DMMLE for Weibull model

Function for parameter estimation in 3-parameter Weibull model

```
###---------------------------------------------------------
### Function for parameter estimation in 3-parameter Weibull model
###---------------------------------------------------------
# x: a vector of positive data
# a0: initial guess for (shape, scale)
# method: DM2 (Duble modified MLE), KD (Modified MLE), CI (Corrected MLE)
# h: correction parameter for CI method

weibull_fit=function(x,a0=c(1,1), method="DM2", h=0.2){


  ###---------------------------------------------------------
  ### Required functions
  ###---------------------------------------------------------

  dweibull3p=function(x, param){
    shape=param[1]
    scale=param[2]
    loc=param[3]
    out=ifelse(x>loc, (shape/scale)*((x-loc)^(shape-1))*exp(-(1/scale)*(x-loc)^shape), 0)
    return(out)
  }



  ##----------------------------##
  ## DATA MODIFIED SCORE VECTOR ##
  ##----------------------------##
  score.dataMod<-function(z,param){
    n=length(z)

    mu<-min(z)   ## Location parameter
    a<-param[1] ## Shape parameter
    s<-param[2] ## Scale parameter
    u<-runif(n) ## Generating uniform data
    z<-sort(z,decreasing = F)
    x<-z[-which.min(z)]
    Salpha<- (n-1)/a+(1/a)*sum(log((x-mu)**a))-(1/s)*sum(log(x-mu)*(x-mu)**a)
    Ssigma<- -(n-1)/s+(1/s**2)*sum((x-mu)**a)
    score<- c(Salpha,Ssigma)
    return(score)
  }
  ##score.dataMod(y,rep(0.1,2))
```

```
##------------------------------------##
## DATA MODIFIED INFORMATION MATRIX ##
##------------------------------------##
InfoMat<-function(z,param){
  n=length(z)

  mu<-min(z)  ## Location parameter
  a<-param[1] ## Shape parameter
  s<-param[2] ## Scale parameter
  u<-runif(n) ## Generating uniform data
  #n<-length(z)
  z<-sort(z,decreasing = F)
  x<-z[-which.min(z)]
  ##d1gama2<-0.422784 ## First order derivative of the gamma(2) function
  ##d2gama2<-0.823681 ## Second order derivative of the gamma(2) function

  ## Computing the second order cumulants
  ##kaa<-((n-1)/a**2)*((1+d2gama2)+log(s)*(2*d1gama2+log(s)))
  ##kss<-(n-1)/s**2
  ##ksa<- -(n-1)*(d1gama2+log(s))/(a*s)
  kaa<- (n-1)/a**2+(1/s)*sum((x-mu)**a*(log(x-mu))**2)
  kss<- 2*sum((x-mu)**a)/s**3-(n-1)/s**2
  ksa<- -(1/s**2)*sum(log(x-mu)*(x-mu)**a)
  InfoMatrix<-matrix(NA,nrow = length(param), ncol = length(param))
  InfoMatrix[1,1]<-kaa
  InfoMatrix[2,2]<-kss
  InfoMatrix[1,2]<-InfoMatrix[2,1]<-ksa
  ##
  return(InfoMatrix)
}
##InfoMat(y,rep(0.1,2))
##
##--------------------------##
## FISHER SCORING ALGORITHM ##
##--------------------------##
FS.dataModified<-function(z,chute,maxiter=20,eps=1e-08){
  mu<-min(z)
  z<-sort(z,decreasing = F)
  x<-z[-which.min(z)]
  tolvec<-double()
  count<-1
  tolvec[1]<-toler<-1
  resAnter<-cbind(chute[1],chute[2])


  while(TRUE){
    if((resAnter[1]>0)&&(resAnter[2]>0)){
      V<-score.dataMod(z,resAnter)%*%solve(InfoMat(z,resAnter))

      ##----------------##
      ## Iterative Step ##
      ##----------------##
      resAtual<-resAnter+V
    }
    else
      return(NA)

    #print(resAtual)

    ##-------------------------##
    ## Computing the Tolerance ##
    ##-------------------------##
    toler<- sum(abs(resAtual-resAnter)/abs(resAnter))
    count<- count+1
    ##
    ##--------------------------##
    ## Updating some Quantities ##
    ##--------------------------##
    resAnter<-resAtual
    tolvec[count]<-toler
    ####cat("Wait: the FS algorithm is running", "\r")
    ##
    ##
    ##s.new<-sum((x-mu)**resAtual[1])/(n-1)
    ##
    if((eps>toler)|(count>maxiter))break
  }
  param.hat<-c(mu,resAnter)
  Hessian<-InfoMat(z,resAtual)
  return(list(Hessian=Hessian, tolvec=tolvec, param.hat=param.hat))
}
##
##
##
##
##
##-------------------------##
## DOBLY MODIFIED APPROACH ##
##-------------------------##
penaltyFunc<-function(z,param){
  n=length(z)

  mu<-min(z)  ## Location parameter
  a<-param[1] ## Shape parameter
  s<-param[2] ## Scale parameter
  u<-runif(n) ## Generating uniform data
  #n<-length(z)
  Info.da<-matrix(NA,nrow = length(param), ncol = length(param))
  Info.ds<-matrix(NA,nrow = length(param), ncol = length(param))

  ##--------------------------------##
  ## Calling the information matrix ##
  ##--------------------------------##
  InfoMatrix<-InfoMat(z,param)
  ##
  z<-sort(z,decreasing = F)
  x<-z[-which.min(z)]
  ##------------------------------------##
  ## Computing the third-order cumulants##
  ##------------------------------------##
  kaaa<- (1/s)*sum((x-mu)**a*(log(x-mu))**3)-(2/a**3)*(n-1)
  ksss<- (2/s**3)*(n-1)-(6/s**4)*sum((x-mu)**a)
  kaas<- kasa<-(-1/s**2)*sum((x-mu)**a*(log(x-mu))**2)
  kssa<- ksas<-(2/s**3)*sum((x-mu)**a*(log(x-mu)))

  ##--------------------------------------##
  ## Prime of I(phi) with respect of alpha##
  ##--------------------------------------##
  Info.da[1,1]<-kaaa
  Info.da[2,2]<-kssa
  Info.da[1,2]<-Info.da[2,1]<-kaas

  ##--------------------------------------##
  ## Prime of I(phi) with respect of sigma##
  ##--------------------------------------##
  Info.ds[1,1]<-kaas
  Info.ds[2,2]<-ksss
  Info.ds[1,2]<-Info.ds[2,1]<-kssa

  ##----------------------------##
  ## Computing the penalty term ##
  ##----------------------------##
  Aa<-sum(diag(solve(InfoMatrix)%*%Info.da))
  As<-sum(diag(solve(InfoMatrix)%*%Info.ds))
  penalty<-c(Aa,As)
  return(penalty)
}
##penaltyFunc(y,rep(0.1,2))

##--------------------------------##
## DOUBLY MODIFIED SCORE FUNCTION ##
##--------------------------------##
score.DoublyMod<-function(z,param){
  doublyModScore<-score.dataMod(z,param)+0.5*penaltyFunc(z,param)
  return(doublyModScore)
}
##score.DoublyMod(y,param)


##----------------------------------------------------##
## FISHER SCORING ALGORITHM FOR DOUBLY MODIFIED SCORE ##
##----------------------------------------------------##
FS.DoublyModified<-function(z,chute,maxiter=20,eps=1e-08){
  mu<-min(z)
  z<-sort(z,decreasing = F)
  x<-z[-which.min(z)]
  tolvec<-double()
  count<-1
  tolvec[1]<-toler<-1
  resAnter<-cbind(chute[1],chute[2])


  while(TRUE){

    if((resAnter[1]>0)&&(resAnter[2]>0)){
      V<-score.DoublyMod(z,resAnter)%*%solve(InfoMat(z,resAnter))

      ##----------------##
      ## Iterative Step ##
      ##----------------##
      resAtual<-resAnter+V
    }
    else
      return(NA)
    ##-------------------------##
    ## Computing the Tolerance ##
    ##-------------------------##
    toler<- sum(abs(resAtual-resAnter)/abs(resAnter))
    count<- count+1
    ##
    ##--------------------------##
    ## Updating some Quantities ##
    ##--------------------------##
    resAnter<-resAtual
    tolvec[count]<-toler
    ####cat("Wait: the FS algorithm is running", "\r")
    ##
    if((eps>toler)|(count>maxiter))break
  }
  doubModEstimates<-c(mu,resAnter)
  Hessian<-InfoMat(z,resAtual)
  return(list(Hessian=Hessian, tolvec=tolvec,doubModEstimates=doubModEstimates))
}


##----------------------------------------------------##
## scale estimator (MMLE) ##
##----------------------------------------------------##
sigma_est=function(x, shape){
  loc=min(x)
  n=length(x)
  out=sum((x[-which.min(x)]-loc)^shape)/(n-1)
  return(out)
}

##----------------------------------------------------##
## modified log-likelihood ##
##----------------------------------------------------##
llf=function(x, shape, scale){
  loc=min(x)
  n=length(x)
  out=(n-1)*log(shape)-(n-1)*log(scale)-(1/scale)*sum((x[-which.min(x)]-loc)^shape)+
    (shape-1)*sum(log((x[-which.min(x)]-loc)))
  return(out)
}


##----------------------------------------------------##
## minus modified log-likelihood function ##
##----------------------------------------------------##
llf_alpha=function(x, shape){
  -llf(x, shape, scale=sigma_est(x, shape))
}

##----------------------------------------------------##
## parameter estimation via Modified MLE (cf. [Kundu and Zaquab, 2009]) ##
##----------------------------------------------------##
weibull_fit_optim=function(x, a0=1){

  mu_est=min(x)

  estimated_params2 <- optim(c(1),
                             llf_alpha, x = x,
                             method = "L-BFGS-B",
                             lower = c(10^(-5)), upper = c(Inf)#Inf
  )

  a_est=estimated_params2$par
  s_est=sigma_est(x, a_est)

  llf_out=-estimated_params2$value

  return(list(shape_est=a_est,scale_est=s_est, loc_est=mu_est, llf_max=llf_out))

}

##----------------------------------------------------##
## corrected log-likelihood function (cf. [Cheng and Iles]) ##
##----------------------------------------------------##

llf_til=function(x, param, h){
  ##param=c(alpha,sigma)
  loc=min(x)
  n=length(x)
  y=x[-which.min(x)]

  out2=sum(log(dweibull3p(x=y, c(param,loc))))
  funcao_para_integral <- function(x) {

    return(dweibull3p(x, c(param,loc)))
  }

  out1=log(integrate(funcao_para_integral, lower = loc, upper = loc+h)$value)
```

```r
  return(out1+out2)
}

menosllf_til=function(x, param, h){
  -llf_til(x, param,h)
}

weibull_fit_optim2=function(x, h, a0=c(1,1)){

  mu_est=min(x)

  estimated_params2 <- optim(a0,
                             menosllf_til, x = x, h=h,
                             method = "L-BFGS-B",
                             lower = c(0,0), upper = c(Inf,Inf)#Inf
  )

  a_est=estimated_params2$par[1]
  s_est=estimated_params2$par[2]

  llf_out=-estimated_params2$value






  return(list(shape_est=a_est,scale_est=s_est, loc_est=mu_est, llf_max=llf_out))

}





#DM2: DMMLE (duble modified MLE)
#KD: MMLE (from Kundu and Zaqab)
#CI: CMLE (from Cheng and Iles)

if(method=="KZ"){
  out=weibull_fit_optim(x, a0[1])
}
if(method=="CI"){
  out=weibull_fit_optim2(x, h, a0)
}
if(method=="DM2"){
  ## Doubly Modified Estimates
  Obj.Doublymod<-FS.DoublyModified(x,a0)
  mu_est=Obj.Doublymod$doubModEstimates[1]
  a_est=Obj.Doublymod$doubModEstimates[2]
  s_est=Obj.Doublymod$doubModEstimates[3]
  llf_out=sum(log(dweibull3p(x[-which.min(x)], param = c(a_est, s_est, mu_est))))
  out=list(shape_est=a_est,scale_est=s_est, loc_est=mu_est, llf_max=llf_out)
}


dgama=function(z,n){
  integrand <- function(t){(t^(z-1))*(exp(-t))*((log(t))^n)}
  integralv<-integrate(integrand, 0, Inf)$value

  return (integralv)
}

mfisher_weibull3p=function(n,param){
  a=param[1]
  s=param[2]

  kaa=((n-1)/(a^2))*((1+(dgama(2,2)))+(log(s))*(2*(dgama(2,1)) + log(s)))
  kss=(n-1)/(s^2)
  ksa=-((n-1)*(dgama(2,1)+(log(s))))/(a*s)

  mI=matrix(c(kaa,ksa,ksa,kss), ncol=2, byrow=T)

  # Inverse of the matrix
  inv_mI <- solve(mI)
  # library("MASS")
  # inv_mI <- tryCatch({
  #   solve(mI)
  # }, error = function(e) {
  #
  #   ginv(mI)
  # })


  return(list(mI=mI,inv_mI=inv_mI))

}


### Comput Confidence Intervals

param_est = c(out$shape_est, out$scale_est)
# Calculating information matrix
info <- mfisher_weibull3p(length(x), param_est)
inv_mI <- info$inv_mI
#info <-  InfoMat(x, param_est)
#inv_mI <- solve(info)

# Standard errors
ep <- sqrt(diag(inv_mI))

# IC 95%
z <- qnorm(0.975)
IC_inf <- param_est - z * ep
```

```
  IC_sup <- param_est + z * ep

  # Results
  resultados <- data.frame(
    Parameters = c("Shape", "Scale"),
    Estimatites = param_est,
    SE = ep,
    CI_Inf = IC_inf,
    CI_Sup = IC_sup
  )



  out$results <- resultados


  #ll_max: maximum log-likelihood
  #n_par: number of parameters to be estimated
  #n: sample size
  aic<-function(ll_max, n_par){-2*ll_max+2*n_par}
  bic<-function(ll_max, n_par,n){-2*ll_max+n_par*log(n) }

  # Compute AIC
  out$AIC <- aic(out$llf_max, 3)
  # Compute BIC
  out$BIC <- bic(out$llf_max, 3, length(x))
  return(out)
}
```

Dataset for example 1

```
data_invest = c(8.958070, 25.150000, 5.015640, 11.006760, 5.279225, 5.013624, 6.435200,
                5.835900, 7.000000, 30.801862, 7.266010, 5.019073, 8.000000, 22.000000,
                5.514300, 5.302930, 7.968871, 6.000000, 7.058850, 15.000000, 5.155455,
                5.243266, 5.219835, 9.950000, 12.041000, 7.300000, 5.743445, 6.272980,
                5.070780, 6.529300, 6.000000, 8.697271, 46.067600, 17.171580, 6.098290,
                10.166405, 5.445229, 7.281530, 7.757070, 6.413050, 5.331970, 6.225743,
                14.399190, 5.850000, 5.653820, 5.484998, 5.257000, 5.841250, 8.265000,
                27.373000, 5.412000, 5.099400, 5.145000, 5.344151, 7.064000, 7.608040,
                6.346600, 5.052410, 5.623980, 14.615000, 5.358124, 5.210000, 68.570220,
                5.097540, 5.400696, 5.959290, 5.148630, 7.860000, 5.104935, 22.000000,
                8.985627, 5.176840, 5.177960, 5.040000, 7.419586, 15.634430, 5.146730,
                5.131308, 7.153940, 6.105075, 5.100000, 5.202800, 6.371200, 5.477140,
                5.309140, 5.776270, 6.301900, 5.100000, 12.041000, 5.123362, 5.133710,
                5.225000, 5.030000, 6.508635, 5.012000, 5.064560, 5.500000, 12.041000,
                5.115196, 13.318230, 5.147040, 7.895000, 5.050000, 5.400000, 6.441160,
                5.090940, 5.369000, 8.607168, 5.455010, 22.230579, 5.376720, 7.275880,
                13.920000, 5.225000, 31.122660, 12.4994, 5.342750, 5.177051, 7.556150,
                5.390042, 6.433160, 7.514053, 10.695323, 8.985627, 5.247600, 6.287972,
                16.052030, 5.659609, 5.100000, 6.098817, 6.101590, 5.281640)
```

Example 1

```
#DMMLE
weibull_fit(data_invest, method="DM2")
```

```
## $shape_est
## [1] 0.5653819
##
## $scale_est
## [1] 1.498483
##
## $loc_est
## [1] 5.012
##
## $llf_max
## [1] -250.8422
##
## $results
##   Parameters Estimatites         SE    CI_Inf    CI_Sup
## 1      Shape   0.5653819 0.03851518 0.4898935 0.6408702
## 2      Scale   1.4984826 0.15579388 1.1931322 1.8038330
##
## $AIC
## [1] 507.6845
##
## $BIC
## [1] 516.3329
```

```
#MMLE
weibull_fit(data_invest, method="KZ")
```

```
## $shape_est
## [1] 0.5690816
##
## $scale_est
## [1] 1.535023
##
## $loc_est
## [1] 5.012
##
## $llf_max
## [1] -250.8152
##
## $results
##   Parameters Estimatites         SE    CI_Inf   CI_Sup
## 1      Shape   0.5690816 0.03876722 0.4930993 0.645064
## 2      Scale   1.5350233 0.16097241 1.2195232 1.850523
##
## $AIC
## [1] 507.6304
##
## $BIC
## [1] 516.2788
```

```
#CMLE
weibull_fit(data_invest, method="CI")
```

```
## $shape_est
## [1] 0.5653053
##
## $scale_est
## [1] 1.515834
##
## $loc_est
## [1] 5.012
##
## $llf_max
## [1] -252.279
##
## $results
##   Parameters Estimatites         SE    CI_Inf    CI_Sup
## 1      Shape   0.5653053 0.03850997 0.4898272 0.6407835
## 2      Scale   1.5158342 0.15824542 1.2056788 1.8259895
##
## $AIC
## [1] 510.558
##
## $BIC
## [1] 519.2064
```

Dataset for example 2

```
#Data Set 2 (gauge lengths of 20 mm).
data_gauge=c(1.312, 1.314, 1.479, 1.552, 1.700, 1.803, 1.861, 1.865, 1.944, 1.958,
    1.966, 1.997, 2.006, 2.021, 2.027, 2.055, 2.063, 2.098, 2.140, 2.179,
    2.224, 2.240, 2.253, 2.270, 2.272, 2.274, 2.301, 2.301, 2.359, 2.382,
    2.382, 2.426, 2.434, 2.435, 2.478, 2.490, 2.511, 2.514, 2.535, 2.554,
    2.566, 2.570, 2.586, 2.629, 2.633, 2.642, 2.648, 2.684, 2.697, 2.726,
    2.770, 2.773, 2.800, 2.809, 2.818, 2.821, 2.848, 2.880, 2.954, 3.012,
    3.067, 3.084, 3.090, 3.096, 3.128, 3.233, 3.433, 3.585, 3.585)
```

Example 2

```
#DMMLE
weibull_fit(data_gauge, method="DM2")
```

```
## $shape_est
## [1] 2.34226
##
## $scale_est
## [1] 1.714474
##
## $loc_est
## [1] 1.312
##
## $llf_max
## [1] -51.72902
```

```
##
## $results
##   Parameters Estimatites        SE   CI_Inf   CI_Sup
## 1      Shape    2.342260 0.2214656 1.908195 2.776324
## 2      Scale    1.714474 0.2598860 1.205107 2.223841
##
## $AIC
## [1] 109.458
##
## $BIC
## [1] 116.1604
```

```
#MMLE
weibull_fit(data_gauge, method="KZ")
```

```
## $shape_est
## [1] 2.380407
##
## $scale_est
## [1] 1.801475
##
## $loc_est
## [1] 1.312
##
## $llf_max
## [1] -51.67451
##
## $results
##   Parameters Estimatites        SE   CI_Inf   CI_Sup
## 1      Shape    2.380407 0.2250725 1.939273 2.821541
## 2      Scale    1.801475 0.2782145 1.256185 2.346766
##
## $AIC
## [1] 109.349
##
## $BIC
## [1] 116.0513
```

```
#CMLE
weibull_fit(data_gauge, method="CI")
```

```
## $shape_est
## [1] 2.275162
##
## $scale_est
## [1] 1.704633
##
## $loc_est
## [1] 1.312
##
## $llf_max
## [1] -55.98713
##
## $results
##   Parameters Estimatites        SE   CI_Inf   CI_Sup
## 1      Shape    2.275162 0.2151214 1.853532 2.696792
## 2      Scale    1.704633 0.2578387 1.199279 2.209988
##
## $AIC
## [1] 117.9743
##
## $BIC
## [1] 124.6766
```