1. Realizamos una funcion la cual me calcula la k-esima suma parcial, y como ejemplo lo realizaremos con la suma parcial numero 1000.

```
> sumaparcial<-function(k){
+   suma = 0
+   for(i in 1:k){
+     #print(suma)
+     suma = suma + 1/(i)^2
+   }
+   return(suma)
+ }
> a<-sumaparcial(1000)
> a

[1] 1.643935
```

2. vamos a hallar una aproximacion a la integral

$$\int_{x=0}^{x=1} e^{-x^2} dx \tag{1}$$

la aproximamos con sumas de riemann,partiendo el intervalo en 1000 segmentos regulares

```
> sumaparcial<-function(k){
+   suma = 0
+   for(i in 1:k){
+     #print(suma)
+     suma = suma + 1/(i)^2
+   }
+   return(suma)
+ }
> a<-sumaparcial(1000)
> a

[1] 1.643935
```

3. Creamos una funcion que me retorne una lista con los numeros primos menores a un n numero dado.El algoritmo lo probamos en dos casos, cuando n=100, y cuando n=0.Lo cual nos retorna:

```
> cribaEratostenes<- function(n){
+   if(n<1){
+     print("no hay primos")
+     return(0)
+   }
+   if(n>1){
+     lista<-2
```

```
+     #print(2)
+   }
+   for (i in 2:n){
+     p=1
+     for(j in 2:((i+1)%/%2)){
+       if( i%%j == 0){
+         p=0
+         break;
+       }
+     }
+     if(p)
+       #print(i)
+       lista <- cbind(lista,i)
+
+   }
+   return(lista)
+ }
> p<-cribaEratostenes(100)
> p

     lista i i i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i  i
[1,]     2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

> ru <-cribaEratostenes(0)

[1] "no hay primos"
```

4. En el caso del discriminante primero creamos la funcion que me retorna las raices, y aplicamos la funcion double para granatizar mayor exactitud en el discriminante,garantizando mayor precision, reduciendo el error en el caso en los cuales la funcion tenga dos raices demasiado "cerca". Por ejemplon probamos el algoritmo en la ecuacion $x^2 + 3x + 2 = 0$

```
> raices<- function(a,b,c){
+   discriminante<-(b^2-4*a*c)
+   #print(discriminante)
+   #print(double(discriminante))
+   if(all(double(discriminante) != 0)){
+     print((-1*b)/2*a)
+   }
+   else{
+     raiz1 = (-1*b+sqrt(discriminante))/2*a
+     raiz2 = (-1*b-sqrt(discriminante))/2*a
+     print(raiz1)
+     print(raiz2)
+   }
+
```

```
+ }
> raices(1,3,2)

[1] -1
[1] -2
```

5. vamos a probar este algoritmopara halla solucion al sistema Ax=b, con

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} b = \begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix}$$

- primero vamos a descomponer la matriz A en una dialgonal(no singular), una triangular superior y otra triangular superior, para evitar que la matriz D sea no singular, en el caso que la matriz A tenga un cero en su diagonal, lecolocamos un 1 en esa posicion, y le colocamos -1 en la misma poscicon a la matriz U, despues sumamos las matrices para garantizar que si nos da la matriz A

```
> descomoposicionD <- function(A,n){
+   D = matrix(0,nrow = n,ncol = n)
+   L = matrix(0,nrow = n,ncol = n)
+   U = matrix(0,nrow = n,ncol = n)
+   solucion = c(D,L,U)
+   for(i in 1:n){
+     if(A[i,i]==0){
+       D[i,i] = -1
+       L[i,i] = 1
+     }else{
+       D[i,i] = A[i,i]
+     }
+   }
+   for(j in 1:n){
+     for(k in 1:n){
+       if(k>j){
+         U[j,k] = A[j,k]
+       }
+       if(k<j){
+         L[j,k] = A[j,k]
+       }
+     }
+   }
+   aux=L
+   L=t(U)
+   U=t(aux)
+   return(D)
```

```
+ }
> descomoposicionL <- function(A,n){
+   D = matrix(0,nrow = n,ncol = n)
+   L = matrix(0,nrow = n,ncol = n)
+   U = matrix(0,nrow = n,ncol = n)
+   solucion = c(D,L,U)
+   for(i in 1:n){
+     if(A[i,i]==0){
+       D[i,i] = -1
+       L[i,i] = 1
+     }else{
+       D[i,i] = A[i,i]
+     }
+   }
+   for(j in 1:n){
+     for(k in 1:n){
+       if(k>j){
+         U[j,k] = A[j,k]
+       }
+       if(k<j){
+         L[j,k] = A[j,k]
+       }
+     }
+   }
+   aux=L
+   L=t(U)
+   U=t(aux)
+   return(L)
+ }
> descomoposicionU <- function(A,n){
+   D = matrix(0,nrow = n,ncol = n)
+   L = matrix(0,nrow = n,ncol = n)
+   U = matrix(0,nrow = n,ncol = n)
+   solucion = c(D,L,U)
+   for(i in 1:n){
+     if(A[i,i]==0){
+       D[i,i] = -1
+       L[i,i] = 1
+     }else{
+       D[i,i] = A[i,i]
+     }
+   }
+   for(j in 1:n){
+     for(k in 1:n){
+       if(k>j){
+         U[j,k] = A[j,k]
```

```
+       }
+       if(k<j){
+         L[j,k] = A[j,k]
+       }
+     }
+   }
+   aux=L
+   L=t(U)
+   U=t(aux)
+   return(U)
+ }
> A = matrix(1:9,nrow = 3,ncol = 3)
> D<-descomoposicionD(A,3)
> U<-descomoposicionU(A,3)
> L<-descomoposicionL(A,3)
> print(D+U+L)

     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

- la matriz de transicion es:

```
> Tj<-function(U,D,L){
+   invD = solve(D)
+   t_j = invD %*% (L+U)
+   return(t_j)
+ }
> t_j <- Tj(U,D,L)
```

- ahora recursivamente solucionamos el sistema, tomando

$$x = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

```
> x=c(1,0,0)
> b=c(1,4,7)
> iteraciones=100
> for(i in 1:iteraciones){
+   invD=solve(D)
+   mat <- invD%*%b
+   re <- t_j%*%x
+   x <- mat-re
+ }
> x
```

```
     [,1]
[1,]    1
[2,]    0
[3,]    0
```