Statistical Computing

610611104 Pan,Hsing-Cheng

HW-7

Newton’s method:

利用Newton’s method計算它的gradient的根。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Initial value | Iteration times | distance | Gradient value | Xn | Error |
| (0,3) | 18 | 9.93237e-12 | (2.2085722e-16,  -4.4419599e-16) | (1.990913897,  0.99545644) |  |

停止條件設為第n次迭代值跟(n-1)次迭代值差距小於error或迭代超過150次。

Steepest descent method:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Initial value | Iteration times | distance | function value | Xn | Error |
| (0,3) | 151 | 0.00031268821 | 1.89557e-5 | (2.06552947,  1.0324054655) |  |

停止條件設為第n次迭代值跟(n-1)次迭代值差距小於error或迭代超過150次。因為它的距離不夠小所以會等到迭代次數超過才會停止，不過可以發現函數值已經小於，而且迭代完的Xn跟牛頓法的值是差不多的。

牛頓跟最陡下降法跟(1)是差不多的作法。

Newton’s method:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Initial value | Iteration times | distance | Gradient value | Xn | Error |
| (0,3) | 3 | 5.32700334e-11 | (4.4408855e-16,  -4.4408936e-16) | (1.9999980,  0.9999985) |  |

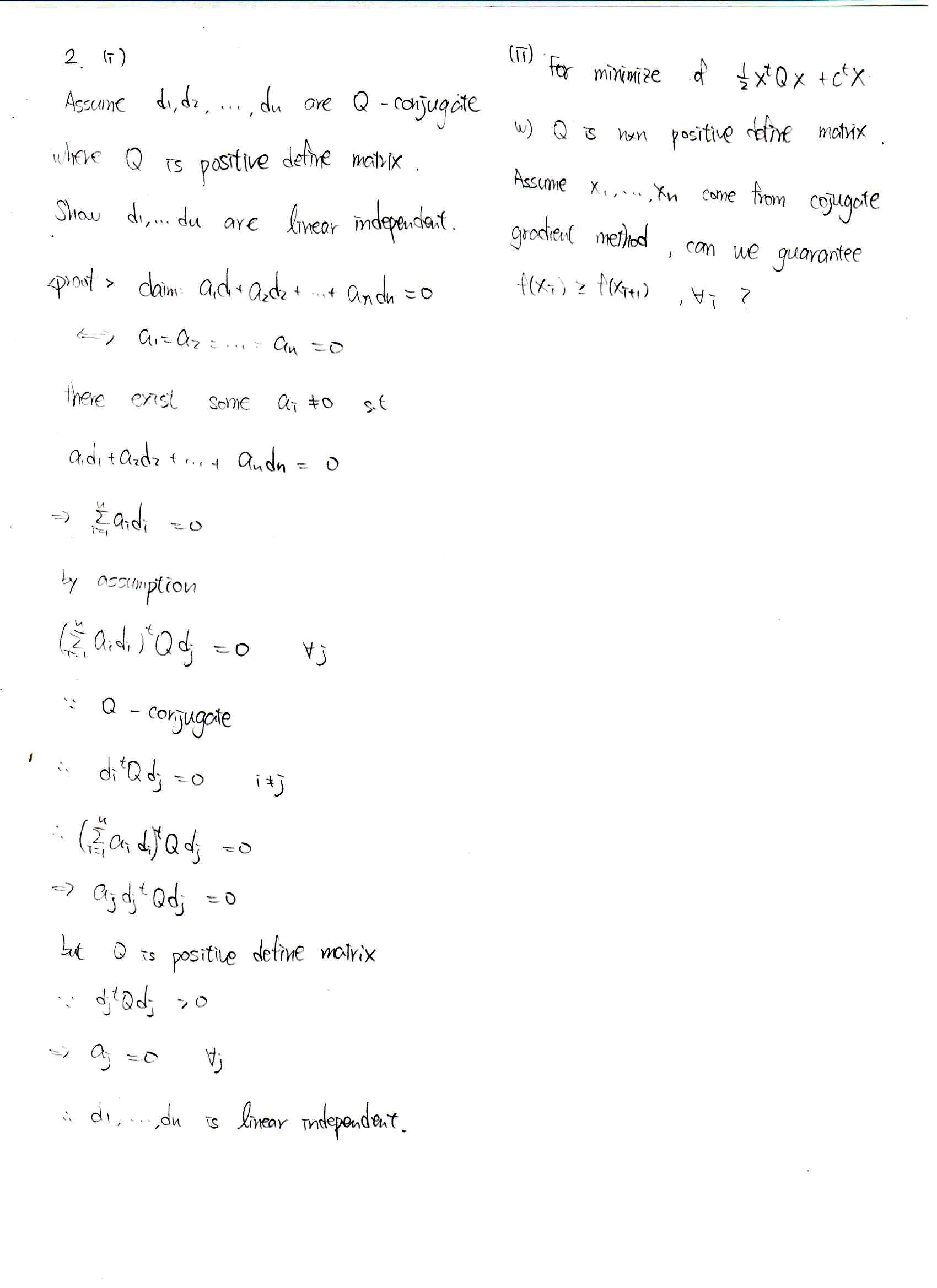
Steepest descent method:

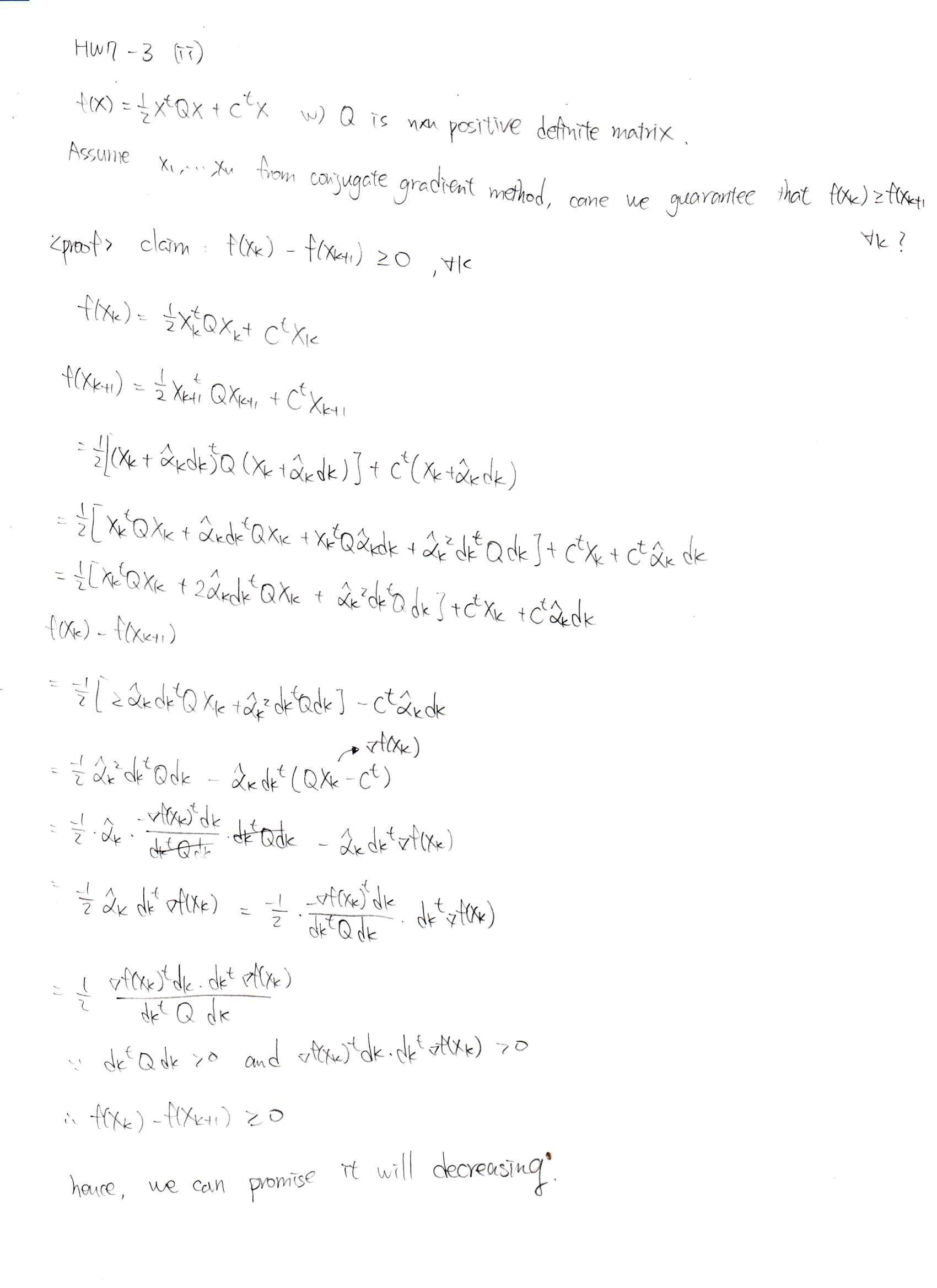
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Initial value | Iteration times | distance | function value | Xn | Error |
| (0,3) | 15 | 1.231471e-11 | 3.936401e-13 | (1.9999994,  0.9999997) |  |

Conjugate Gradient method:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Initial value | iteration | distance | Function value | Xn |
| (0,3) | 2 | 0.542107097 | 0.0054402239 | (1397143653,  1.01971956) |
| (0,3) | 73 | 2.4885938e-12 | 5e-12 | (1.999998,0.999998) |

因為定理告訴我們應該會n步就收斂，所以這裡的停止條件用n小於2就停止，而函數值跟X的值跟上面方法做得滿接近的。如果用第n次迭代值跟(n-1)次迭代值的距離小於當停止條件，會迭代到73次才停止。





Code:

|  |
| --- |
| rm(list=ls())  graphics.off()  ###############  n<-1  f<-function(x1,x2){  (x1-2)^2+(x1-2\*x2)^2  }  h<-10^-6  df.x1<-function(x1,x2){  y<-(f(x1+h,x2)-f(x1,x2))/h  return(y)  }  df.x2<-function(x1,x2){  y<-(f(x1,x2+h)-f(x1,x2))/h  return(y)  }  df.x1.x1<-function(x1,x2){  y<-(df.x1(x1+h,x2)-df.x1(x1,x2))/h  return(y)  }  df.x1.x2<-function(x1,x2){  y<-(df.x1(x1,x2+h)-df.x1(x1,x2))/h  return(y)  }  df.x2.x1<-function(x1,x2){  y<-(df.x2(x1+h,x2)-df.x2(x1,x2))/h  return(y)  }  df.x2.x2<-function(x1,x2){  y<-(df.x2(x1,x2+h)-df.x2(x1,x2))/h  return(y)  }  gradient<-function(x1,x2){  y<-matrix(c(df.x1(x1,x2),df.x2(x1,x2)),nrow = 2)  return(y)  }  hessian<-function(x1,x2)  {  y<-matrix(c(df.x1.x1(x1,x2),df.x1.x2(x1,x2),df.x2.x1(x1,x2),df.x2.x2(x1,x2)),nrow = 2,byrow = TRUE)  return(y)  }  mini<-function(x1,x2,alpha){  y<-((x1-alpha\*gradient(x1,x2)[1,])-2)^2+((x1-alpha\*gradient(x1,x2)[1,])-2\*(x2-alpha\*gradient(x1,x2)[2,]))^2  return(y)  }  dmini<-function(x1,x2,alpha){  y<-(mini(x1,x2,alpha+h)-mini(x1,x2,alpha))/h  return(y)  }  walk<-function(x1,x2){  conv<-FALSE  n<-1  h<-10^-6  xa<-0  while(!conv){  xb<-xa  xn<-xb-dmini(x1,x2,xb)/((dmini(x1,x2,xb+h)-dmini(x1,x2,xb))/h)  xa<-xn  if(abs(mini(x1,x2,xa))<10^-10 | n>150){conv<-TRUE}  # cat("now is",n,"iteration and x value is:",xa,"and f(x) is:",f(xa),"\n")  n<-n+1  }  return(xa)  }  ######  #steepest descent method  #####  error<-10^-10  xb<-matrix(c(0,3),nrow=2)  conv<-FALSE  while (!conv){  xn<-xb-walk(xb[1,],xb[2,])\*gradient(xb[1,],xb[2,])  cat("the length of walk is:",walk(xb[1,],xb[2,]),"\n")  xa<-xn  r<-sqrt((xb[1,]-xa[1,])^2+(xb[2,]-xa[2,])^2)  cat("now iteration time is:",n,"the distance of value is:",r,"the function value is:",f(xa[1,],xa[2,]),"\n")  if(r<error| n>150) (conv<-TRUE)  xb<-xa  n<-n+1  }  ######  #Newton's method  #####  error<-10^-10  xb<-matrix(c(0,3),nrow=2)  conv<-FALSE  while (!conv){  xn<-xb-solve(hessian(xb[1,],xb[2,]))%\*%gradient(xb[1,],xb[2,])  xa<-xn  r<-sqrt(sum((xb-xa)^2))  cat("now iteration time is:",n,"the distance of value is:",r,"the function value is:",gradient(xa[1,],xa[2,]),"\n")  if(r<error| n>150) (conv<-TRUE)  xb<-xa  n<-n+1  }  #####  #Conjugate Gradient method  #####  error<-10^-10  Q<-matrix(c(4,-4,-4,8),nrow = 2,byrow = TRUE)  xb<-matrix(c(0,3),nrow=2)  d<- -gradient(xb[1,],xb[2,])  conv<-FALSE  while(!conv){  dk<-d  alpha\_hat<- (t(-gradient(xb[1,],xb[2,]))%\*%dk)/(t(dk)%\*%Q%\*%dk)  alpha\_hat<-as.numeric(alpha\_hat)  xn<-xb+alpha\_hat\*dk  xa<-xn  lk<- (t(-gradient(xa[1,],xa[2,]))%\*%Q%\*%dk)/(t(dk)%\*%Q%\*%dk)  lk<-as.numeric(lk)  d<- -gradient(xa[1,],xa[2,])+lk\*dk  r<-sqrt(sum((xb-xa)^2))  cat("now iteration time is:",n,"the distance of value is:",r,"the function value is:",f(xa[1,],xa[2,]),"\n")  xb<-xa  if(r<error| n>150) (conv<-TRUE)  n<-n+1  } |