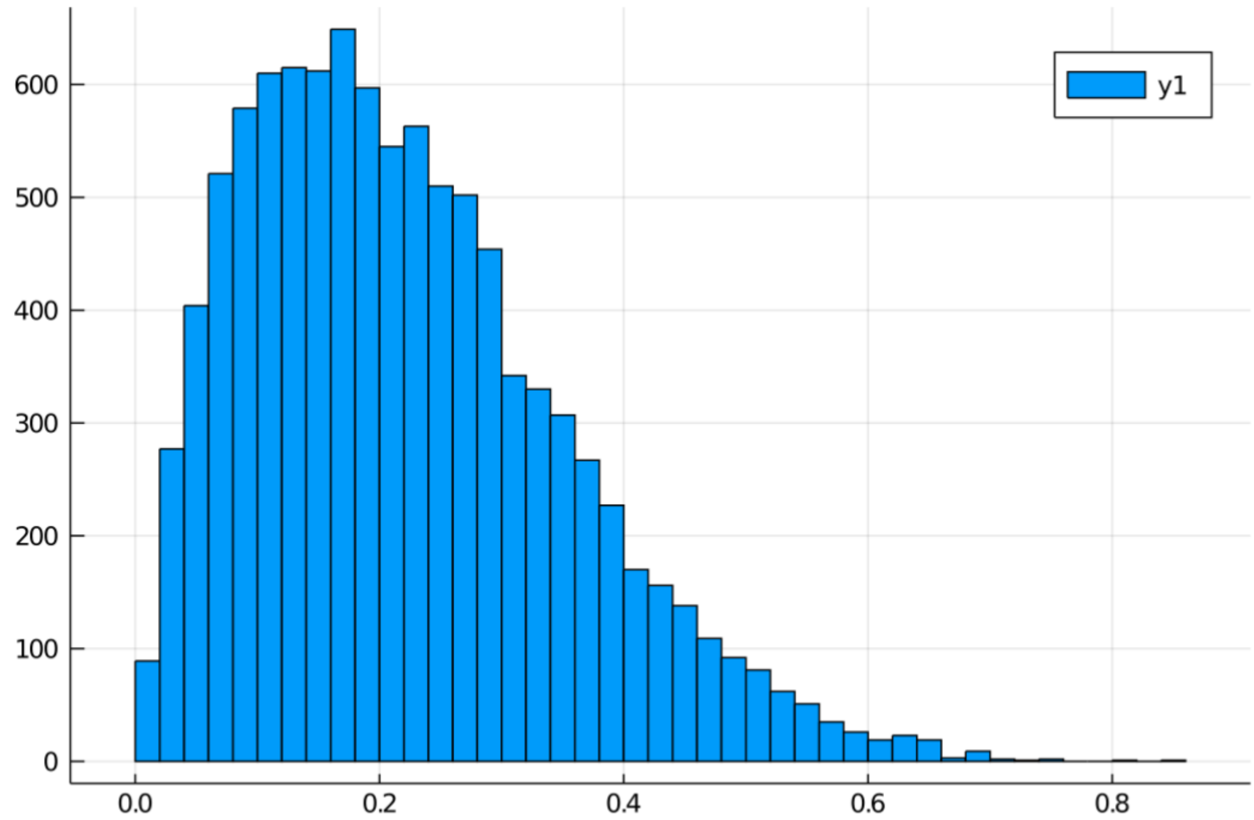
1. 令 $c = \max_{x \in A} \frac{f(x)}{g(x)}$
2. 生成 $U_1 \sim U(0,1)$
3. 用 inverse method 來生成 $y = G^{-1}(U_1)$, 其中 G 為 proposal 的 cdf
4. 生成 $U_2 \sim U(0,1)$
5. 若 $U_2 \leq \frac{f(y)}{c \cdot g(y)}$, 則令 x = y, 反之, 回到 Step2

(b)

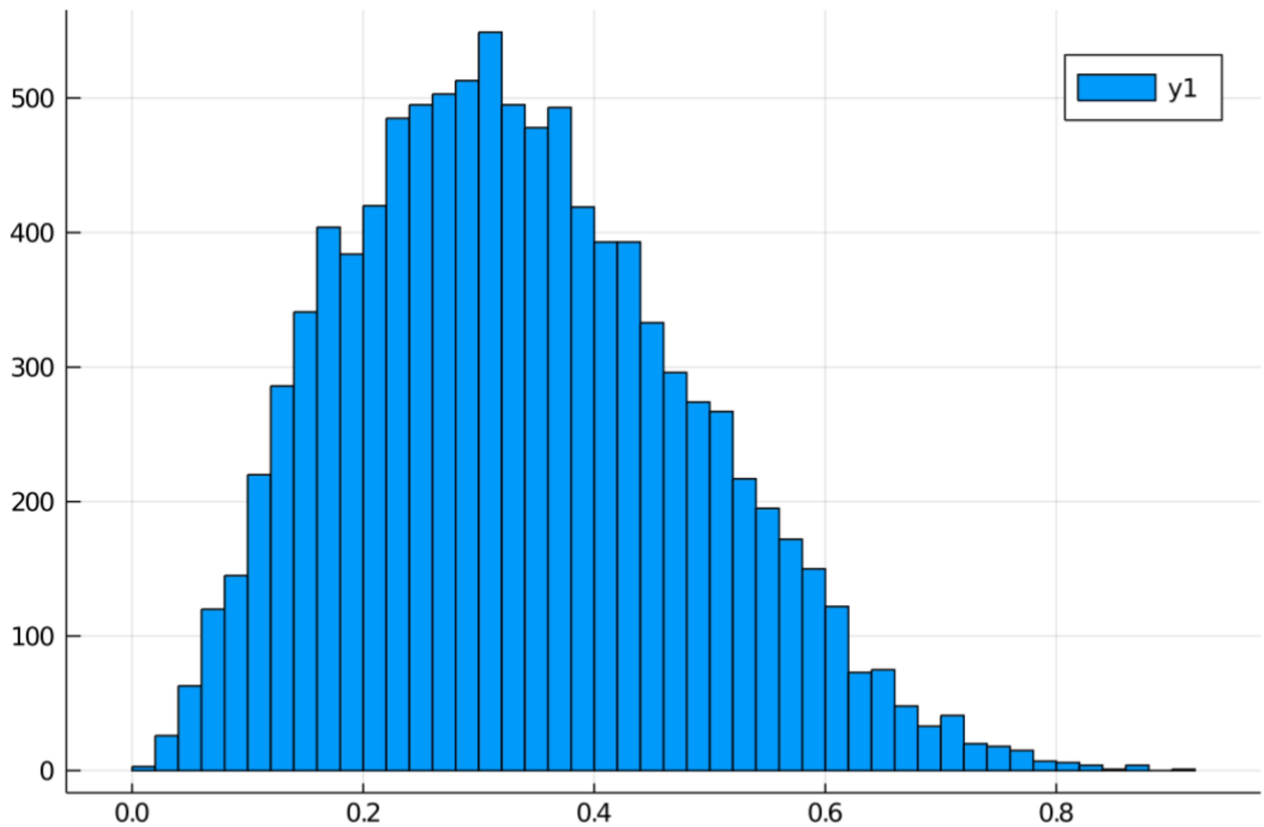
```julia
function p1b(nsim = 10000)
    function find_y()
        y = Array{Float64}(undef, 3)
        y[3] = -1
        while y[3] < 0
            y[1:2] = rand(2)
            y[3] = 1 - sum(y[1:2])
        end
        return y
    end
    f(x) = gamma(sum([2, 3, 4])) / prod(gamma.([2, 3, 4])) * x[1] * x[2]^2 * x[3]^3
    output = Array{Float64}(undef, nsim, 3)
    N = Array{Float64}(undef, nsim)

    # Step1
    m = Model(Ipopt.Optimizer)
    @variable(m, 0 <= x[1:3] <= 1)
    @NLobjective(m, Max, 40320 / 12 * x[1] * x[2]^2 * x[3]^3)
    @constraint(m, x[1] + x[2] + x[3] == 1)
    optimize!(m)
    c = JuMP.objective_value(m)
```

```julia
    i = 1
    j = 1
    while i <= nsim
        y = find_y()        # Step2 + Step3
        U₂ = rand(1)[1]     # Step4
        if U₂ <= f(y)/c     # Step5
            output[i,:] = y
            N[i] = j
            i += 1
            j = 1
        else
            j += 1
        end
    end
    return (output = output, N = N)
end   > p1b

a = p1b();   ✓
```

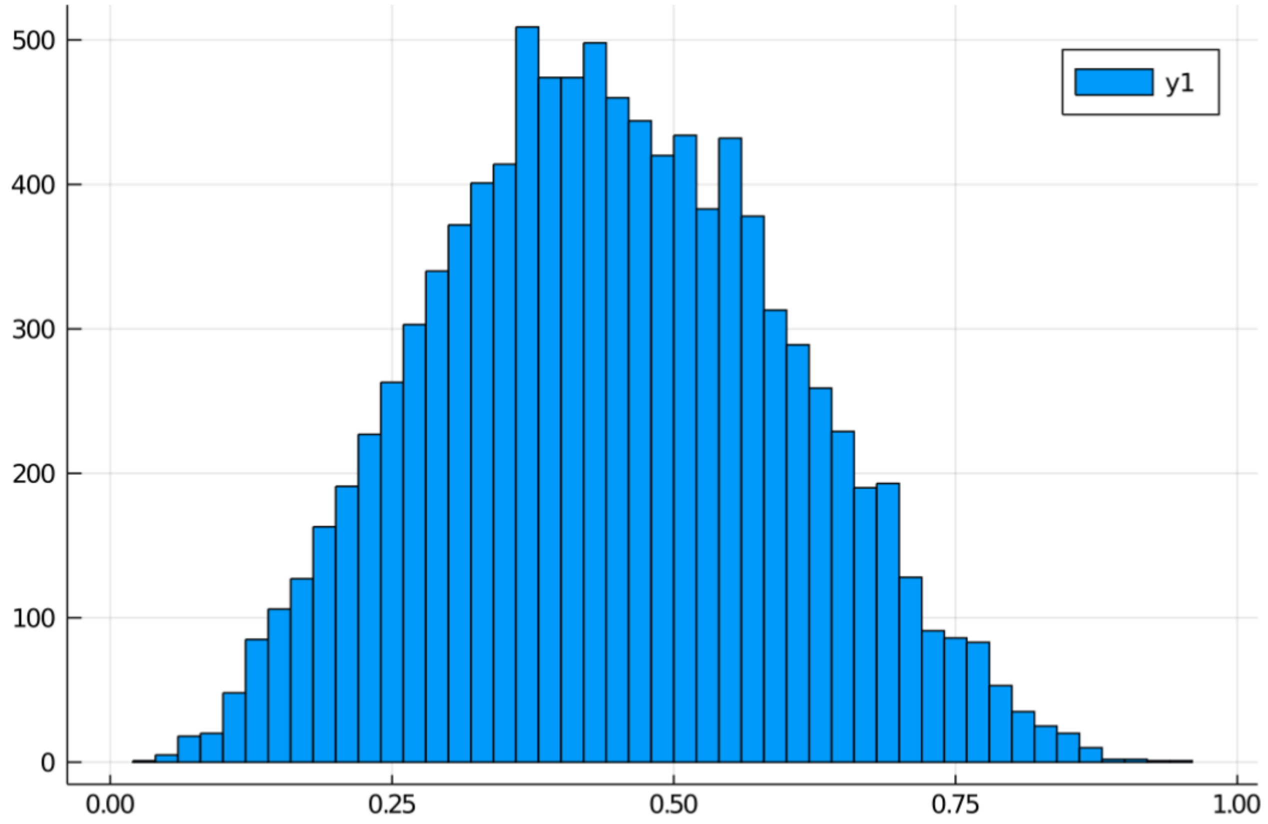
```julia
gr()   Plots.GRBackend()
histogram(a.output[:,1])   Plot{Plots.GRBackend() n=1}
histogram(a.output[:,2])   Plot{Plots.GRBackend() n=1}
histogram(a.output[:,3])   Plot{Plots.GRBackend() n=1}
```



600
500
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
y1



500
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
y1




```
mean(a.output[:,1]), mean(a.output[:,2]), mean(a.output[:,3])   (0.221…, 0.334…, 0.444…)
cov(a.output)   3x3 Array{Float64,2}:
                  0.0167895    -0.00720845   -0.00958106
                 -0.00720845    0.0219472    -0.0147388
                 -0.00958106   -0.0147388     0.0243198
mean(a.N), var(a.N)   (3.85…, 11.0…)
```

Sample mean = (0.221, 0.334, 0.444)

$$\text{Covariance matrix} = \begin{bmatrix} 0.0167895 & -0.00720845 & -0.00958106 \\ -0.00720845 & 0.0219472 & -0.0147388 \\ -0.00958106 & -0.0147388 & 0.0243198 \end{bmatrix}$$

EN = 3.85, Var(N) = 11.0

(c)

Algorithm:

1. 生成 $\mathrm{U}_1, \mathrm{U}_2, \dots, \mathrm{U}_{\sum_{i=1}^{d} \alpha_i} \sim U(0,1)$
2. 算得 $\mathrm{Gamma}(\alpha_i, 1)$ ，$i = 1, \dots, d$ 的隨機值

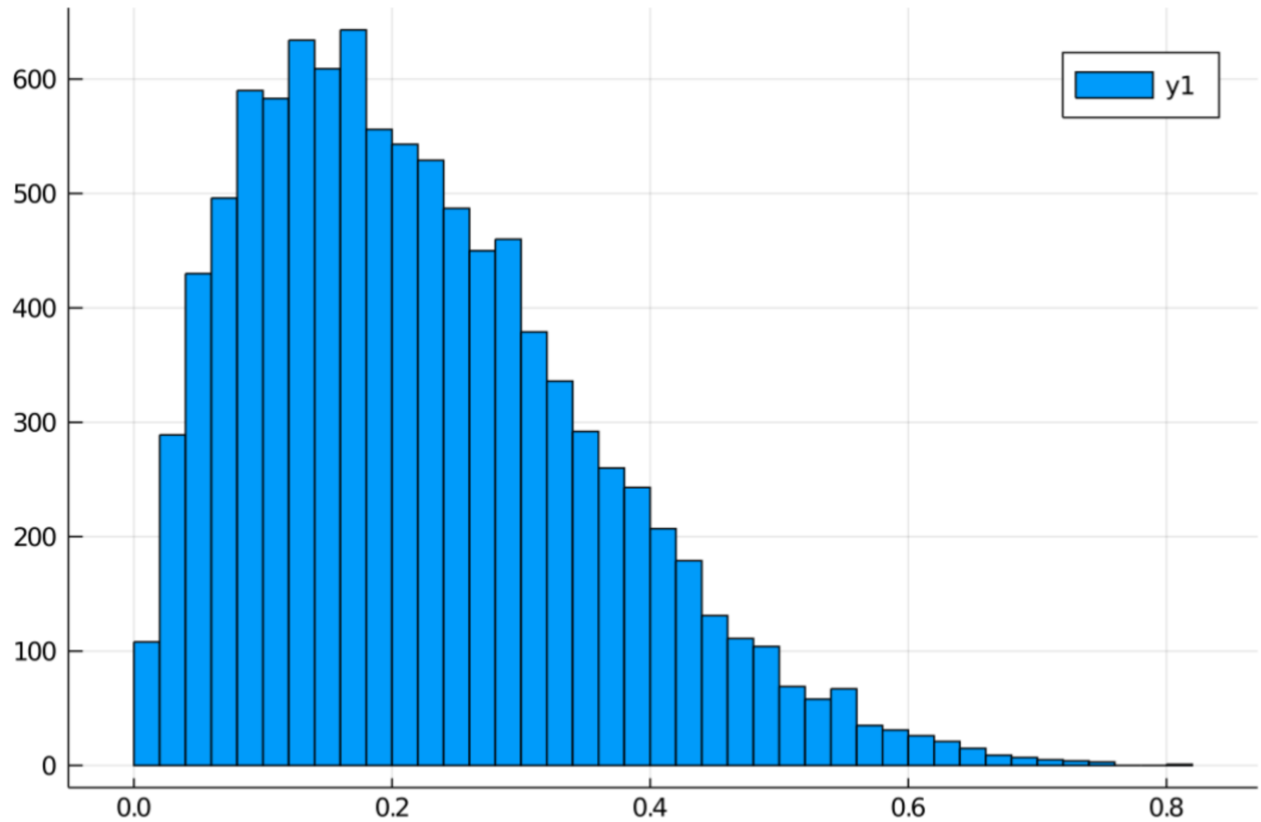
$$y_1 = -\log(U_1) - \cdots - \log(U_{\alpha_1})$$
$$y_2 = -\log(U_{\alpha_1+1}) - \cdots - \log(U_{\alpha_1+\alpha_2})$$
$$\vdots$$
$$y_d = -\log\left(U_{\sum_{i=1}^{d-1} \alpha_i + 1}\right) - \cdots - \log\left(U_{\sum_{i=1}^{d} \alpha_i}\right)$$

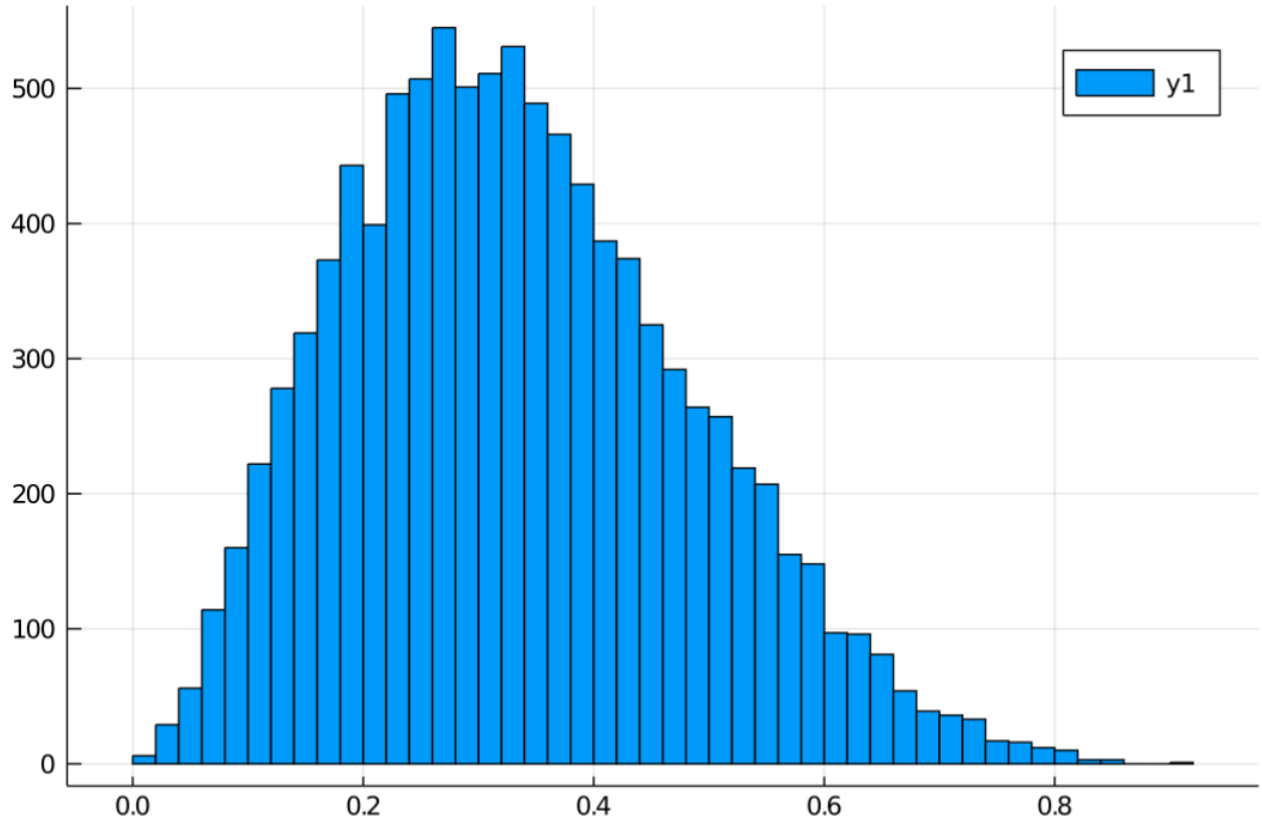
3. 得到Dirichlet 分配的隨機值$(x_1, \dots, x_d)$，$x_j = \frac{y_j}{\sum_{i=1}^{d} y_i}$ ，$j = 1, \dots, d$

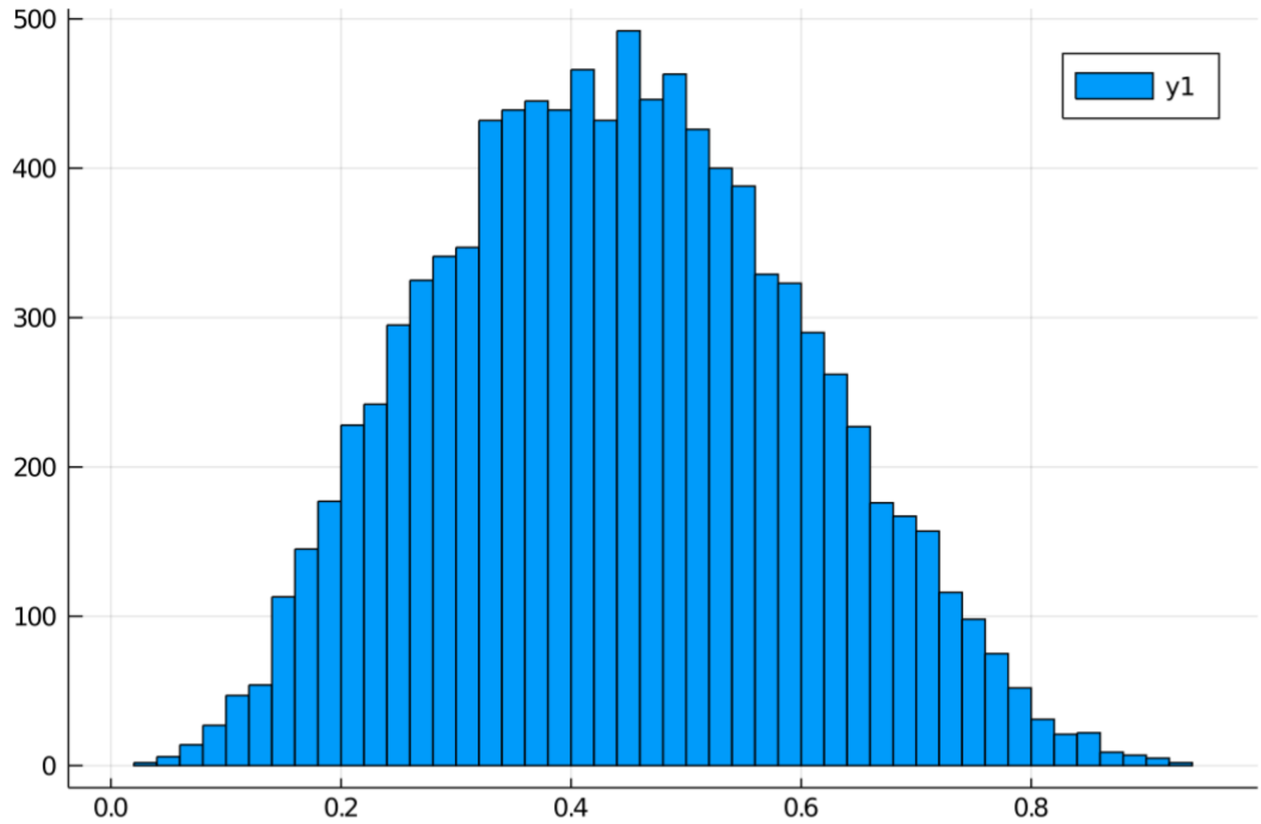
(d)

```
function p1d(nsim = 10000)
    output = Array{Float64}(undef, nsim, 3)
    for i in 1:nsim
        tmp = -log.(rand(9))
        y = [sum(tmp[1:2]), sum(tmp[3:5]), sum(tmp[6:9])]
        output[i,:] = y / sum(y)
    end
    return output
end    > p1d

a = p1d()    > 10000×3 Array{Float64,2}:

histogram(a[:,1])    Plot{Plots.GRBackend() n=1}
histogram(a[:,2])    Plot{Plots.GRBackend() n=1}
histogram(a[:,3])    Plot{Plots.GRBackend() n=1}

mean(a[:,1]), mean(a[:,2]), mean(a[:,3])    (0.223…, 0.335…, 0.442…)
```

Sample mean = (0.223, 0.335, 0.442)




y1
0
100
200
300
400
500
0.0
0.2
0.4
0.6
0.8


y1
0
100
200
300
400
500
0.0
0.2
0.4
0.6
0.8

Problem 2. Variance Reduction(35%)

> **IMPORTANT:** For this problem, you are allow to use functions such as rnorm() or so, but do NOT directly use any advanced function such as directly sample composite random variables or do importance sampling. If you are not sure whether a function can be used, please ask.

We want to estimate the probability

$$P\left\{A = \sum_{i=1}^{N} X_i > 10\right\}$$

where X_i are i.i.d. normal with mean 0 and variance 1, and N is a Poison random variable with mean 5, independent to X_i.

For problem (a)-(f), **write down the algorithm for each of them, sample 10^5 data point, and report your sample mean and variance.**

(a) (5%) Naive Monte Carlo;

(b) (5%) using $B = N$ as a control variate (with optimal constant);

(c) (5%) variance reduction through conditioning on N;

(d) (5%) importance sampling by sampling N from a Poison distribution with mean 10 instead;

(e) (5%) importance sampling by sampling X_i from a normal distribution with mean 2 and variance 1 instead;

(f) (5%) importance sampling by sampling N from a Poison distribution with mean 10 AND sampling X_i from a normal distribution with mean 2 and variance 1;

(g) (5%) Among methods (a)-(f), which one will you prefer? Why?

Hint for (d)-(f): It will be easier if you consider the probability as $E[f(N, X_1, \cdots, X_N)]$.

(a)

Algorithm:

1. 生成 N ~ Poisson(5)
2. 生成 $X_1, \ldots, X_N \overset{iid}{\sim} N(0,1)$
3. 若 $\sum_{i=1}^{N} X_i > 10$, a = 1, 否則 a = 0
4. 重複上述步驟 100000 次

```
p2a(nsim = 1e5) = sum.(randn.([pois_rand(5) for _ in 1:nsim])) .> 10    > p2a
a = p2a()    > 100000-element BitArray{1}:
mean(a)    0.0000600
var(a)    0.0000600…
```

Mean = 0.00006

Variance = 0.00006

(b)

Algorithm:

1. 生成 $N_i \sim \text{Poisson}(5),\ i = 1, \dots, 100000$
2. 生成 $X_1, \dots, X_{N_i} \overset{iid}{\sim} N(0,1),\ i = 1, \dots, 100000$
3. 令 $Y_i = I\left[\sum_{j=1}^{N_i} X_j > 10\right],\ i = 1, \dots, 100000$
4. 令 $c = -\frac{Cov(Y,N)}{Var(N)}$
5. 回傳 $a_i = Y_i + c * (N_i - 5),\ i = 1, \dots, 100000$

```
function p2b(nsim = 1e5)
    N = [pois_rand(5) for _ in 1:nsim]
    x = sum.(randn.(N)) .> 10
    return x .- cov(Int64[x...], N)/var(N) * (N .- 5)
end    > p2b

a = p2b()    > Vector{Float64} with 100000 elements
mean(a)    0.0000498…
var(a)    4.998746547854668e-5
```

Mean = 0.0000498

Variance = 4.998746547854668e-5

(c)

Algorithm:

1. 生成 $N \sim \text{Poisson}(5)$
2. 回傳 $a = P(X > 10),\ X \sim N(0, N)$
3. 重複上述步驟 100000 次

```
function p2c(nsim = 1e5)
    N = [pois_rand(5) for _ in 1:nsim]
    return normccdf.(0, sqrt.(N), 10)
end    > p2c

a = p2c()    > Vector{Float64} with 100000 elements
mean(a)    0.0000788…
var(a)    6.379520224795173e-8
```

Mean = 0.0000788

Variance = 6.379520224795173e-8

(d)

Algorithm:

1. 生成 $\mathrm{N} \sim \mathrm{Poisson}(10)$
2. 生成 $X_1, \dots, X_N \overset{iid}{\sim} \mathrm{N}(0,1)$
3. 令 $\mathrm{Y} = I[\sum_{i=1}^{N} X_i > 10]$
4. 回傳 $\mathrm{a} = \mathrm{Y} * \frac{\mathrm{f}(\mathrm{N};\lambda=5)}{\mathrm{f}(\mathrm{N};\lambda=10)}$, f 為 Poisson 的 pdf
5. 重複上述步驟 100000 次

```
function p2d(nsim = 1e5)
    N = [pois_rand(10) for _ in 1:nsim]
    x = sum.(randn.(N)) .> 10
    return x .* poispdf.(5, N) ./ poispdf.(10, N)
end    > p2d

a = p2d()    > Vector{Float64} with 100000 elements
mean(a)    0.0000881…
var(a)    7.926150832438573e-5
```

Mean = 0.0000881

Variance = 7.926150832438573e-5

(e)

Algorithm:

1. 生成 $\mathrm{N} \sim \mathrm{Poisson}(10)$
2. 生成 $X_1, \dots, X_N \overset{iid}{\sim} \mathrm{N}(0,1)$
3. 令 $\mathrm{Y} = I[\sum_{i=1}^{N} X_i > 10]$
4. 回傳 $\mathrm{a} = \mathrm{Y} * \frac{\mathrm{f}(x_1,\dots,x_N;\mu=1)}{\mathrm{f}(x_1,\dots,x_N;\mu=2)}$, f 為 N 個獨立 $\mathrm{Normal}(\mu, 1)$的 joint pdf
5. 重複上述步驟 100000 次

```
function p2e(nsim = 100000)
    output = Array{Float64}(undef, nsim)
    N = [pois_rand(5) for _ in 1:nsim]
    x = map(x -> x .+ 2, randn.(N))
    for i in 1:nsim
        output[i] = (sum(x[i]) > 10) .* prod(normpdf.(x[i])) ./ prod(normpdf.(2, 1, x[i]))
    end
    return output
end    > p2e

a = p2e()    > Vector{Float64} with 100000 elements
mean(a)    0.0000564…
var(a)    6.774405707264774e-6
```

Mean = 0.0000564

Variance = 6.774405707264774e-6

(f)

Algorithm:

1. 生成 $N \sim \text{Poisson}(10)$
2. 生成 $X_1, \dots, X_N \overset{iid}{\sim} N(2,1)$
3. 令 $Y = I[\sum_{i=1}^{N} X_i > 10]$
4. 回傳 $a = Y * \frac{f(x_1,\dots,x_N;\mu=1)}{f(x_1,\dots,x_N;\mu=2)} * \frac{g(N;\lambda=5)}{g(N;\lambda=10)}$,

   f 為 N 個獨立 Normal(μ, 1)的 joint pdf,
   g 為 Poisson 的 pdf
5. 重複上述步驟 100000 次

```
function p2f(nsim = 100000)
    output = Array{Float64}(undef, nsim)
    N = [pois_rand(10) for _ in 1:nsim]
    x = map(x -> x .+ 2, randn.(N))
    for i in 1:nsim
        r₁ = prod(normpdf.(x[i])) / prod(normpdf.(2, 1, x[i]))
        r₂ = poispdf(5, N[i]) / poispdf(10, N[i])
        output[i] = (sum(x[i]) > 10) .* r₁ .* r₂
    end
    return output
end    > p2f

a = p2f()    > Vector{Float64} with 100000 elements
mean(a)    0.0000755…
var(a)    5.005924163314944e-6
```

Mean = 0.0000755

Variance = 5.005924163314944e-6

(g)

我認為(c)小題的做法最好，因為變異數是上面方法中最小的。

Problem 3. Bootstrap and Jacknife

IMPORTANT: For this problem, you are allow to use functions such as rnorm() or so, but do NOT directly use any advanced function such as bootstrap package. If you are not sure whether a function can be used, please ask.

You obtain the following data with 20 data points:

0.0839	0.0205	0.3045	0.7816	0.0003
0.0095	0.4612	0.9996	0.9786	0.7580
0.0002	0.7310	0.0777	0.4483	0.4449
0.7943	0.1447	0.0431	0.8621	0.3273

The sample median is $\hat{med} = 0.3861$, which is an estimator of the population median. Now we want to construct its confidence interval .

(a) (10%) Write down the procedure of Jacknife. Write a code for it, plot the histogram of your jackknife sample, and report the 95%-confidence interval you obtained.

(b) (10%) Write down the procedure of Bootstrap (sampling with replacement.) Write a code for it, plot the histogram of your bootstrap sample, and use it to construct the 95%-confidence interval with 10^4 bootstrap sampling.

(c) (10%) Suppose now we know that the data is coming from a Beta distribution $\beta(a, b)$ with unknown (a, b). Write down the procedure of **Parametric Bootstrap**. Write a code for it, plot the histogram of your bootstrap sample, and use it to construct the 95%-confidence interval with 10^4 bootstrap sampling. (*Hint.* Check the slides Week 5-3. You are allow to use a package to directly estimate Beta distribution for this problem.)

```
data = [0.0839, 0.0205, 0.3045, 0.7816, 0.0003,
        0.0095, 0.4612, 0.9996, 0.9786, 0.7580,
        0.0002, 0.7310, 0.0777, 0.4483, 0.4449,
        0.7943, 0.1447, 0.0431, 0.8621, 0.3273]
```

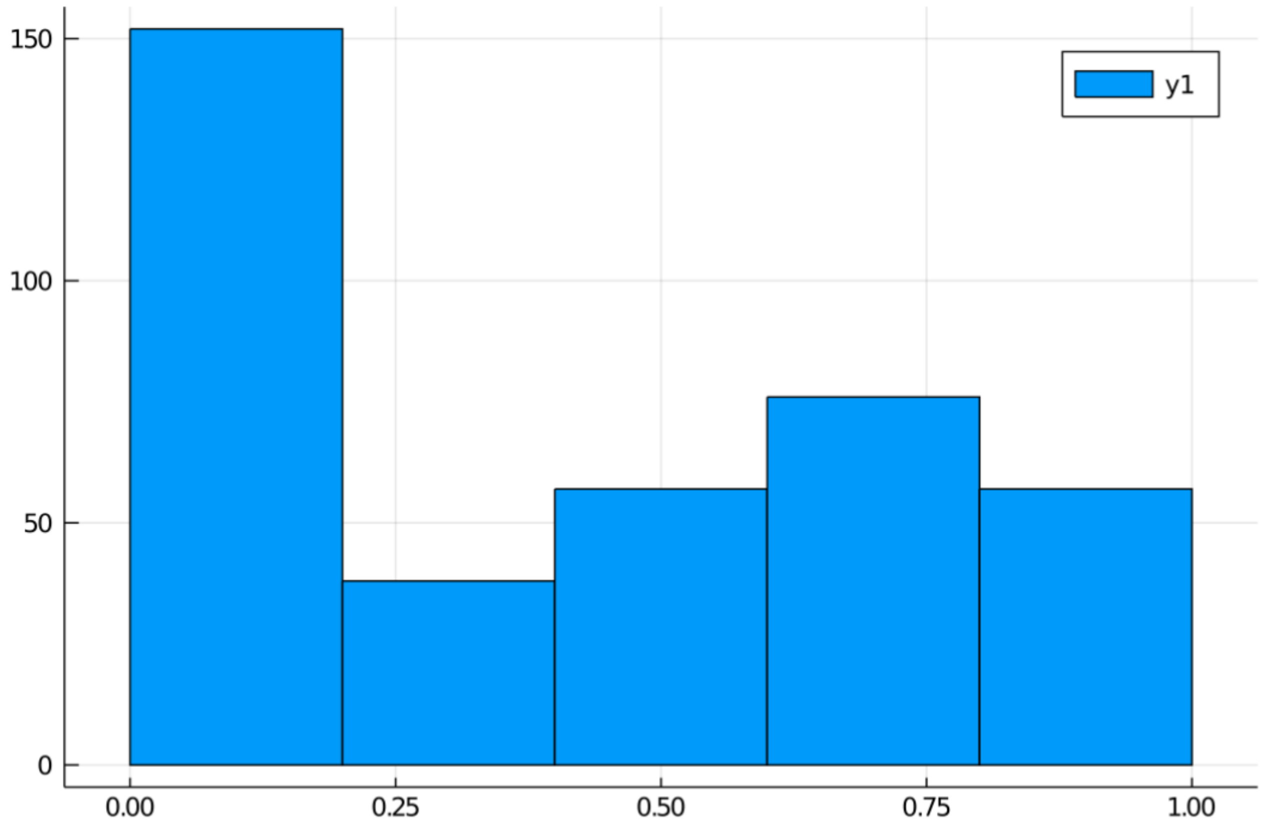
(a)

Jacknife:

1. 原始資料為 $X = \{X_1, X_2, \dots, X_n\}$
2. 令 $X_{(i)}^* = X \setminus X_i$
3. 可得 $S(X_{(1)}^*), S(X_{(2)}^*), \dots, S(X_{(n)}^*)$, S為我們有興趣的統計量
4. 計算統計量的變異數 $= \frac{n-1}{n} \sum_{i=1}^{n} \left(S(X_{(i)}^*) - \overline{S(X^*)} \right)^2$

```julia
function p3a(data)
    x = [deleteat!(copy(data), i) for i in 1:length(data)]
    p = histogram(vcat(x...))
    m = median.(x)
    sd = sqrt(sum((m .- mean(m)).^2) * (length(data)-1) / length(data))
    CI = (mean(data) - 1.96 * sd,
          mean(data) + 1.96 * sd)
    return (CI = CI, plot = p)
end  > p3a

a = p3a(data);  ✓
a.plot  Plot{Plots.GRBackend() n=1}
a.CI  (-0.0888…, 0.916…)
```

Confidence interval = (-0.0888, 0.916)



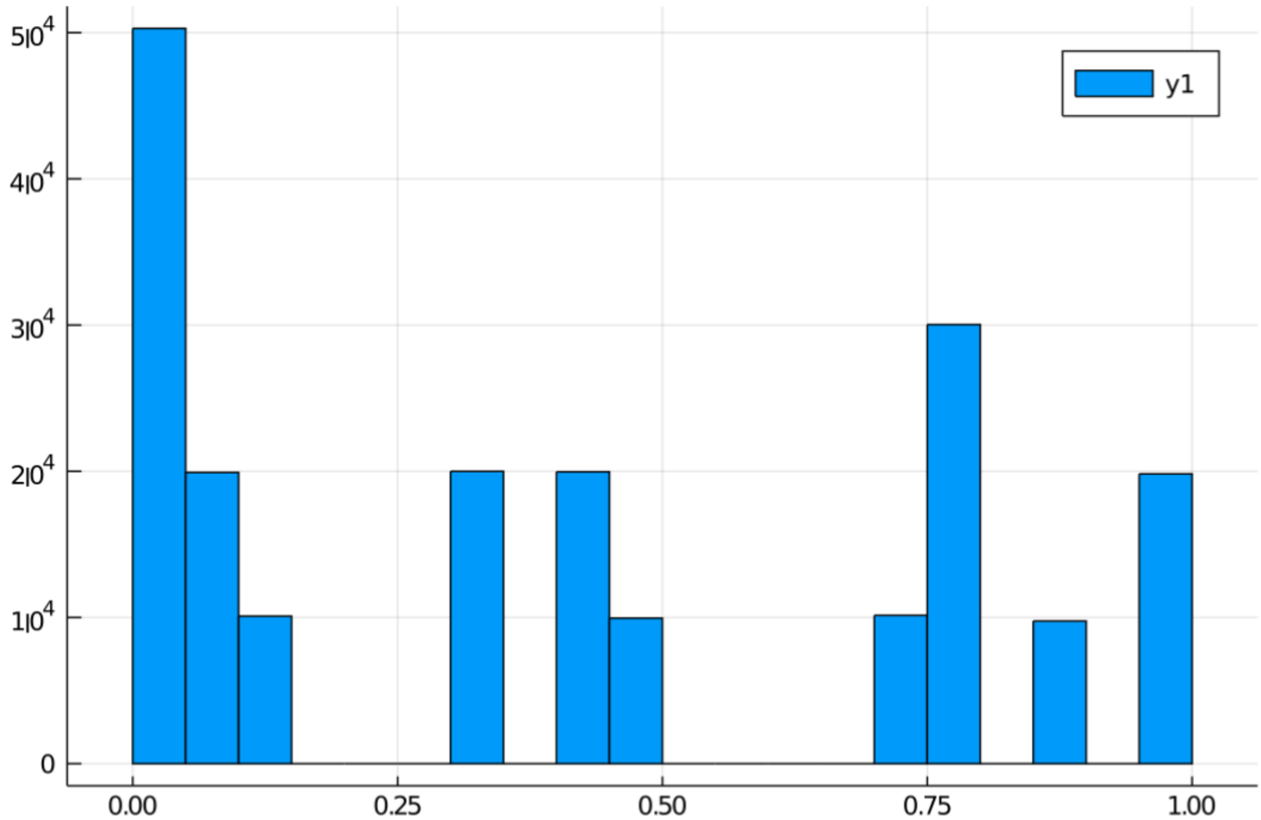
(b)

Bootstrap:

1. 原始資料為 $X = \{X_1, X_2, \dots, X_n\}$
2. $X_i^* = (X_{i1}, \dots, X_{in})$為$DiscreteUniform(\{X_1, X_2, \dots, X_n\})$的隨機值, $i = 1, \dots, B$
3. 可得$S(X_i^*), i = 1, \dots, B$, S 為我們有興趣的統計量
4. 計算 Bootstrap 結果的統計量的樣本變異數

```
function p3b(data)
    x = [data[rand(1:length(data), length(data))] for _ in 1:1e4]
    p = histogram(vcat(x...))
    m = median.(x)
    sd = std(m)
    θₒ = median(data)
    z = (m .- θₒ) ./ std.(x)
    t = sort(z)[[9750, 250]]
    CI = Tuple(θₒ .- t * std(data))
    return (CI = CI, plot = p)
end    > p3b

a = p3b(data);    ✓
a.plot    Plot{Plots.GRBackend() n=1}
a.CI    (0.0263…, 0.725…)
```

Confidence interval = (0.0263, 0.725)



(c)

Parametric bootstrap:
1. 給定分配族的分配函數G$(x;\boldsymbol{\theta})$
2. 利用原始資料求得 $\widehat{\boldsymbol{\theta}}$ (MLE)
3. 用估計的分配函數$\hat{G}(x;\hat{\theta})$模擬 B 組 size 為 n 的樣本
4. 可得 B 個 Bootstrap 的統計量及其變異數

```
function p3c(data)
    f = fit(Beta, data)
    x = [betainvcdf.(f.α, f.β, rand(length(data))) for _ in 1:1e4]
    p = histogram(vcat(x...))
    m = sort(median.(x))
    CI = (m[250], m[9750])
    return (CI = CI, plot = p)
end      > p3c

a = p3c(data);      ✓
a.plot      Plot{Plots.GRBackend() n=1}
a.CI      (0.0871…, 0.684…)
```

Confidence interval = (0.0871, 0.684)