Assignment -1

Saurav Singh

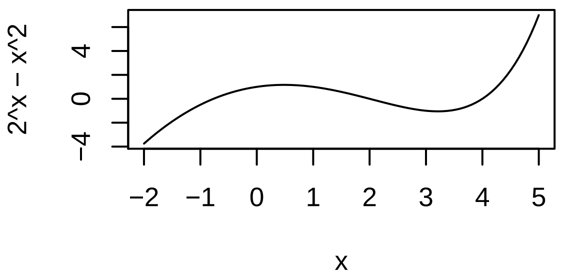
29 January 2017

Introduction to Linear and Non Linear Optimization.

A.Finding the roots of f(x)=0.

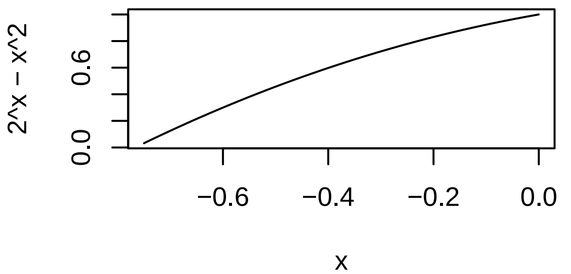
1. Naive Approach ($f(x) = 2^x - x^2$ for x belongs to [-2,5])

```
par(mfrow=c(2,2))
curve(2^x-x^2,-2,5)
```



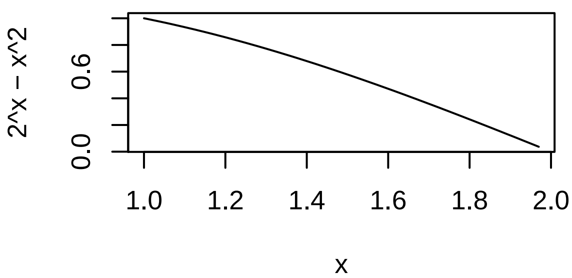
- graphical solution

```
par(mfrow=c(2,2))
curve(2^x-x^2,-0.75,0)
```



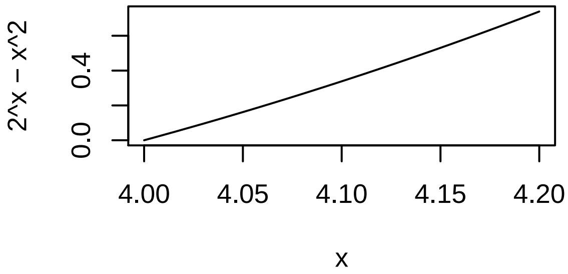
So,The first root is -0.75

```
par(mfrow=c(2,2))
curve(2^x-x^2,1,1.97)
```



The second root is 1.97

```r
par(mfrow=c(2,2))
curve(2^x-x^2,4,4.2)
```



And the last root is 4.00

- random search

```r
f<-function(x){2^x-x^2}
randomsearch<-function(a,b,f){
  for(i in range(a,b)){
    c<-(a+b)/2
    if (f(c)*f(a)<0){
      b<-c
    }
    else
    {
      a<-c
    }
  }
  c
}
randomsearch(-2,0,f)
```

```
## [1] -0.5
```

```r
randomsearch(0,3,f)
```

```
## [1] 2.25
```

```r
randomsearch(3,5,f)
```

```
## [1] 4.5
```

2. Bracketing ($f(x) = cos(1/x^2)$ for x belongs to [0.3,0.9])

```r
par(mfrow=c(2,2))
curve(cos(1/x^2),0.3,0.9)
f<- function(x){cos(1/x^2)}
bracket<-function(a,b,f,iter=400){
  for(i in c(1:iter)){
    c<-(a+b)/2
    if (f(c)*f(a)<0){
      b<-c
    }
    else
    {
      a<-c
    }
  }
```

```
  c
}
bracket(0.3,0.34,f)
```

```
## [1] 0.301572
```

```
bracket(0.34,0.4,f)
```

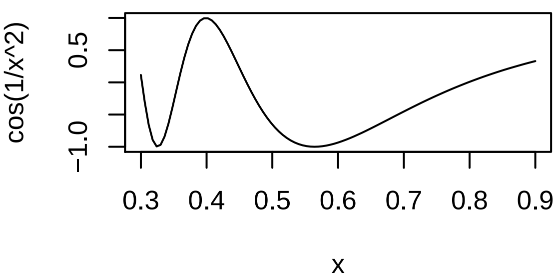
```
## [1] 0.3568248
```

```
bracket(0.4,0.6,f)
```

```
## [1] 0.4606589
```

```
bracket(0.6,0.9,f)
```

```
## [1] 0.7978846
```



3. Bisection ($f(x) = cos(1/x^2)$ for x belongs to [0.3,0.9])
   As We can see from the graph it has 4 roots.

```
bisection<-function (f, a, b, tol = 0.001, m = 100)
{
    iter <- 0
    f.a <- f(a)
    f.b <- f(b)
    while (abs(b - a) > tol) {
        iter <- iter + 1
        if (iter > m) {
            warning("maximum number of iterations exceeded")
            break
        }
        xmid <- (a + b)/2
        ymid <- f(xmid)
        if (f.a * ymid > 0) {
            a <- xmid
            f.a <- ymid
        }
        else {
            b <- xmid
            f.b <- ymid
        }
    }
    root <- (a + b)/2
    return(root)
}
par(mfrow=c(2,2))
curve(cos(1/x^2),0.3,0.9)
```

```
f <- function(x) {cos(1/x^2)}
bisection(f,0.3,0.33)
```

```
## [1] 0.3014062
```

```
bisection(f,0.33,0.4)
```

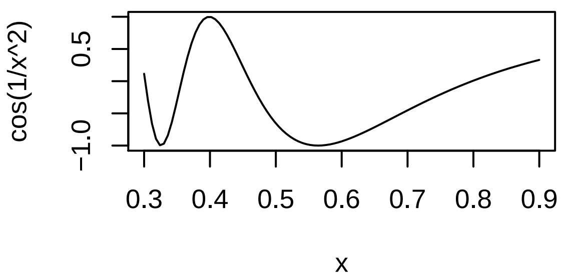
```
## [1] 0.3570703
```

```
bisection(f,0.4,0.6)
```

```
## [1] 0.4605469
```

```
bisection(f,0.6,0.9)
```

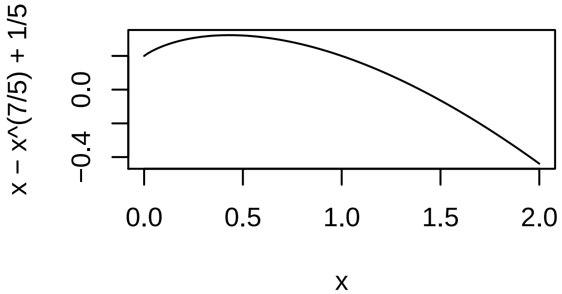
```
## [1] 0.7977539
```



4. Fixed Point Method ($f(x) = x - x^{7/5} + 1/5$ for x_0=0.2)

```
fixedpoint <- function(fun, x0, tol=1e-07, niter=500){
  xold <- x0
  xnew <- fun(xold)
  for (i in 1:niter) {
    xold <- xnew
    xnew <- fun(xold)
    if ( abs((xnew-xold)) < tol )
      return(xnew)
  }
  stop("exceeded allowed number of iterations")
}
par(mfrow=c(2,2))
curve(x-x^(7/5)+1/5,0,2)
f <- function(x) {x - x^7/5 + 1/5}
gfun <- function(x) {x-x+x^7/5 + 1/5}
x=fixedpoint(gfun,0.2)
f(x)
```

```
## [1] 0.4
```



5. Newton's Method ($f(x) = e^{-x}log(x) - x^2 + x^3/3 + 1$ for x_0=2.750,0.805,0.863 and 1.915)

```
newton<-function (f, fp, x, tol = 0.001, m = 100)
{
    iter <- 0
    oldx <- x
    x <- oldx + 10 * tol
    while (abs(x - oldx) > tol) {
        iter <- iter + 1
        if (iter > m)
            stop("No solution found")
        oldx <- x
        x <- x - f(x)/fp(x)
    }
    return(x)
}
par(mfrow=c(2,2))
curve(exp(-x)*log(x)-x^2+((x^3)/3)+1,0.1,3)
f <- function(x) {exp(-x)*log(x)-x^2+((x^3)/3)+1}
fp <- function(x) {-exp(-x)*log(x)-2*x+x^2+exp(-x)/x}
newton(f,fp,2.750)
```

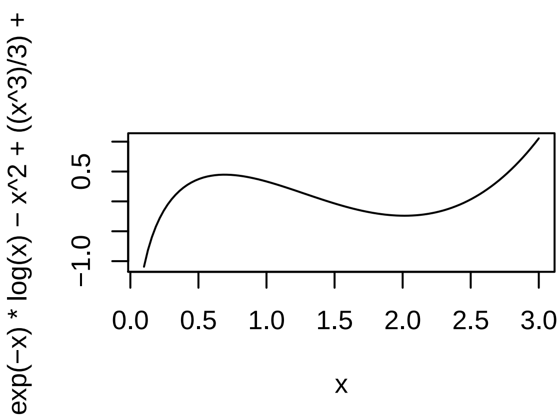
```
## [1] 2.471198
```

```
newton(f,fp,0.805)
```

```
## [1] 2.471198
```

```
newton(f,fp,0.863)
```

```
## [1] 1.451199
```



B.Solving system of linear Equations.

Consider the matrix A, the vector b, and the solution of the linear system Ax = b given below.

```
A<-matrix(floor(rnorm(25,1,2)),5,5)
b<-c(floor(rnorm(5)))
A
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    2    1    1    3   -1
## [2,]   -2    0    1    1    3
```

```
## [3,]    1    1    0    0    2
## [4,]    2    4    3    2    5
## [5,]   -2    3    0   -2    0
```

```r
b
```

```
## [1] 0 0 0 1 0
```

```r
solve(A,b)
```

```
## [1]  0.12500000 -0.08333333  0.56250000 -0.25000000 -0.02083333
```

Direct Methods

1. LU Factorization.

```r
A <- matrix( c ( 1, 2, 2, 1 ), nrow=2, byrow=TRUE)
B <- matrix( c( 2, 6, 3, 8 ), nrow=2, byrow=TRUE )

luDecomposition <-function (A)
  {
    nCOL <- ncol(A)
    U <- matrix(rep(0, each = nCOL * nCOL), nrow = nCOL, byrow = T)
    L <- matrix(rep(0, each = nCOL * nCOL), nrow = nCOL, byrow = T)
    U[upper.tri(A, diag = TRUE)] <- A[upper.tri(A, diag = TRUE)]
    L[lower.tri(A, diag = FALSE)] <- A[lower.tri(A, diag = FALSE)]
    diag(L) <- 1
    return(list(U = U, L = L))
}
luA <- luDecomposition(A)
L <- luA$L
U <- luA$U
print( L )
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    2    1
```

```r
print( U )
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    0    1
```

```r
y <- solve(L,B)
print( y )
```

```
##      [,1] [,2]
## [1,]    2    6
## [2,]   -1   -4
```

```r
x <- solve(U,y)
print( x )
```

```
##      [,1] [,2]
## [1,]    4   14
## [2,]   -1   -4
```

2. Cholesky Factorization.

```
A<-matrix(floor(rnorm(25,1,2)),5,5)
B<-c(floor(rnorm(5)))
print(A)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    2   -1    2
## [2,]    2   -1   -3    1    1
## [3,]    6    2   -1    6   -1
## [4,]   -1    1    4    1    0
## [5,]    1   -1    2    5   -1
```

```
x <- t(A)%*%A
y <- t(A)%*%B
cholesky <- function (A, tol = 1e-07)
{
  nROW <- ncol(A)
  L <- matrix(rep(0, each = nROW * nROW), nrow = nROW, byrow = T)
  for (i in 1:nROW) {
    Aii <- A[i, i] - sum(L[i, 1:i] * L[i, 1:i])
    if (Aii < 0) {
      stop("Matrix no positive definate")
    }
    else {
      L[i, i] <- sqrt(Aii)
    }
    if ((i + 1) <= nROW) {
      for (k in (i + 1):nROW) {
        L[k, i] <- (A[k, i] - sum(L[k, 1:i] * L[i, 1:i]))/L[i, i]
      }
    }
  }
  return(L)
}
c <- cholesky(x)
print(x)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   42    8  -14   42   -5
## [2,]    8    7    3    7   -2
## [3,]  -14    3   34    3    0
## [4,]   42    7    3   64  -12
## [5,]   -5   -2    0  -12    7
```

```
print(y)
```

```
##      [,1]
## [1,]   -1
## [2,]    0
## [3,]    5
## [4,]    4
## [5,]   -2
```

```
k <- print(t(c))
```

```
##          [,1]     [,2]      [,3]      [,4]       [,5]
## [1,] 6.480741 1.234427 -2.160247  6.4807407 -0.7715167
```

```
## [2,] 0.000000 2.340126  2.421522 -0.4273274 -0.4476763
## [3,] 0.000000 0.000000  4.844540  3.7227030 -0.1202609
## [4,] 0.000000 0.000000  0.000000  2.8211476 -2.3903779
## [5,] 0.000000 0.000000  0.000000  0.0000000  0.6899121
```

```
z <- solve(k,y)
print(z)
```

```
##           [,1]
## [1,]  1.613297
## [2,] -2.563426
## [3,]  1.758077
## [4,] -1.038412
## [5,] -2.898920
```

```
l <- solve(c,z)
print(l)
```

```
##              [,1]
## [1,]   0.2489371
## [2,]  -1.2267377
## [3,]   1.0870825
## [4,]  -2.5602392
## [5,] -13.4006160
```

3. QR Decomposition.

```
hilbert <- function(n) { i <- 1:n; 1 / outer(i - 1, i, "+") }
h9 <- hilbert(9); h9
```

```
##            [,1]      [,2]       [,3]       [,4]       [,5]       [,6]
##  [1,] 1.0000000 0.5000000 0.33333333 0.25000000 0.20000000 0.16666667
##  [2,] 0.5000000 0.3333333 0.25000000 0.20000000 0.16666667 0.14285714
##  [3,] 0.3333333 0.2500000 0.20000000 0.16666667 0.14285714 0.12500000
##  [4,] 0.2500000 0.2000000 0.16666667 0.14285714 0.12500000 0.11111111
##  [5,] 0.2000000 0.1666667 0.14285714 0.12500000 0.11111111 0.10000000
##  [6,] 0.1666667 0.1428571 0.12500000 0.11111111 0.10000000 0.09090909
##  [7,] 0.1428571 0.1250000 0.11111111 0.10000000 0.09090909 0.08333333
##  [8,] 0.1250000 0.1111111 0.10000000 0.09090909 0.08333333 0.07692308
##  [9,] 0.1111111 0.1000000 0.09090909 0.08333333 0.07692308 0.07142857
##             [,7]       [,8]       [,9]
##  [1,] 0.14285714 0.12500000 0.11111111
##  [2,] 0.12500000 0.11111111 0.10000000
##  [3,] 0.11111111 0.10000000 0.09090909
##  [4,] 0.10000000 0.09090909 0.08333333
##  [5,] 0.09090909 0.08333333 0.07692308
##  [6,] 0.08333333 0.07692308 0.07142857
##  [7,] 0.07692308 0.07142857 0.06666667
##  [8,] 0.07142857 0.06666667 0.06250000
##  [9,] 0.06666667 0.06250000 0.05882353
```

```
qr(h9)$rank
```

```
## [1] 7
```

```
qrh9 <- qr(h9, tol = 1e-10)
qrh9$rank
```

```
## [1] 9
```

```r
y <- 1:9/10
x <- qr.solve(h9, y, tol = 1e-10)
x <- qr.coef(qrh9, y)
h9 %*% x
```

```
##       [,1]
##  [1,]  0.1
##  [2,]  0.2
##  [3,]  0.3
##  [4,]  0.4
##  [5,]  0.5
##  [6,]  0.6
##  [7,]  0.7
##  [8,]  0.8
##  [9,]  0.9
```

4. Singular Value Decomposition.

```r
A<-matrix(floor(rnorm(25,1,2)),5,5)
B<-c(floor(rnorm(5)))
k <- function (x, nu = min(n, p), nv = min(n, p), LINPACK = FALSE)
{
    x <- as.matrix(x)
    if (any(!is.finite(x)))
        stop("infinite or missing values in 'x'")
    dx <- dim(x)
    n <- dx[1L]
    p <- dx[2L]
    if (!n || !p)
        stop("a dimension is zero")
    La.res <- La.svd(x, nu, nv)
    res <- list(d = La.res$d)
    if (nu)
        res$u <- La.res$u
    if (nv) {
        if (is.complex(x))
            res$v <- Conj(t(La.res$vt))
        else res$v <- t(La.res$vt)
    }
    res
}

asvd <- k(A)
print(asvd)
```

```
## $d
## [1] 7.8852492 5.8366482 4.0241457 3.1781803 0.6795616
##
## $u
##               [,1]       [,2]       [,3]        [,4]       [,5]
## [1,] -0.6833816340  0.1926153 -0.2470875  0.22985366  0.6180647
## [2,] -0.3802054653 -0.4334090 -0.5193366 -0.56356221 -0.2833506
## [3,]  0.0002606711 -0.4574466 -0.3180810  0.78332464 -0.2756258
## [4,] -0.2747490052  0.6968498 -0.1646078  0.11826080 -0.6307392
```

```
## [5,] -0.5594160014 -0.2831942  0.7355039  0.04451752 -0.2527984
##
## $v
##              [,1]       [,2]         [,3]       [,4]       [,5]
## [1,] -0.352378562  0.2851489 -0.033229110  0.7291868  0.5115681
## [2,] -0.424930215 -0.6083242 -0.340646802  0.3428908 -0.4645015
## [3,] -0.688460926 -0.2394715  0.238764707 -0.5054402  0.3952170
## [4,] -0.005109993  0.1494402 -0.908734552 -0.2871337  0.2634337
## [5,] -0.470381869  0.6848013 -0.006964979 -0.1131237 -0.5449244
```

```
adiag <- diag(1/asvd$d)
print(adiag)
```

```
##           [,1]      [,2]      [,3]      [,4]     [,5]
## [1,] 0.1268191 0.0000000 0.0000000 0.0000000 0.000000
## [2,] 0.0000000 0.1713312 0.0000000 0.0000000 0.000000
## [3,] 0.0000000 0.0000000 0.2484999 0.0000000 0.000000
## [4,] 0.0000000 0.0000000 0.0000000 0.3146455 0.000000
## [5,] 0.0000000 0.0000000 0.0000000 0.0000000 1.471537
```

```
adiag[3,3] = 0
solution = asvd$v %*% adiag %*% t(asvd$u) %*% B
print(solution)
```

```
##             [,1]
## [1,]  0.02465235
## [2,] -0.48482124
## [3,]  0.21842507
## [4,]  0.24375481
## [5,] -0.72112089
```

```
check <- A %*% solution
# final Answer
print(check)
```

```
##            [,1]
## [1,] -2.1600181
## [2,] -0.3363312
## [3,] -0.2059947
## [4,] -2.1066028
## [5,] -1.5236752
```

Iterative Methods.

1. Jacobi

```
a <- matrix(c(2,1,5,7), nrow=2, byrow = TRUE)
b <- matrix(c(11, 13), nrow=2, byrow = TRUE)
jacobi <- function (a, b, e = 0.001)
{
T<- array(0,dim=c(5,1))
 n<-5
 l<-0
  for (i in c(1:n))
    T[i][0]<-0
  while (l!=n)
  {
    l<-0
```

```
    for (i in c(1:n))
    {
      x[i][0]<-(1/a[i][i])*(b[i][0]);
      for (j in c(1:n))
      {
        if (j!=i)
          x[i][0] <- x[i][0]-(1/a[i][i])*(a[i][j]*T[j][0]);
      }
    }
    for(i in c(1:n))
    {
      k<-abs(x[i][0]-T[i][0]);
      if (k<=e)
      {
        l<-l+1;
      }
    }
    for (i in c(1:n))
      T[i][0] <- x[i][0];
  }
  for (i in c(1:n))
    print(x[i][0])
}
```

Output
x1=7.11096
x2=-3.22174

2. Gauss-Seidel

```
a<-matrix(floor(rnorm(25,1,2)),5,5)
b<-c(floor(rnorm(5)))
x <-c(0,0,0,0,0)
Seidel <- function(a , b, x)
{
n <- 5
m <- 5
i <- 0
j <- 0
y<- array(0,dim=c(0,5))

  while (m > 0)
  {
    for (i in c(1:n))
    {
      y[i] <- (b[i] / a[i][i])
      for (j in c(1:n))
      {
        if (j != i)
        y[i] <- y[i] - ((a[i][j] / a[i][i]) * x[j]);
        x[i] <- y[i];
      }

    }
    m <- m - 1
```

```
  }
return (y)
}
```

Output
x1=-1.33333 x2=0.33333 x3=-7.33333 x4=-1.4444 x5=nan
x1=nan x2=nan x3=nan x4=nan x5=nan
x1=nan x2=nan x3=nan x4=nan x5=nan

3. Success Overelaxation Method

```
library("optR")
```

```
## Loaded optR Version:            1.2.5
```

```
a<-matrix(floor(rnorm(25,1,2)),5,5)
b<-c(floor(rnorm(5)))
SOR<- function(a,b,w = 1.3, tol = 1e-07)
{
  n <-5
  D <- diag(a)
  luA <- LUsplit(a)
  L <- luA$L
  U <- luA$U
  e <- max(eigen(inv.optR(D+w*L ) * ( D*(1-w) - w*U)))
  if(abs(e)>1)
  {
    print("Since the modulus of largest eigen value of iterative matrix is not less than 1")
    print("The Process is not convergent")
    return
  }
  err <- 10000000 * runif(1,5.0,1)
  x <-c(0,0,0,0,0)
  p <- 0
  v <- matrix(c(0,0,0,0,0), nrow = 5, byrow = TRUE)
  while((sum(abs(err)) >= tol) == v)
  {
    for (i in c(1:n))
    {
      for ( j in c(1:n))
        if( j != i)
          p <- p + a[i][j] * x[j]
    }
    x[i] <- (1 - w) * x[i] + (w/a[i][i]) * (b[i]-p)
  }
  for (i in c(1:n))
    print(x[i][0])
}
```

4. Block Iterative Method

```
a <- matrix(c(2,1,5,7), nrow=2, byrow = TRUE)
b <- matrix(c(11, 13), nrow=2, byrow = TRUE)
jacobi <- function (a, b, e = 0.001)
{
T<- array(0,dim=c(5,1))
```

```
 n<-5
 l<-0
  for (i in c(1:n))
    T[i][0]<-0
  while (l!=n)
  {
    l<-0
    for (i in c(1:n))
    {
      x[i][0]<-(1/a[i][i])*(b[i][0]);
      for (j in c(1:n))
      {
        if (j!=i)
          x[i][0] <- x[i][0]- T[j][0]
      }
    }
    for(i in c(1:n))
    {
      k<-abs(x[i][0]-T[i][0]);
      if (k<=e)
      {
        l<-l+1;
      }
    }
    for (i in c(1:n))
      T[i][0] <- x[i][0];
  }
  for (i in c(1:n))
    print(x[i][0])
}
```