Praktikum Komputasi Statistika I

Pertemuan 2

Transformasi Distribusi Variabel Random

Banyak metode untuk membangkitkan sampel dari distribusi tertentu. Pada praktikum ini hanya akan dipelajari satu metode yaitu *Inverse Cumulative Distribution Function Technique*. Metode ini merupakan metode yang paling sederhana dan mudah untuk digunakan. Dalam penggunaanya, metode ini memanfaatkan fungsi distribusi kumulatif (CDF) dari suatu distribusi untuk membangkitkan sampel random.

Misalkan diketahui suatu distribusi yang memiliki CDF yaitu $F(x)$, sifat dari distribusi kumulatif yaitu naik secara monoton dan nilainya terbatas pada interval [0,1]. Dengan batasan yang ada, kita dapat melakukan interpolasi terhadap nilai-nilai di fungsi kumulatif tersebut. Contoh sederhana, ingin dicari nilai c yang peluang kumulatifnya kurang dari 0.95 pada distribusi normal standar. Langkah yang dilakukan adalah mencari nilai kuantil dari distribusi normal standar pada titik 0.95, atau dituliskan$F^{-1}(x < c) = 0.95$ diperoleh nilai c = 1.644854.

Untuk sebarang fungsi distribusi kumulatif kontinu F, misalkan $U \sim Uniform(0,1)$, maka $X = F^{-1}(U)$ akan memiliki fungsi distribusi kumulatif yang sama dengan F.

Kasus Distribusi Variabel Kontinu

Akan dibangkitkan sampel random (n=10) berdistribusi eksponensial dengan parameter λ. Diketahui fungsi distribusi kumulatifnya yaitu $F(x) = 1 - \exp(-\lambda x)$. Misalkan $U \sim unif(0, 1)$, maka $X = F^{-1}(U)$ dapat ditransformasikan menjadi

$$1 - \exp(-\lambda x) = U$$

$$\exp(-\lambda x) = 1 - U$$

$$-\lambda x = \log(1 - U)$$

$$x = -\frac{1}{\lambda}\log(1 - U)$$

Kemudian dapat dilakukan komputasi untuk nilai x. Misalkan $\lambda = 4$. Berikut adalah *syntax* untuk komputasi contoh di atas.

```
bangkit.eksponensial=function(n,lambda){
 U=runif(n,0,1)
 x=-1/lambda*log(1-U)
 return(x)
}
bangkit.eksponensial(10,4)
```

Output

```
> bangkit.eksponensial(10,4)
 [1] 0.13147505 0.53643253 0.70530732 0.01165671 0.18774995 0.55737791
 [7] 0.20042543 0.15248417 0.78567174 0.15097939
```

Untuk meyakinkan bahwa sampel random yang kita bangkitkan adalah berdistribusi eksponensial dengan parameter $\lambda = 4$, dapat kita lakukan uji hipotesis menggunakan salah satu uji *Goodness of Fit* (GOF) yaitu *Kolmogorov-Smirnov Test*. Berikut *syntax* untuk uji tersebut.

```
data=bangkit.eksponensial(10,4)
ks.test(data,'pexp',4)
```

```
> ks.test(data,'pexp',4)

        One-sample Kolmogorov-Smirnov test

data:  data
D = 0.18954, p-value = 0.8017
alternative hypothesis: two-sided
```

Hipotesis awal (H_0) yakni data berdistribusi eksponensial dengan parameter $\lambda = 4$. Diperoleh nilai *p-value* > α = 0.05, maka hipotesis awal tidak ditolak sehingga dapat disimpulkan data berdistribusi eksponensial dengan parameter λ = 4.

Kasus Distribusi Variabel Diskrit

Misalkan X merupakan distribusi variabel random diskrit dengan *probability mass function* (pmf) $P(X = x_i) = p_i$. Proses membangkitkan bilangan random melalui invers transform:

1. Bangkitkan bilangan acak berdistribusi Uniform, $U \sim Uniform(0,1)$
2. Tentukan indeks *k* yang memenuhi $\sum_{j=1}^{k-1} p_j < U \leq \sum_{j=1}^{k} p_j$
3. Kembalikan nilai $X = x_k$

Contoh

Diketahui pmf dari variabel X sebagai berikut :

x_i	$P(X = x_i)$
1	0,1
2	0,4
3	0,2
4	0,3

Bangkitkan 100 bilangan acak dari distribusi tersebut lalu tunjukkan bahwa hasil random berdistribusi X!

Penyelesaian

1. Pertama, hitung nilai *Cumulative Distribution Function* (CDF) seperti di bawah ini :

x_i	$P(X = x_i)$	$\sum P(X = x_i)$
1	0,1	0,1
2	0,4	0,5
3	0,2	0,7
4	0,3	1

2. Berdasarkan tabel di atas, diperoleh informasi untuk membangkitkan bilangan random. Bila :
   $0 < U \leq 0{,}1$, maka $X = 1$
   $0{,}1 < U \leq 0{,}5$, maka $X = 2$
   $0{,}5 < U \leq 0{,}7$, maka $X = 3$
   $0{,}7 < U \leq 1$, maka $X = 4$
3. *Syntax*

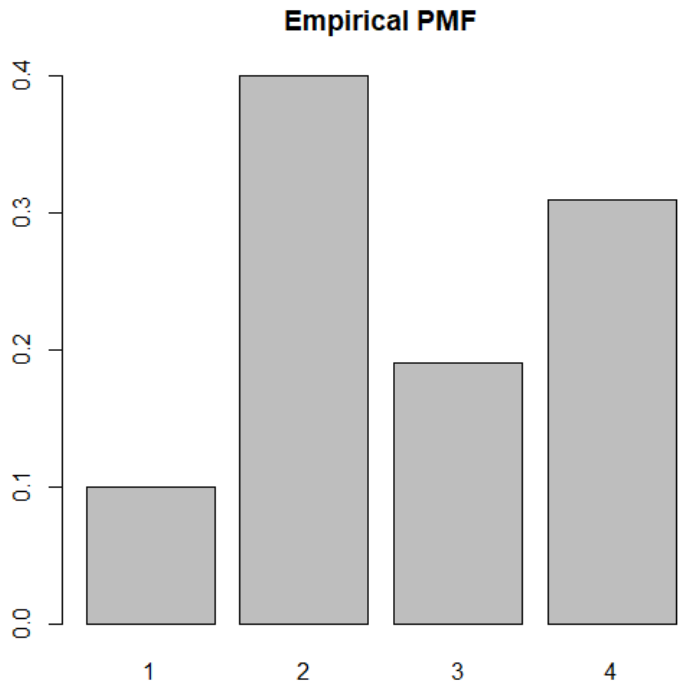
```
bangkit.diskrit=function(n){
      U=runif(n,0,1)
      x=NULL
      for(i in 1:n){
            if(U[i]<=0.1){
                  x[i]=1
            }
            else if(U[i]>0.1 && U[i]<=0.5){
                  x[i]=2
            }
            else if(U[i]>0.5 && U[i]<=0.7){
                  x[i]=3
            }
            else{
                  x[i]=4
            }
      }
return(x)
}
```

4. *Output*

```
> data=bangkit.diskrit(100)
> data
  [1] 4 4 4 2 2 1 1 2 4 3 4 1 2 3 4 4 4 2 4 2 1 3 2 1 3 1 3 4 4 4 2 4 2 2 3 2 2
 [38] 3 2 2 2 2 1 2 1 2 4 3 4 2 1 2 3 1 3 2 3 4 3 2 2 2 2 4 3 2 4 2 4 2 3 3 4 4
 [75] 1 4 3 4 4 3 2 4 1 4 4 3 3 3 4 1 4 3 2 2 4 2 2 2 2 2
```

Dengan menggunakan diagram batang, akan ditunjukkan bahwa data yang dibangkitkan berdistribusi *X*

```
> barplot(prop.table(table(data)),main="Empirical PMF")
```



Dapat dilihat bahwa proporsi setiap nilai x nilainya dekat dengan nilai *pmf x*. Maka data yang dibangkitkan dapat dikatakan berdistribusi *X*.

Secara praktisnya, untuk membangkitkan bilangan random dapat menggunakan fungsi yang telah tersedia pada R misalkan fungsi random ataupun menggunakan fungsi kuantil. Akan tetapi, pada praktikum ini lebih menekankan pada capaian memahami algoritma R untuk mengkomputasi kan masalah-masalah yang ada pada bidang Statistika.

Untuk distribusi-distribusi yang memiliki fungsi distribusi kumulatif yang lebih rumit, dapat menggunakan metode lainnya seperti untuk membangkitkan distribusi Normal dapat menggunakan metode *Box-Muller*. Ada juga metode-metode lain untuk membangkitkan sampel random yang distribusinya terlalu rumit, metode ini memanfaatkan simulasi monte carlo pada rantai markov, misalnya metode *Metropolis Hasting Algorithm* dan *Gibbs Sampling*.

PRAKTIKUM KOMPUTASI STATISTIKA I
PERTEMUAN 2

APLIKASI PENGGUNAAN METODE NEWTON RAPHSON UNTUK MENGESTIMASI MLE

Kasus 1:

Diketahui distribusi dengan satu parameter tak diketahui dengan fungsi densitas peluang $f(x|\theta)$. Akan diestimasi nilai dari parameter θ yaitu $\hat{\theta}$ dengan metode *Maximum Likelihood Estimator* (MLE). Misalkan ada sampel random independen $x = (X_1, X_2, \ldots, X_n)$ berdistribusi $f(x|\theta)$. Fungsi *likelihood* dari distribusi tersebut yaitu :

$$L(\theta) = \prod_{i=1}^{n} f(x_i|\theta) = f(x_1|\theta).f(x_2|\theta)\ldots f(x_n|\theta)$$

Definisikan fungsi log likelihood yaitu $l(\theta)$ = log $L(\theta)$. Metode MLE mencari nilai dari θ yang menghasilkan nilai maksimum pada fungsi $l(\theta)$. Ini berarti mencari akar dari turunan pertama fungsi tersebut disamadenganken nol atau $\frac{\partial}{\partial\theta}L(\theta) = l'(\theta) = 0$. Akar yang diperoleh kita sebut dengan $\hat{\theta}$, kemudian dicek apakah $\frac{\partial^2}{\partial\theta^2}L(\theta) = l''(\theta) < 0$ di titik $\theta = \hat{\theta}$. Apabila memenuhi, maka nilai $\hat{\theta}$ adalah MLE dari dari fungsi $f(x|\theta)$.

Untuk mencari nilai dari $\hat{\theta}$, kita akan menggunakan metode Newton Raphson dengan formula sebagai berikut

$$\hat{\theta}^{(n+1)} = \hat{\theta}^{(n)} - \frac{l'(\hat{\theta}^{(n)})}{l''(\hat{\theta}^{(n)})}$$

Di mana $\hat{\theta}^{(n)}$ merupakan nilai akar dari fungsi $l(\theta)$ pada iterasi ke-n. Nilai $\hat{\theta}^{(0)}$ ditentukan di awal. Proses dilakukan secara iteratif hingga selisih mutlak dari nilai $\hat{\theta}^{(n+1)}$ dan $\hat{\theta}^{(n)}$ kurang dari batas nilai ε yang telah ditentukan. Dalam kasus ini batas toleransi yang ditentukan yaitu $\varepsilon = 10^{-6}$. Secara matematis, proses akan berhenti apabila,

$$|\hat{\theta}^{(n+1)} - \hat{\theta}^{(n)}| < \varepsilon$$

Nilai $\hat{\theta}^{(n)}$ pada iterasi terakhir adalah MLE dari $f(x|\theta)$.

Catatan: pemilihan nilai awal $\hat{\theta}^{(0)}$ berpengaruh terhadap keseluruhan proses

Contoh #1

Diketahui sampel random berukuran 20 dari distribusi $EXP(\lambda)$. Diketahui fungsi densitas peluangnya yaitu $f(x|\lambda) = \lambda \exp(-\lambda x)$. Akan dicari estimasi parameter λ dengan metode MLE.

Langkah pertama : mendefinisikan fungsi log likelihood $l(\lambda)$ beserta turunannya.

$$l(\lambda) = log \prod_{i=1}^{20} f(x_i|\lambda) = \sum_{i=1}^{20} logf(x_i|\lambda)$$

$$l'(\lambda) = \frac{\partial}{\partial\lambda} l(\lambda) = \sum_{i=1}^{20} \frac{\partial}{\partial\lambda} logf(x_i|\lambda)$$

$$l''(\lambda) = \frac{\partial}{\partial\lambda} l'(\lambda) = \sum_{i=1}^{20} \frac{\partial^2}{\partial\lambda^2} logf(x_i|\lambda)$$

Langkah kedua : membuat komputasinya.

```
#fungsi log likelihood
loglik=expression(log(lambda*exp(-lambda*x)))
#turunan pertama
dloglik=D(loglik,"lambda")
dloglik
#turunan kedua
ddloglik=D(dloglik,"lambda")
ddloglik
```

Diperoleh output sebagai berikut :

```
> loglik=expression(log(lambda*exp(-lambda*x)))
> dloglik=D(loglik,"lambda")
> dloglik
(exp(-lambda * x) - lambda * (exp(-lambda * x) * x))/(lambda *
    exp(-lambda * x))
> ddloglik=D(dloglik,"lambda")
> ddloglik
-((exp(-lambda * x) * x + ((exp(-lambda * x) * x) - lambda *
    (exp(-lambda * x) * x * x)))/(lambda * exp(-lambda * x)) +
    (exp(-lambda * x) - lambda * (exp(-lambda * x) * x)) * (exp(-lambda *
        x) - lambda * (exp(-lambda * x) * x))/(lambda * exp(-lambda *
        x))^2)
```

Langkah ketiga : membuat komputasi fungsi MLE. Di dalam fungsi MLE tersebut, terdapat fungsi untuk nilai *dloglik* (turunan pertama) dan *ddloglik* (turunan kedua). Isi masing - masing fungsi merupakan hasil dari penurunan yang sudah dilakukan di kedua, sehingga nantinya output di *copy*

lalu *paste* ke dalam fungsinya. Karena fungsi akan mempengaruhi hasil fungsi MLE, pastikan syntax fungsi turunan sudah benar. Berikut syntaxnya :

```
#fungsi mle
mle.exp=function(x,lambda0){

##fungsi turunan pertama
dloglik=function(x,lambda){
      return(sum((exp(-lambda * x) - lambda * (exp(-lambda * x) *
x))/(lambda *
            exp(-lambda * x))))
}

##fungsi turunan kedua
ddloglik=function(x,lambda){
return(sum(-((exp(-lambda * x) * x + ((exp(-lambda * x) * x) - lambda *
      (exp(-lambda * x) * x * x)))/(lambda * exp(-lambda * x)) +
      (exp(-lambda * x) - lambda * (exp(-lambda * x) * x)) * (exp(-lambda
*
            x) - lambda * (exp(-lambda * x) * x))/(lambda * exp(-lambda
*
            x))^2)))
}

#batas toleransi
tol=10^(-6)

#inisiasi
iter=1
lambda=lambda0

#lambda hat
lambda.hat=lambda-dloglik(x,lambda)/ddloglik(x,lambda)

#proses iteratif
while (abs(lambda.hat-lambda)>tol) {
lambda=lambda.hat
lambda.hat=lambda-dloglik(x,lambda)/ddloglik(x,lambda)
iter=iter+1
}

#output
cat('Nilai estimasi lambda =',lambda.hat,'\n')
cat('Nilai diperoleh pada iterasi ke',iter,'\n')
}
```

Selanjutnya kita akan mengaplikasikan pada sampel random berukuran 20. Data sampel random yang dimiliki sebagai berikut : 0.038, 0.177, 0.131, 0.044, 0.048, 0.034, 0.088, 0.086, 0.298, 0.383, 0.283, 0.092, 0.044, 0.009, 0.075, 0.132, 0.013, 0.585, 0.180, 0.101. Akan dicari estimasi parameter λ dengan metode MLE ketika nilai awalnya yaitu 7, rata-rata dari sampel, dan 16. Berikut syntaxnya :

```
#data
data=c(0.038,0.177,0.131,0.044,0.048,0.034,0.088,0.086,0.298,0.383,0.2
83,
0.092,0.044,0.009,0.075,0.132,0.013,0.585,0.180,0.101)

#nilai awal = 7
mle.exp(data,7)

#nilai awal = rata - rata sampel
mle.exp(data,mean(data))

#nilai awal = 16
mle.exp(data,16)
```

Output yang diperoleh sebagai berikut :

```
> #nilai awal = 7
> mle.exp(data,7)
Nilai estimasi lambda = 7.039775
Nilai diperoleh pada iterasi ke 3
>
> #nilai awal = rata - rata sampel
> mle.exp(data,mean(data))
Nilai estimasi lambda = 7.039775
Nilai diperoleh pada iterasi ke 11
>
> #nilai awal = 16
> mle.exp(data,16)
Error in while (abs(lambda.hat - lambda) > tol) { :
  missing value where TRUE/FALSE needed
```

Dari output diatas, kita peroleh bahwa :

- MLE dari distribusi eksponensial menggunakan metode Newton Raphson dengan nilai awal 7 yaitu $\hat{\lambda}$= 7.039775. Nilai diperoleh pada iterasi ke-3.
- MLE dari distribusi eksponensial menggunakan metode Newton Raphson dengan nilai awal berupa rata-rata dari data yaitu $\hat{\lambda}$ = 7.039775. Nilai diperoleh pada iterasi ke-11.
- MLE dari distribusi eksponensial menggunakan metode Newton Raphson dengan nilai awal 16 tidak diperoleh karena proses tersebut tidak konvergen.

Untuk melakukan pengecekan nilai estimasi parameternya, dapat menggunakan library MASS dengan fungsi **`fitdistr()`**. Berikut syntaxnya :

```
#cek estimasi parameter
library(MASS)
fitdistr(data,'exponential')
```

Diperoleh output sebagai berikut :

```
> fitdistr(data,'exponential')
     rate
  7.039775
 (1.574141)
```

Dari hasil pengecekan estimasi parameter menggunakan fungsi fidsistr() diperoleh rate = 7.039775 , hasilnya sama dengan estimasi parameter dengan fungsi MLE yang dibuat.

Contoh #2

Distribusi poisson memiliki fungsi densitas peluang yaitu $f(x|\mu) = \frac{e^{-\mu}\mu^x}{x!}$. Akan dicari estimasi parameter μ pada sebuah data yang dibangkitkan dengan metode MLE. langkah -langkah yang dilakukan sama seperti pada contoh #1, yaitu mendefinisikan fungsi log likelihood $l(\lambda)$ beserta turunannya, membuat komputasi dari fungsi log likelihood beserta turunannya, kemudian membuat komputasi fungsi MLE. Berikut adalah syntaxnya :

```
#fungsi log likelihood
loglik=expression(log(exp(-mu)*(mu^(x))/factorial(x)))
#turunan pertama
dloglik=D(loglik,"mu")
dloglik
#turunan kedua
ddloglik=D(dloglik,"mu")
ddloglik

#fungsi mle
mle.poi=function(x,mu0){

##fungsi turunan pertama
dloglik=function(x,mu){
      return(sum((exp(-mu) * (mu^((x) - 1) * (x)) - exp(-mu) *
(mu^(x)))/factorial(x)/(exp(-mu) *
     (mu^(x))/factorial(x))))
}
```

```
##fungsi turunan kedua
ddloglik=function(x,mu){
return(sum((exp(-mu) * (mu^(((x) - 1) - 1) * ((x) - 1) * (x)) - exp(-
mu) *
    (mu^((x) - 1) * (x)) - (exp(-mu) * (mu^((x) - 1) * (x)) -
    exp(-mu)        *        (mu^(x))))/factorial(x)/(exp(-mu)        *
(mu^(x))/factorial(x)) -
    (exp(-mu) * (mu^((x) - 1) * (x)) - exp(-mu) * (mu^(x)))/factorial(x)
*
        ((exp(-mu)   *   (mu^((x)   -   1)   *   (x))   -   exp(-mu)   *
(mu^(x)))/factorial(x))/(exp(-mu) *
        (mu^(x))/factorial(x))^2))
}

#batas toleransi
tol=10^(-6)

#inisiasi
iter=1
mu=mu0

#mu hat
mu.hat=mu-dloglik(x,mu)/ddloglik(x,mu)

#proses iteratif
while(abs(mu.hat-mu)>tol){
mu=mu.hat
mu.hat=mu-dloglik(x,mu)/ddloglik(x,mu)
iter=iter+1
}

#output
cat('Nilai estimasi mu =',mu.hat,'\n')
cat('Nilai diperoleh pada iterasi ke',iter,'\n')
}
```

Selanjutnya kita akan mengaplikasikan pada sampel random yang dibangkitkan dari distribusi poisson, dengan n = 25 dan mu = 5 sebagai berikut :

```
#data
set.seed(77)
x=rpois(25,5)
```

Diperoleh output nya :

```
> x
 [1]  4  6  7  9  6  5  7  7  5  5  8 11  3  0  5  4  5  6  8  1  0  8  4  5  5
```

Akan dicari estimasi parameter μ dengan metode MLE ketika nilai awalnya yaitu 4, rata-rata dari sampel, dan 13. Berikut syntaxnya :

```
#nilai awal = 4
mle.poi(x,4)

#nilai awal = rata - rata sampel
mle.poi(x,mean(x))

#nilai awal = 12
mle.poi(x,12)
```

Diperoleh output sebagai berikut :

```
> #nilai awal = 4
> mle.poi(x,4)
Nilai estimasi mu = 5.36
Nilai diperoleh pada iterasi ke 5
> #nilai awal = rata - rata sampel
> mle.poi(x,mean(x))
Nilai estimasi mu = 5.36
Nilai diperoleh pada iterasi ke 1
> #nilai awal = 12
> mle.poi(x,12)
Error in while (abs(mu.hat - mu) > tol) { :
  missing value where TRUE/FALSE needed
```

Dari output diatas, kita peroleh bahwa :

- MLE dari distribusi poisson menggunakan metode Newton Raphson dengan nilai awal 4 yaitu $\hat{\mu}$= 5.36. Nilai diperoleh pada iterasi ke-5.
- MLE dari distribusi poisson menggunakan metode Newton Raphson dengan nilai awal berupa rata-rata dari data yaitu$\hat{\mu}$ = 5.36. Nilai diperoleh pada iterasi ke-1.
- MLE dari distribusi poisson menggunakan metode Newton Raphson dengan nilai awal 12 tidak diperoleh karena proses tersebut tidak konvergen.

Untuk melakukan pengecekan nilai estimasi parameternya, dapat menggunakan library MASS dengan fungsi **`fitdistr()`**. Berikut syntaxnya :

```
#cek estimasi parameter
library(MASS)
fitdistr(data,'poisson')
```

Diperoleh output sebagai berikut :

```
> fitdistr(x,'poisson')
    lambda
  5.3600000
 (0.4630335)
```

Dari hasil pengecekan estimasi parameter menggunakan fungsi fidsistr() diperoleh mu = 5.3600000 , hasilnya sama dengan estimasi parameter dengan fungsi MLE yang dibuat.

Kasus 2:

Diketahui fungsi distribusi dengan k parameter tak diketahui dengan fungsi densitas peluang $f(x|\theta_1, \theta_2, \ldots, \theta_k)$. Akan diestimasi nilai dari masing-masing parameter θ_i yaitu $\hat{\theta}_i$ dengan metode Maximum Likelihood Estimator (MLE). Misalkan ada sampel random independen $x = (X_1, X_2, \ldots, X_n)$ berdistribusi $f(x|\theta_1, \theta_2, \ldots, \theta_k)$. Fungsi likelihood dari distribusi tersebut yaitu

$$L(\Theta) = L(\theta_1, \theta_2, \ldots, \theta_k) = \prod_{i=1}^{n} f(x_i|\theta_1, \theta_2, \ldots, \theta_k)$$

Definisikan fungsi log likelihood yaitu $l(\Theta)$ = log $L(\Theta)$. Metode MLE mencari nilai dari Θ yang menghasilkan nilai maksimum pada fungsi $l(\Theta)$. Ini berarti mencari akar dari turunan pertama fungsi tersebut disamadengankan nol atau $\frac{\partial}{\partial \Theta} L(\Theta) = l'(\theta) = 0$ kemudian dicek apakah $\frac{\partial^2}{\partial \Theta^2} L(\Theta) = l''(\Theta) < 0$ di titik $\Theta = \hat{\Theta}$. Apabila memenuhi, maka nilai $\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_k$ adalah MLE dari dari fungsi $f(x|\theta_1, \theta_2, \ldots, \theta_k)$.

Untuk mencari nilai dari $\hat{\Theta}$, kita akan menggunakan metode Newton Raphson dengan formula sebagai berikut

$$\hat{\Theta}^{(n+1)} = \hat{\Theta}^{(n)} - \left(H^{(n)}\right)^{-1}\left(G^{(n)}\right)$$

Dengan

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_k \end{bmatrix} \qquad \mathbf{H} = \begin{bmatrix} \frac{\partial^2 L}{\partial \theta_1^2} & \cdots & \frac{\partial^2 L}{\partial \theta_1 \partial \theta_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 L}{\partial \theta_k \partial \theta_1} & \cdots & \frac{\partial^2 L}{\partial \theta_k^2} \end{bmatrix} \qquad \mathbf{G} = \begin{bmatrix} \frac{\partial L}{\partial \theta_1} \\ \frac{\partial L}{\partial \theta_2} \\ \vdots \\ \frac{\partial L}{\partial \theta_k} \end{bmatrix}$$

Vektor parameter | Matriks Hessian | Vektor Gradien

Di mana $\hat{\Theta}^{(n)}$ merupakan nilai akar dari fungsi $l(\Theta)$ pada iterasi ke-n. Nilai $\hat{\Theta}^{(0)}$ ditentukan di awal. Proses dilakukan secara iteratif hingga selisih mutlak dari nilai $\hat{\Theta}^{(n+1)}$ dan $\hat{\Theta}^{(n)}$ kurang dari batas

nilai ε yang telah ditentukan. Dalam kasus ini batas toleransi yang ditentukan yaitu $\varepsilon = 10^{-6}$. Secara matematis, proses akan berhenti apabila

$$|\hat{\theta}^{(n+1)} - \hat{\theta}^{(n)}| < \varepsilon$$

Nilai $\hat{\theta}^{(n)}$ pada iterasi terakhir adalah MLE dari $f(x|\theta)$.

Catatan: pemilihan nilai awal $\hat{\theta}^{(0)}$ berpengaruh terhadap keseluruhan proses

Contoh

Diketahui sampel random berukuran 25 dari distribusi $Gamma(\alpha, \beta)$. Diketahui fungsi densitas peluangnya yaitu

$$f(x|\alpha, \beta) = \frac{1}{\Gamma(\alpha)\beta^{\alpha}} x^{\alpha-1} e^{-x/\beta}$$

Akan dicari estimasi parameter $\theta = (\alpha, \beta)$ dengan metode MLE. Definisikan fungsi log likelihood $l(\theta)$ beserta turunannya yaitu vektor Gradien dan matriks Hessian.

Langkah pertama : mendefinisikan fungsi log likelihood $l(\theta)$ beserta turunannya yaitu vektor Gradien dan matriks Hessian.

$$l(\boldsymbol{\theta}) = \log \prod_{i=1}^{25} f(x_i|\boldsymbol{\theta}) = \sum_{i=1}^{25} \log f(x_i|\boldsymbol{\theta})$$

$$\mathbf{G} = \begin{bmatrix} \frac{\partial}{\partial \alpha} l(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \beta} l(\boldsymbol{\theta}) \end{bmatrix}$$

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2}{\partial \alpha^2} l(\boldsymbol{\theta}) & \frac{\partial^2}{\partial \alpha \partial \beta} l(\boldsymbol{\theta}) \\ \frac{\partial^2}{\partial \beta \partial \alpha} l(\boldsymbol{\theta}) & \frac{\partial^2}{\partial \beta^2} l(\boldsymbol{\theta}) \end{bmatrix}$$

Langkah kedua : membuat komputasinya.

```
#cara komputasi turunannya
loglik=expression(log(1/(gamma(a)*b^a)*x^(a-1)*exp(-x/b)))
dla=D(loglik,"a")
dla
ddla=D(dla,"a")
ddla
dlb=D(loglik,"b")
dlb
ddlb=D(dlb,"b")
ddlb
ddlab=D(dla,"b")
ddlab
```

Diperoleh potongan output sebagai berikut :

```
> loglik=expression(log(1/(gamma(a)*b^a)*x^(a-1)*exp(-x/b)))
> dla=D(loglik,"a")
> dla
(1/(gamma(a) * b^a) * (x^(a - 1) * log(x)) - (gamma(a) * digamma(a) *
    b^a + gamma(a) * (b^a * log(b)))/(gamma(a) * b^a)^2 * x^(a -
    1)) * exp(-x/b)/(1/(gamma(a) * b^a) * x^(a - 1) * exp(-x/b))
> ddla=D(dla,"a")
> ddla
(1/(gamma(a) * b^a) * (x^(a - 1) * log(x) * log(x)) - (gamma(a) *
    digamma(a) * b^a + gamma(a) * (b^a * log(b)))/(gamma(a) *
    b^a)^2 * (x^(a - 1) * log(x)) - ((((gamma(a) * digamma(a) *
    digamma(a) + gamma(a) * trigamma(a)) * b^a + gamma(a) * digamma(a) *
    (b^a * log(b)) + (gamma(a) * digamma(a) * (b^a * log(b)) +
    gamma(a) * (b^a * log(b) * log(b))))/(gamma(a) * b^a)^2 -
    (gamma(a) * digamma(a) * b^a + gamma(a) * (b^a * log(b))) *
        (2 * ((gamma(a) * digamma(a) * b^a + gamma(a) * (b^a *
            log(b))) * (gamma(a) * b^a)))/((gamma(a) * b^a)^2)^2) *
    x^(a - 1) + (gamma(a) * digamma(a) * b^a + gamma(a) * (b^a *
    log(b)))/(gamma(a) * b^a)^2 * (x^(a - 1) * log(x)))) * exp(-x/b)/(1/(gamma(a) *
    b^a) * x^(a - 1) * exp(-x/b)) - (1/(gamma(a) * b^a) * (x^(a -
    1) * log(x)) - (gamma(a) * digamma(a) * b^a + gamma(a) *
    (b^a * log(b)))/(gamma(a) * b^a)^2 * x^(a - 1)) * exp(-x/b) *
    ((1/(gamma(a) * b^a) * (x^(a - 1) * log(x)) - (gamma(a) *
        digamma(a) * b^a + gamma(a) * (b^a * log(b)))/(gamma(a) *
        b^a)^2 * x^(a - 1)) * exp(-x/b))/(1/(gamma(a) * b^a) *
    x^(a - 1) * exp(-x/b))^2
```

Langkah ketiga : membuat komputasi fungsi MLE nya, caranya analog dengan kasus 1, tetapi pada kasus ini ada lebih banyak fungsi yang dibuat yaitu dla, dlb, ddla, ddlb, ddlab. Ingat, nilai ddlab=ddlba menurut aturan turunan parsial. Berikut adalah syntaxnya :

```
#cara membuat fungsi MLE
mle.gamma=function(x,a0,b0){

dla=function(x,a,b){
return(sum((1/(gamma(a) * b^a) * (x^(a - 1) * log(x)) - (gamma(a) *
digamma(a)*b^a + gamma(a) * (b^a * log(b)))/(gamma(a) * b^a)^2 * x^(a -
1)) * exp(-x/b)/(1/(gamma(a) * b^a) * x^(a - 1) * exp(-x/b))))
}

dlb=function(x,a,b){
return(sum((1/(gamma(a) * b^a) * x^(a - 1) * (exp(-x/b) * (x/b^2)) -
gamma(a)*(b^(a - 1) * a)/(gamma(a) * b^a)^2 * x^(a - 1) * exp(-
x/b))/(1/(gamma(a) * b^a) * x^(a - 1) * exp(-x/b))))
}

ddla=function(x,a,b){
return(sum((1/(gamma(a) * b^a) * (x^(a - 1) * log(x) * log(x)) -
(gamma(a) * digamma(a) * b^a + gamma(a) * (b^a * log(b)))/(gamma(a) *
b^a)^2 * (x^(a - 1) * log(x)) - ((((gamma(a) * digamma(a) * digamma(a)
+ gamma(a) * trigamma(a)) * b^a + gamma(a) * digamma(a) * (b^a * log(b))
+ (gamma(a) * digamma(a) * (b^a * log(b)) + gamma(a) * (b^a * log(b) *
log(b))))/(gamma(a) * b^a)^2 - (gamma(a) * digamma(a) * b^a + gamma(a)
* (b^a * log(b))) * (2 * ((gamma(a) * digamma(a) * b^a + gamma(a) * (b^a
* log(b))) * (gamma(a) * b^a)))/((gamma(a) * b^a)^2)^2) * x^(a - 1) +
(gamma(a) * digamma(a) * b^a + gamma(a) * (b^a * log(b)))/(gamma(a) *
b^a)^2 * (x^(a - 1) * log(x)))) * exp(- x/b)/(1/(gamma(a) * b^a) * x^(a
- 1) * exp(-x/b)) - (1/(gamma(a) * b^a) * (x^(a - 1) * log(x)) - (gamma(a)
* digamma(a) * b^a + gamma(a) * (b^a * log(b)))/(gamma(a) * b^a)^2 *
x^(a - 1)) * exp(-x/b) * ((1/(gamma(a) * b^a) * (x^(a - 1) * log(x)) -
(gamma(a) * digamma(a) * b^a + gamma(a) * (b^a * log(b)))/(gamma(a) *
b^a)^2 * x^(a - 1)) * exp(-x/b))/(1/(gamma(a) * b^a) *
x^(a - 1) * exp(-x/b))^2))
}

ddlb=function(x,a,b){
return(sum((1/(gamma(a) * b^a) * x^(a - 1) * (exp(-x/b) * (x/b^2) *
(x/b^2) - exp(-x/b) * (x * (2 * b)/(b^2)^2)) - gamma(a) * (b^(a - 1) *
a)/(gamma(a) * b^a)^2 * x^(a - 1) * (exp(-x/b) * (x/b^2)) - ((gamma(a)
* (b^((a - 1) - 1) * (a - 1) * a)/(gamma(a) * b^a)^2 - gamma(a) * (b^(a
- 1) * a) * (2 * (gamma(a) * (b^(a - 1) * a) * (gamma(a) *
b^a)))/((gamma(a) * b^a)^2)^2) * x^(a - 1) * exp(-x/b) + gamma(a) * (b^(a
- 1) * a)/(gamma(a) * b^a)^2 * x^(a - 1) * (exp(-x/b) *
(x/b^2))))/(1/(gamma(a) * b^a) * x^(a - 1) * exp(-x/b)) - (1/(gamma(a)
* b^a) * x^(a - 1)    * (exp(-x/b) * (x/b^2)) - gamma(a) * (b^(a - 1) *
a)/(gamma(a) * b^a)^2 * x^(a - 1) * exp(-x/b)) * (1/(gamma(a) * b^a) *
```

```
x^(a - 1)  * (exp(-x/b) * (x/b^2)) - gamma(a) * (b^(a - 1) * a)/(gamma(a)
* b^a)^2 * x^(a - 1) * exp(-x/b))/(1/(gamma(a) * b^a) * x^(a - 1) *
exp(-x/b))^2))
}

ddlab=ddlba=function(x,a,b){
return(sum(((1/(gamma(a) * b^a) * (x^(a - 1) * log(x)) - (gamma(a) *
digamma(a)*b^a + gamma(a) * (b^a * log(b)))/(gamma(a) * b^a)^2 * x^(a -
1)) * (exp(-x/b) * (x/b^2)) - (gamma(a) * (b^(a - 1) * a)/(gamma(a) *
b^a)^2 * (x^(a - 1) * log(x)) + ((gamma(a) * digamma(a) *(b^(a - 1) *
a) + gamma(a) * (b^(a - 1) * a * log(b) + b^a * (1/b)))/(gamma(a) *
b^a)^2 - (gamma(a) * digamma(a) * b^a + gamma(a) * (b^a * log(b))) * (2
* (gamma(a) * (b^(a - 1) * a) * (gamma(a) * b^a)))/((gamma(a) * b^a)^2)^2)
* x^(a - 1)) * exp(-x/b))/(1/(gamma(a) * b^a) * x^(a - 1) * exp(-x/b))
- (1/(gamma(a) * b^a) * (x^(a - 1) * log(x)) - (gamma(a) * digamma(a) *
b^a + gamma(a) * (b^a * log(b)))/(gamma(a) * b^a)^2 * x^(a - 1)) * exp(-
x/b) * (1/(gamma(a) * b^a) * x^(a - 1) * (exp(-x/b) * (x/b^2)) - gamma(a)
* (b^(a - 1)    *  a)/(gamma(a)  *  b^a)^2  *  x^(a  -  1)  *  exp(-
x/b))/(1/(gamma(a) * b^a) * x^(a - 1) * exp(-x/b))^2))
}

tol=10^(-6)
iter=1
teta=matrix(c(a0,b0),nrow=2)
a=teta[1,1]
b=teta[2,1]

Gm=matrix(c(dla(x,a,b),dlb(x,a,b)),nrow=2)
Hm=matrix(c(ddla(x,a,b),ddlba(x,a,b),ddlab(x,a,b),ddlb(x,a,b)),nrow=2,
ncol=2)

teta.hat=teta-solve(Hm)%*%Gm
galat=abs(teta.hat-teta)
while (any(galat>tol)) {
teta=teta.hat
a=teta[1,1]
b=teta[2,1]
Gm=matrix(c(dla(x,a,b),dlb(x,a,b)),nrow=2)
Hm=matrix(c(ddla(x,a,b),ddlba(x,a,b),ddlab(x,a,b),ddlb(x,a,b)),nrow=2,
ncol=2)
teta.hat=teta-solve(Hm)%*%Gm
galat=abs(teta.hat-teta)
iter=iter+1
}
```

```
cat('Nilai estimasi alpha =',teta.hat[1,1],'\n')
cat('Nilai estimasi bera =',teta.hat[2,1],'\n')
cat('Nilai diperoleh pada iterasi ke',iter,'\n')
}
```

Selanjutnya kita akan mengaplikasikan pada data sampel random berikut ini. Akan dicari estimasi parameter α dan β dengan metode MLE. Ketika nilai awalnya (0.25,0.5) iterasi akan konvergen dan menghasilkan nilai estimasi parameter. Ingat, pemilihan nilai awal sangat berpengaruh. Kemudian dicoba ketika nilai awal (2,4) maka iterasi tidak konvergen. Berikut syntaxnya :

```
#data
data=c(1.043,0.235,0.708,1.426,1.055,0.561,0.693,1.113,0.317,0.542,0.3
28,
0.826,0.347,0.604,0.976,0.533,0.284,0.387,0.506,0.556,0.331,0.366,0.16
6,
0.347,0.426)

mle.gamma(data,0.25,0.5) #konvergen
mle.gamma(data,2,4) #tidak konvergen
```

Diperoleh output sebagai berikut :

```
> mle.gamma(data,0.25,0.5) #konvergen
Nilai estimasi alpha = 3.790138
Nilai estimasi bera = 0.1548862
Nilai diperoleh pada iterasi ke 11
> mle.gamma(data,2,4) #tidak konvergen
Error in solve.default(Hm) :
  system is computationally singular: reciprocal condition number = 1.38684e-16
```

Dari output diatas, kita peroleh bahwa :

- MLE dari distribusi gamma menggunakan metode Newton Raphson dengan nilai awal (0.25,0.5) yaitu $\widehat{\alpha}$= 3.790138 dan $\hat{\beta}$= 0.1548862. Nilai diperoleh pada iterasi ke-11.
- MLE dari distribusi gamma menggunakan metode Newton Raphson dengan nilai awal (2,4) tidak diperoleh karena proses tersebut tidak konvergen.

Untuk melakukan pengecekan nilai estimasi parameternya, dapat menggunakan library MASS dengan fungsi **`fitdistr()`**. Berikut syntaxnya :

```
#cek estimasi parameter
fitdistr(data,'gamma')
1/6.456548
```

Diperoleh output sebagai berikut :

```
> fitdistr(data,'gamma')
     shape        rate
  3.790256    6.456548
 (1.028354) (1.873124)
> 1/6.456548
[1] 0.1548815
```

Shape adalah α dan rate adalah $1/\beta$. Diperoleh estimasi menggunakan fungsi MLE yang dibuat sangat mendekati nilai estimasi dari fungsi fitdistr(), jadi hasil estimasi ini sudah dianggap baik.