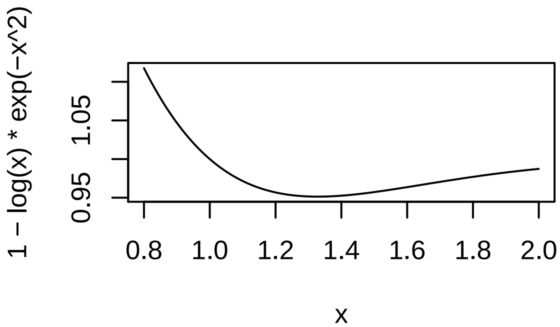
Assignment_2

Saurav Singh

28 February 2017

```
par(mfrow=c(2,2))
curve(1 - log(x) * exp(-x^2),0.8,2)
```



1. Unconstrained optimization in one dimension (f(x) =1 - log(x) * exp(-x^2).)

Newton's Method

```
newton <- function(f, tol=1E-12,x0=1,N=20) {
  h <- 0.001
  i <- 1; x1 <- x0
  p <- numeric(N)
  while (i<=N) {
    df.dx <- (f(x0+h)-f(x0))/h
    x1 <- (x0 - (f(x0)/df.dx))
    p[i] <- x1
    i <- i + 1
    if (abs(x1-x0) < tol) break
    x0 <- x1
  }
  return(p[1:(i-1)])
}
f <-function(x) { {4*x^2 + 5 * x^3} }
p <- newton(f, x0=1, N=10)
p
```

```
##  [1] 0.609018722 0.358938307 0.203837583 0.111601773 0.059294687
##  [6] 0.030899785 0.015986740 0.008318030 0.004418590 0.002441218
```

Golden-Section Search

```
f <- function(x) {1 - log(x) * exp(-x^2)}
golden.section.search = function(f, a, b, tolerance)
{
  golden.ratio = 2/(sqrt(5) + 1)

  ### Use the golden ratio to set the initial test points
  x1 = b - golden.ratio*(b - a)
  x2 = a + golden.ratio*(b - a)
```

```
### Evaluate the function at the test points
f1 = f(x1)
f2 = f(x2)

iteration = 0

while (abs(b - a) > tolerance)
{
  iteration = iteration + 1
  cat('', '\n')
  cat('Iteration #', iteration, '\n')
  cat('f1 =', f1, '\n')
  cat('f2 =', f2, '\n')

  if (f2 > f1)
    # then the minimum is to the left of x2
    # let x2 be the new upper bound
    # let x1 be the new upper test point
  {
    cat('f2 > f1', '\n')
    ### Set the new upper bound
    b = x2
    cat('New Upper Bound =', b, '\n')
    cat('New Lower Bound =', a, '\n')
    ### Set the new upper test point
    ### Use the special result of the golden ratio
    x2 = x1
    cat('New Upper Test Point = ', x2, '\n')
    f2 = f1

    ### Set the new lower test point
    x1 = b - golden.ratio*(b - a)
    cat('New Lower Test Point = ', x1, '\n')
    f1 = f(x1)
  }
  else
  {
    cat('f2 < f1', '\n')
    # the minimum is to the right of x1
    # let x1 be the new lower bound
    # let x2 be the new lower test point

    ### Set the new lower bound
    a = x1
    cat('New Upper Bound =', b, '\n')
    cat('New Lower Bound =', a, '\n')

    ### Set the new lower test point
    x1 = x2
    cat('New Lower Test Point = ', x1, '\n')

    f1 = f2
```

```r
      ### Set the new upper test point
      x2 = a + golden.ratio*(b - a)
      cat('New Upper Test Point = ', x2, '\n')
      f2 = f(x2)
    }
  }

  ### Use the mid-point of the final interval as the estimate of the optimzer
  cat('', '\n')
  cat('Final Lower Bound =', a, '\n')
  cat('Final Upper Bound =', b, '\n')
  estimated.minimizer = (a + b)/2
  cat('Estimated Minimizer =', estimated.minimizer, '\n')
}
golden.section.search(f,0,3,0.0000001)
```

```
##
## Iteration # 1
## f1 = 0.9633667
## f2 = 0.9801575
## f2 > f1
## New Upper Bound = 1.854102
## New Lower Bound = 0
## New Upper Test Point =  1.145898
## New Lower Test Point =  0.7082039
##
## Iteration # 2
## f1 = 1.208942
## f2 = 0.9633667
## f2 < f1
## New Upper Bound = 1.854102
## New Lower Bound = 0.7082039
## New Lower Test Point =  1.145898
## New Upper Test Point =  1.416408
##
## Iteration # 3
## f1 = 0.9633667
## f2 = 0.9531783
## f2 < f1
## New Upper Bound = 1.854102
## New Lower Bound = 1.145898
## New Lower Test Point =  1.416408
## New Upper Test Point =  1.583592
##
## Iteration # 4
## f1 = 0.9531783
## f2 = 0.9625577
## f2 > f1
## New Upper Bound = 1.583592
## New Lower Bound = 1.145898
## New Upper Test Point =  1.416408
## New Lower Test Point =  1.313082
##
## Iteration # 5
```

```
## f1 = 0.9514301
## f2 = 0.9531783
## f2 > f1
## New Upper Bound = 1.416408
## New Lower Bound = 1.145898
## New Upper Test Point =  1.313082
## New Lower Test Point =  1.249224
##
## Iteration # 6
## f1 = 0.9532662
## f2 = 0.9514301
## f2 < f1
## New Upper Bound = 1.416408
## New Lower Bound = 1.249224
## New Lower Test Point =  1.313082
## New Upper Test Point =  1.352549
##
## Iteration # 7
## f1 = 0.9514301
## f2 = 0.9515269
## f2 > f1
## New Upper Bound = 1.352549
## New Lower Bound = 1.249224
## New Upper Test Point =  1.313082
## New Lower Test Point =  1.28869
##
## Iteration # 8
## f1 = 0.9518106
## f2 = 0.9514301
## f2 < f1
## New Upper Bound = 1.352549
## New Lower Bound = 1.28869
## New Lower Test Point =  1.313082
## New Upper Test Point =  1.328157
##
## Iteration # 9
## f1 = 0.9514301
## f2 = 0.95137
## f2 < f1
## New Upper Bound = 1.352549
## New Lower Bound = 1.313082
## New Lower Test Point =  1.328157
## New Upper Test Point =  1.337474
##
## Iteration # 10
## f1 = 0.95137
## f2 = 0.9513944
## f2 > f1
## New Upper Bound = 1.337474
## New Lower Bound = 1.313082
## New Upper Test Point =  1.328157
## New Lower Test Point =  1.322399
##
## Iteration # 11
```

```
## f1 = 0.951378
## f2 = 0.95137
## f2 < f1
## New Upper Bound = 1.337474
## New Lower Bound = 1.322399
## New Lower Test Point =  1.328157
## New Upper Test Point =  1.331716
##
## Iteration # 12
## f1 = 0.95137
## f2 = 0.9513739
## f2 > f1
## New Upper Bound = 1.331716
## New Lower Bound = 1.322399
## New Upper Test Point =  1.328157
## New Lower Test Point =  1.325958
##
## Iteration # 13
## f1 = 0.9513709
## f2 = 0.95137
## f2 < f1
## New Upper Bound = 1.331716
## New Lower Bound = 1.325958
## New Lower Test Point =  1.328157
## New Upper Test Point =  1.329517
##
## Iteration # 14
## f1 = 0.95137
## f2 = 0.9513707
## f2 > f1
## New Upper Bound = 1.329517
## New Lower Bound = 1.325958
## New Upper Test Point =  1.328157
## New Lower Test Point =  1.327317
##
## Iteration # 15
## f1 = 0.95137
## f2 = 0.95137
## f2 < f1
## New Upper Bound = 1.329517
## New Lower Bound = 1.327317
## New Lower Test Point =  1.328157
## New Upper Test Point =  1.328677
##
## Iteration # 16
## f1 = 0.95137
## f2 = 0.9513701
## f2 > f1
## New Upper Bound = 1.328677
## New Lower Bound = 1.327317
## New Upper Test Point =  1.328157
## New Lower Test Point =  1.327836
##
## Iteration # 17
```

```
## f1 = 0.9513699
## f2 = 0.95137
## f2 > f1
## New Upper Bound = 1.328157
## New Lower Bound = 1.327317
## New Upper Test Point =  1.327836
## New Lower Test Point =  1.327638
##
## Iteration # 18
## f1 = 0.9513699
## f2 = 0.9513699
## f2 < f1
## New Upper Bound = 1.328157
## New Lower Bound = 1.327638
## New Lower Test Point =  1.327836
## New Upper Test Point =  1.327959
##
## Iteration # 19
## f1 = 0.9513699
## f2 = 0.9513699
## f2 > f1
## New Upper Bound = 1.327959
## New Lower Bound = 1.327638
## New Upper Test Point =  1.327836
## New Lower Test Point =  1.327761
##
## Iteration # 20
## f1 = 0.9513699
## f2 = 0.9513699
## f2 < f1
## New Upper Bound = 1.327959
## New Lower Bound = 1.327761
## New Lower Test Point =  1.327836
## New Upper Test Point =  1.327883
##
## Iteration # 21
## f1 = 0.9513699
## f2 = 0.9513699
## f2 < f1
## New Upper Bound = 1.327959
## New Lower Bound = 1.327836
## New Lower Test Point =  1.327883
## New Upper Test Point =  1.327912
##
## Iteration # 22
## f1 = 0.9513699
## f2 = 0.9513699
## f2 > f1
## New Upper Bound = 1.327912
## New Lower Bound = 1.327836
## New Upper Test Point =  1.327883
## New Lower Test Point =  1.327865
##
## Iteration # 23
```

```
## f1 = 0.9513699
## f2 = 0.9513699
## f2 > f1
## New Upper Bound = 1.327883
## New Lower Bound = 1.327836
## New Upper Test Point =  1.327865
## New Lower Test Point =  1.327854
##
## Iteration # 24
## f1 = 0.9513699
## f2 = 0.9513699
## f2 < f1
## New Upper Bound = 1.327883
## New Lower Bound = 1.327854
## New Lower Test Point =  1.327865
## New Upper Test Point =  1.327872
##
## Iteration # 25
## f1 = 0.9513699
## f2 = 0.9513699
## f2 > f1
## New Upper Bound = 1.327872
## New Lower Bound = 1.327854
## New Upper Test Point =  1.327865
## New Lower Test Point =  1.327861
##
## Iteration # 26
## f1 = 0.9513699
## f2 = 0.9513699
## f2 < f1
## New Upper Bound = 1.327872
## New Lower Bound = 1.327861
## New Lower Test Point =  1.327865
## New Upper Test Point =  1.327868
##
## Iteration # 27
## f1 = 0.9513699
## f2 = 0.9513699
## f2 > f1
## New Upper Bound = 1.327868
## New Lower Bound = 1.327861
## New Upper Test Point =  1.327865
## New Lower Test Point =  1.327864
##
## Iteration # 28
## f1 = 0.9513699
## f2 = 0.9513699
## f2 > f1
## New Upper Bound = 1.327865
## New Lower Bound = 1.327861
## New Upper Test Point =  1.327864
## New Lower Test Point =  1.327863
##
## Iteration # 29
```

```
## f1 = 0.9513699
## f2 = 0.9513699
## f2 < f1
## New Upper Bound = 1.327865
## New Lower Bound = 1.327863
## New Lower Test Point =  1.327864
## New Upper Test Point =  1.327864
##
## Iteration # 30
## f1 = 0.9513699
## f2 = 0.9513699
## f2 > f1
## New Upper Bound = 1.327864
## New Lower Bound = 1.327863
## New Upper Test Point =  1.327864
## New Lower Test Point =  1.327863
##
## Iteration # 31
## f1 = 0.9513699
## f2 = 0.9513699
## f2 < f1
## New Upper Bound = 1.327864
## New Lower Bound = 1.327863
## New Lower Test Point =  1.327864
## New Upper Test Point =  1.327864
##
## Iteration # 32
## f1 = 0.9513699
## f2 = 0.9513699
## f2 < f1
## New Upper Bound = 1.327864
## New Lower Bound = 1.327864
## New Lower Test Point =  1.327864
## New Upper Test Point =  1.327864
##
## Iteration # 33
## f1 = 0.9513699
## f2 = 0.9513699
## f2 > f1
## New Upper Bound = 1.327864
## New Lower Bound = 1.327864
## New Upper Test Point =  1.327864
## New Lower Test Point =  1.327864
##
## Iteration # 34
## f1 = 0.9513699
## f2 = 0.9513699
## f2 < f1
## New Upper Bound = 1.327864
## New Lower Bound = 1.327864
## New Lower Test Point =  1.327864
## New Upper Test Point =  1.327864
##
## Iteration # 35
```

```
## f1 = 0.9513699
## f2 = 0.9513699
## f2 < f1
## New Upper Bound = 1.327864
## New Lower Bound = 1.327864
## New Lower Test Point =  1.327864
## New Upper Test Point =  1.327864
##
## Iteration # 36
## f1 = 0.9513699
## f2 = 0.9513699
## f2 > f1
## New Upper Bound = 1.327864
## New Lower Bound = 1.327864
## New Upper Test Point =  1.327864
## New Lower Test Point =  1.327864
##
## Final Lower Bound = 1.327864
## Final Upper Bound = 1.327864
## Estimated Minimizer = 1.327864
```

Unconstrained optimization in multiple dimensions ($f(x1, x2) = exp(0.1 * ((x2 - x1^2))^2 + 0.05 * (1 - x1)^2)$ *using starting point x(0) = [-0.3, 0.8] and the default tolerance for the convergence test 10-6.)*

Steepest Descent Method

```
library("pracma")
```

```
##
## Attaching package: 'pracma'

## The following object is masked _by_ '.GlobalEnv':
##
##     newton
```

```
dummy <- function(x)
{
  z <- x[1]
  y <- x[2]
  rez <- exp(0.1 * ((y - z^2))^2 + 0.05*(1 - z)^2)
  rez
}
n <- 0
eps <- 1
a <- 0.09
x <- c(-0.3,0.8)
#Computation loop
while (eps > 1e-10  && n<100)
{
gradf <- grad(dummy,x)
eps <- abs(gradf)+abs(gradf)
y <- x- a * gradf
x <- y
n <- n+1
}
#display end values
print(n)
```

```
## [1] 100
```

```r
print(x)
```

```
## [1] 0.4738503 0.1973280
```

```r
print(eps)
```

```
## [1] 0.098699077 0.009511314
```

Newton Method for Unconstrained optimisation in n dimension

```r
library("pracma")
dummy <- function(x)
{
  z <- x[1]
  y <- x[2]
  rez <- exp(0.1 * ((y - z^2))^2 + 0.05*(1 - z)^2)
  rez
}
n <- 0
eps <- 1
x <- c(1,1)

#Computation loop
while (eps>1e-10 && n<100 )
{
  gradf <- grad(dummy , x )
  eps <- abs(gradf)+abs(gradf)
  Hf <- hessian(dummy,x)
  k <- solve(Hf,-gradf)
  y <- x + k
  x <- y
  n <- n+1
}
print(n)
```

```
## [1] 1
```

```r
print(x)
```

```
## [1] 1 1
```

Quasi-Newton for unconstrained optimisation in n dimension

```r
library("pracma")
dummy <- function(x)
{
  z <- x[1]
  y <- x[2]
  rez <- exp(0.1 * ((y - z^2))^2 + 0.05*(1 - z)^2)
  rez
}
x <- c(-0.3,0.8)
a <- 0.09
B0 <- hessian(dummy, x)
while (eps>1e-10 && n<100 )
```

```
{
  grad1 <- grad(dummy,x)
  eps <- abs(grad1)+abs(grad1)
  Bk <- hessian(dummy,y)
  p <- solve(Bk, -grad1)
  s <- a * p
  y <-  x + s
  x <- y
  grad2 <- grad(dummy, y)
  yk <- grad2 - grad1
  ykt <- transpose(yk)
  Bk1 <- (Bk + (yk * ykt)/(ykt * s) - (Bk * s)* transpose((Bk * s))/(transpose(s) * Bk* s))
  n <- n+1
}
print(n)
```

```
## [1] 1
```

```
print(y)
```

```
## [1] 1 1
```

Direct search methods (Nelder-Mead simplex direct search) $(f(x1,x2) = ((x1^2 + x2 - 11)^2 + ((x1 + x2^2 - 7)^2)$ **using starting point x(0) = [0,-2].**

```
 library("neldermead")
```

```
## Loading required package: optimbase

## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:pracma':
##
##     expm, lu, tril, triu

##
## Attaching package: 'optimbase'

## The following objects are masked from 'package:pracma':
##
##     ones, size, zeros

## Loading required package: optimsimplex

##
## Attaching package: 'neldermead'

## The following objects are masked from 'package:pracma':
##
##     fminbnd, fminsearch
```

```
banana <- function(x){
  z <- x[1]
  y <- x[2]
  rez <- exp(0.1 * ((y - z^2))^2 + 0.05*(1 - z)^2)
  rez
}
```

```r
opt <- optimset(MaxIter=10)
sol <- fminsearch(banana, c(0,-2), opt)
```

```
## fminsearch:  Exiting: Maximum number of iterations has been exceeded
##          - increase MaxIter option.
##          Current function value: 1.04100200551003
```

```r
# Final Answer
sol
```

```
##
## Number of Estimated Variable(s): 2
##
## Estimated Variable(s):
##   Initial     Final
## 1       0  0.118125
## 2      -2 -0.100000
##
## Cost Function:
## function (x = NULL, index = NULL, fmsfundata = NULL)
## {
##     fminsearch <- list(f = fmsfundata$Fun(x), index = index,
##         this = list(costfargument = fmsfundata))
##     return(fminsearch)
## }
## <environment: namespace:neldermead>
##
## Cost Function Argument(s):
## $Fun
## function (x)
## {
##     z <- x[1]
##     y <- x[2]
##     rez <- exp(0.1 * ((y - z^2))^2 + 0.05 * (1 - z)^2)
##     rez
## }
##
## attr(,"class")
## [1] "optimbase.functionargs"
##
## Optimization:
## - Status: "maxiter"
## - Initial Cost Function Value: 1.568312
## - Final Cost Function Value: 1.041002
## - Number of Iterations (max): 10 (10)
## - Number of Function Evaluations (max): 20 (400)
##
## Simplex Information:
##
## - Simplex at Initial Point:
## Dimension: n=2
## Number of vertices: nbve=3
##   Vertex #1/3 : fv=1.568312e+00, x=0.000000e+00 -2.000000e+00
##   Vertex #2/3 : fv=1.567176e+00, x=7.500000e-03 -2.000000e+00
##   Vertex #3/3 : fv=1.633949e+00, x=0.000000e+00 -2.100000e+00
```

```
##
## - Simplex at Optimal Point:
## Dimension: n=2
## Number of vertices: nbve=3
##   Vertex #1/3 : fv=1.041002e+00, x=1.181250e-01 -1.000000e-01
##   Vertex #2/3 : fv=1.042524e+00, x=1.225781e-01 -1.625000e-01
##   Vertex #3/3 : fv=1.045303e+00, x=1.385156e-01 2.875000e-01
##
## Nelder-Mead Object Definition:
## List of 53
##  $ method                  : chr "variable"
##  $ simplex0method          : chr "pfeffer"
##  $ simplex0length          : num 1
##  $ simplexsize0            : num 0.1
##  $ historysimplex          : list()
##  $ coords0                 : NULL
##  $ rho                     : num 1
##  $ chi                     : num 2
##  $ gamma                   : num 0.5
##  $ sigma                   : num 0.5
##  $ tolfstdeviation         : num 0
##  $ tolfstdeviationmethod   : logi FALSE
##  $ tolsimplexizeabsolute   : num 1e-04
##  $ tolsimplexizerelative   : num 2.22e-16
##  $ tolsimplexizemethod     : logi FALSE
##  $ toldeltafv              : num 1e-04
##  $ tolssizedeltafvmethod   : logi TRUE
##  $ simplex0deltausual      : num 0.05
##  $ simplex0deltazero       : num 0.0075
##  $ restartsimplexmethod    : chr "oriented"
##  $ restartmax              : num 3
##  $ restarteps              : num 2.22e-16
##  $ restartstep             : num 1
##  $ restartnb               : num 0
##  $ restartflag             : logi FALSE
##  $ restartdetection        : chr "oneill"
##  $ kelleystagnationflag    : logi FALSE
##  $ kelleynormalizationflag: logi TRUE
##  $ kelleystagnationalpha0 : num 1e-04
##  $ kelleyalpha             : num 1e-04
##  $ startupflag             : logi TRUE
##  $ boxnbpoints             : chr "2n"
##  $ boxnbpointseff          : num 0
##  $ boxineqscaling          : num 0.5
##  $ checkcostfunction       : logi FALSE
##  $ scalingsimplex0         : chr "tox0"
##  $ guinalphamin            : num 1e-05
##  $ boxboundsalpha          : num 1e-06
##  $ boxtermination          : logi FALSE
##  $ boxtolf                 : num 1e-05
##  $ boxnbmatch              : num 5
##  $ boxkount                : num 0
##  $ boxreflect              : num 1.3
##  $ tolvarianceflag         : logi FALSE
```

```
## $ tolabsolutevariance    : num 0
## $ tolrelativevariance    : num 2.22e-16
## $ variancesimplex0       : num 0
## $ mymethod               : NULL
## $ myterminate            : NULL
## $ myterminateflag        : logi FALSE
## $ greedy                 : logi FALSE
## $ output                 :List of 4
##   ..$ algorithm : chr "Nelder-Mead simplex direct search"
##   ..$ funcCount : num 20
##   ..$ iterations: num 10
##   ..$ message   : chr "Optimization terminated:\n the current x satisfies the termination criteria us
## $ exitflag               : logi FALSE
## - attr(*, "class")= chr "neldermead"
```