Gráficos

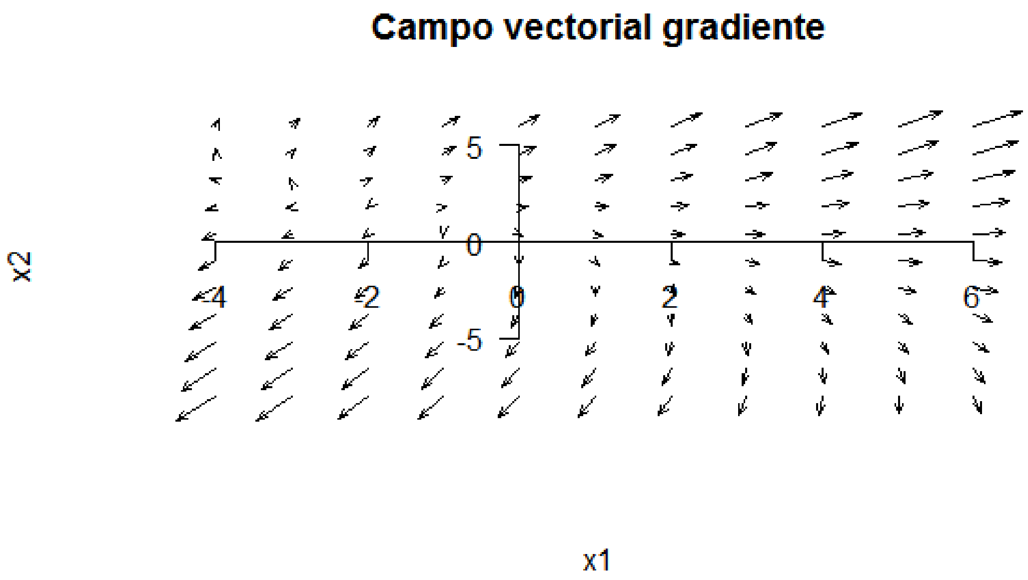
Todos los gráficos se realizaron para la función:

$$f(x, y) = \frac{1}{2}x^T Ax - b^T x + c$$

Con $A = \begin{pmatrix} 3 & 2 \\ 2 & 6 \end{pmatrix}$,$b = \begin{pmatrix} 2 \\ -8 \end{pmatrix}, c = 0, x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ o sea

$$f(x, y) = \frac{3}{2}x_1^2 + 2x_1x_2 + 3x_2^2 + 2x_1 - 8x_2$$

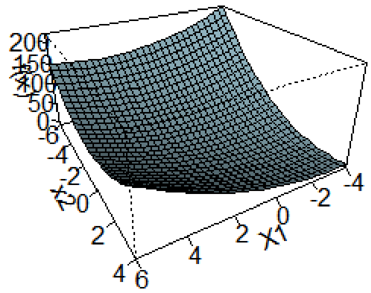
Que tiene un mínimo en $[2, -2]^T$



Código:

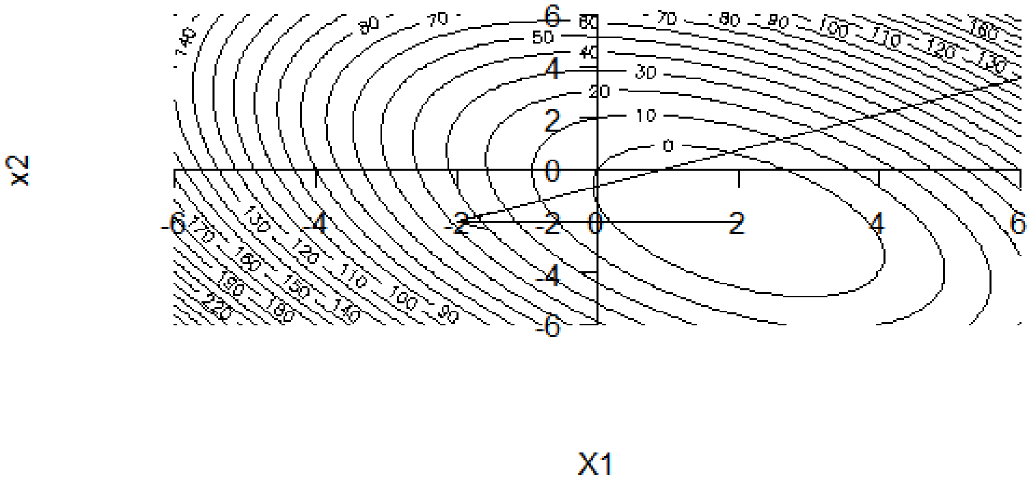
```
# campo vectorial gradiente


expand.outer <- function(x, y, vecfun) {
  xy.pairs <- expand.grid(x=x, y=y, KEEP.OUT.ATTRS = FALSE)
  x.exp <- xy.pairs$x
  y.exp <- xy.pairs$y
  list(values=matrix(vecfun(x.exp,y.exp), nrow=2, byrow=TRUE), x=x.exp,
      y=y.exp)
}
```

```
plotVectorField <- function(vecfun, xlim, ylim, grid.points)   {
  gp <- if(length(grid.points)>1) grid.points else rep(grid.points,2)
  maxlength <- c(diff(xlim),diff(ylim))/(gp-1)*0.9

  x0 <- seq(xlim[1], xlim[2], length=gp[1])
  y0 <- seq(ylim[1], ylim[2], length=gp[2])
  xy.data <- expand.outer(x0, y0, vecfun)
  x0 <- xy.data$x
  y0 <- xy.data$y
  dx <- xy.data$values[1,]
  dy <- xy.data$values[2,]

  k <- min( maxlength / c(max(abs(dx)),max(abs(dy))) )
  x1 <- x0 + k*dx
  y1 <- y0 + k*dy

  plot.default( axes=FALSE,range(x0,x1), range(y0,y1),main="Campo vectorial gradiente", xlab="x1",
               ylab="x2", type="n", frame.plot=F)
  arrows(x0,y0,x1,y1,length = 0.08, angle = 20, code = 2)
  axis(1, pos=0)
  axis(2, pos=0, las=1)
}
plotVectorField(function(x1,x2) c(3*x1+2*x2+2,2*x1+6*x2-8), c(-4,6), c(-8,6), 11)
```



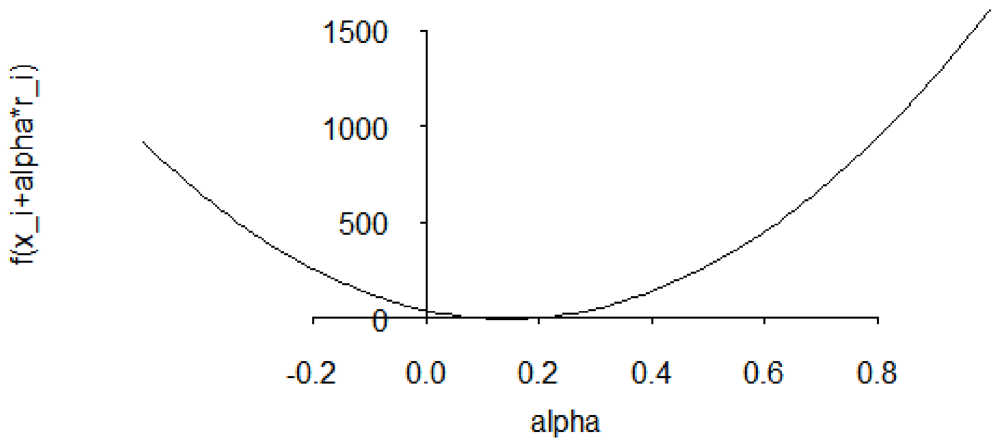
Codigo:

```
nx <- 30
ny <- 30
x1 <- seq(-4, 6, length = nx)
x2 <- seq(-6, 4, length = ny)
z <- outer(x1, x2, function(x1,x2) (3/2)*(x1)^2 +2*x1*x2+3*(x2)^2+2*x1-8*x2)
persp(x1, x2, z, theta = 150, phi = 27,expand = 0.5, col = "lightblue",
      ltheta = 120, shade = 0.75, ticktype = "detailed",
      xlab = "X1", ylab = "x2", zlab = "f(x)") -> res
round(res, 3)
```



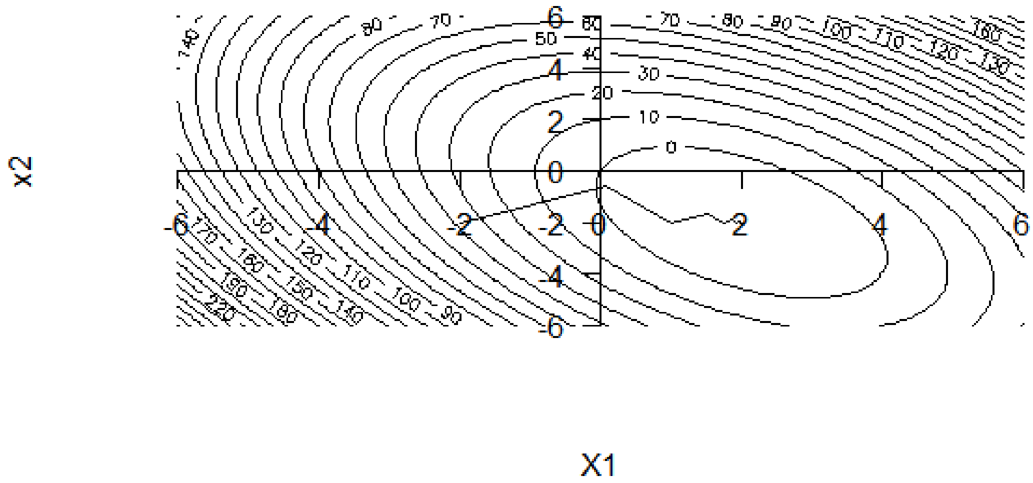
Codigo:

```
x1 = seq(-6, 6, 0.1)
x2= seq(-6, 6, 0.1)
z <- outer(x1, x2, function(x1,x2) (3/2)*(x2)^2 +2*x1*x2+3*(x1)^2+2*x2-8*x1)
contour(x1,x2,z,nlevels=30, axes= FALSE,xlab = "X1", ylab = "x2")
segments(-2, -2, 6, 7/2)
axis(1, pos=0,at = c(-6,-4,-2,0,2,4,6))
axis(2, pos=0,at = c(-6,-4,-2,0,2,4,6),las=1)
arrows(x0=2, y0=-2, x1 = -2, y1 = -2,length = 0.25, angle = 15)
```




```
alpha = seq(-0.5, 1, 0.1)
f <- function(x) (3/2) *(-2+8*x)^2 + 2 * (-2+8*x)*(-2+24*x)  +3*(-2+24*x)^2+2*(-2+8*x)-8*(-
2+24*x)
x1=-2+8*alpha
x2=-2+24*alpha
z <- outer(x1, x2, function(x1,x2) (3/2)*(x1)^2 +2*x1*x2+3*(x2)^2+2*x1-8*x2)
curve(f, from=-0.5, to=1, axes=FALSE, xlab = NA,
     ylab = NA,lty=1)
axis(1, pos=0,at = seq(-0.2, 0.8, by = 0.2),tck = -.015, las=1)
axis(2, pos=0,tck = -.015, las=1)
mtext(side = 1, "alpha", line = 2)
mtext(side = 2, "f(x_i+alpha*r_i)", line = 2)
```

Grafico método steepest descent

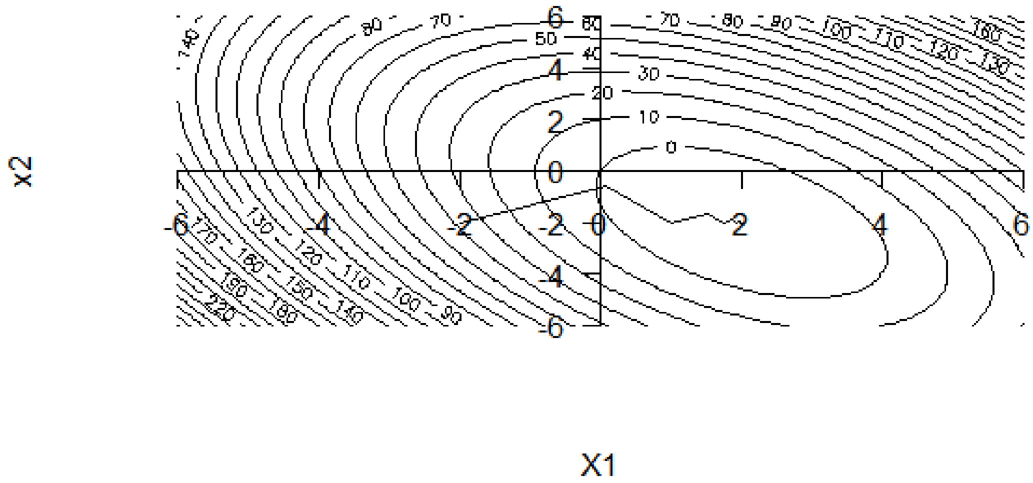


Codigo:

```
x1 = seq(-6, 6, 0.1)

x2= seq(-6, 6, 0.1)


z <- outer(x1, x2, function(x1,x2) (3/2)*(x2)^2 +2*x1*x2+3*(x1)^2+2*x2-8*x1)

contour(x1,x2,z,nlevels=30, axes= FALSE,xlab = "X1", ylab = "x2")

axis(1, pos=0,at = c(-6,-4,-2,0,2,4,6))

axis(2, pos=0,at = c(-6,-4,-2,0,2,4,6),las=1)

l<-function(x) (-4/3)+ (7/3)*x

f<-function(x1,x2) 3*x2+2*x1+2

g<-function(x1,x2) 2*x2+6*x1-8

alpha<-function(A,ri)

{

  n=t(ri)%*%ri

  d=t(ri)%*%A%*%ri

  n/d

}

ri=c(-2,-2)

gradient.descent <- function(A,b,c,x,max.iterations, minError)

  {
```

```
    xi=x

    for (iteration in 1:max.iterations) {

      ri=b-A%*%xi
      sol=solve(A)%*%b
      error=xi-sol
      if (t(error)%*%error < minError)
      {break()}
      al=alpha(A=A,ri=ri)
      oldxi=xi
      xi[1]=xi[1]+al*ri[1]
      xi[2]=xi[2]+al*ri[2]
      segments(oldxi[1], oldxi[2], xi[1], xi[2])
    }

  }
A= matrix(c(3,2,2,6), nrow=2)
b=c(2,-8)
x=c(-2,-2)
gradient.descent(A=A,b=b,x=x,max.iterations=10,minError=0 )
```

Grafico método conjugate gradients



Codigo:

```
x1 = seq(-6, 6, 0.1)
x2= seq(-6, 6, 0.1)

z <- outer(x1, x2, function(x1,x2) (3/2)*(x2)^2 +2*x1*x2+3*(x1)^2+2*x2-8*x1)
contour(x1,x2,z,nlevels=30, axes= FALSE,xlab = "X1", ylab = "x2")
axis(1, pos=0,at = c(-6,-4,-2,0,2,4,6))
axis(2, pos=0,at = c(-6,-4,-2,0,2,4,6),las=1)
l<-function(x) (-4/3)+ (7/3)*x
f<-function(x1,x2) 3*x2+2*x1+2
g<-function(x1,x2) 2*x2+6*x1-8
alpha<-function(A,ri,di)
{
  n=t(ri)%*%ri
  d=t(di)%*%A%*%di
  n/d
}
beta<-function(ri, rinew)
{
  n=t(rinew)%*%rinew
```

```
  #A= matrix(c(3,2,2,6), nrow=2)
  d=t(ri)%*%ri
  n/d
}
ri=c(-2,-2)
gradient.conjugate <- function(A,b,c,x,max.iterations,minError)
{
  xi=x

  for (iteration in 1:max.iterations) {
    sol=solve(A)%*%b
    error=xi-sol
    if (t(error)%*%error < minError)
    {break()}
    ri=b-A%*%xi
    di=ri
    al=alpha(A=A,ri=ri,di=di)
    oldxi=xi
    xi[1]=xi[1]+al*di[1]
    xi[2]=xi[2]+al*di[2]
    temp=A%*%di
    oldri=ri
    ri[1]=ri[1]-al*temp[1]
    ri[2]=ri[2]-al*temp[2]
    bet=beta(ri=oldri, rinew=ri)
    di[1]=ri[1]+bet*di[1]
    di[2]=ri[2]+bet*di[2]
    segments(oldxi[1], oldxi[2], xi[1], xi[2])
  }



}
```

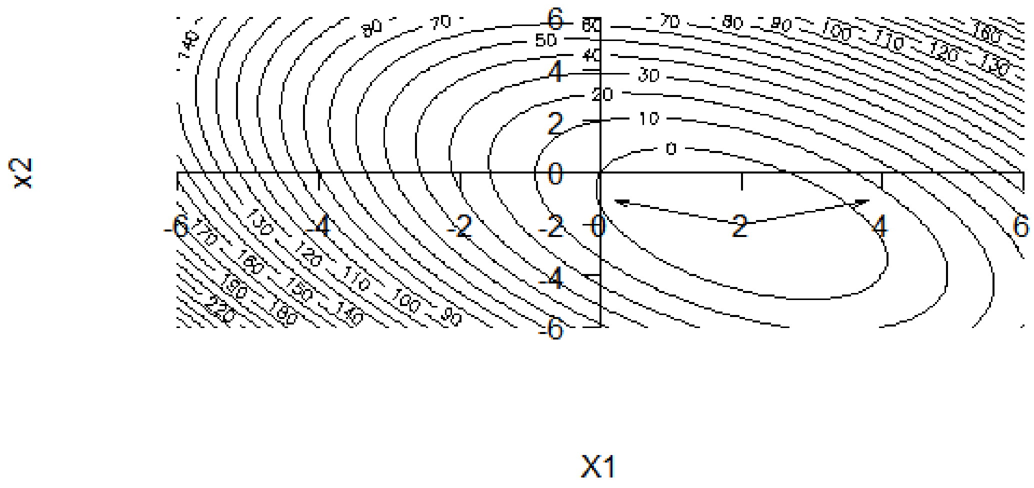
A= matrix(c(3,2,2,6), nrow=2)

b=c(2,-8)

x=c(-2,-2)

gradient.conjugate(A=A,b=b,x=x,max.iterations=10,minError=0)

Gráfico de auto vectores



Codigo:

A= matrix(c(3,2,2,6), nrow=2)

vectors=eigen(A)$vectors

x1 = seq(-6, 6, 0.1)

x2= seq(-6, 6, 0.1)

z <- outer(x1, x2, function(x1,x2) (3/2)*(x2)^2 +2*x1*x2+3*(x1)^2+2*x2-8*x1)

contour(x1,x2,z,nlevels=30, axes= FALSE,xlab = "X1", ylab = "x2")

axis(1, pos=0,at = c(-6,-4,-2,0,2,4,6))

axis(2, pos=0,at = c(-6,-4,-2,0,2,4,6),las=1)

v1=eigen(A)$vectors[,1]

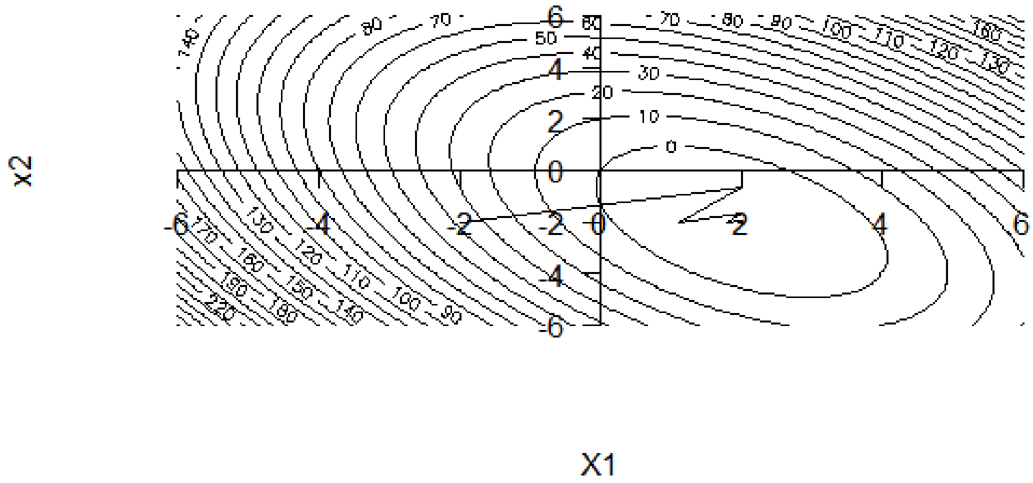
v2=eigen(A)$vectors[,2]

k=v1[1]/v1[2]

arrows(x0=2, y0=-2, x1 = 2+1.8, y1 = -2+(-k)*(-1.8),length = 0.10, angle = 15)

arrows(x0=2, y0=-2, x1 = 2-(1.8), y1 = -2+(k)*(1.8),length = 0.10, angle = 15)

Grafico método de Jacobi

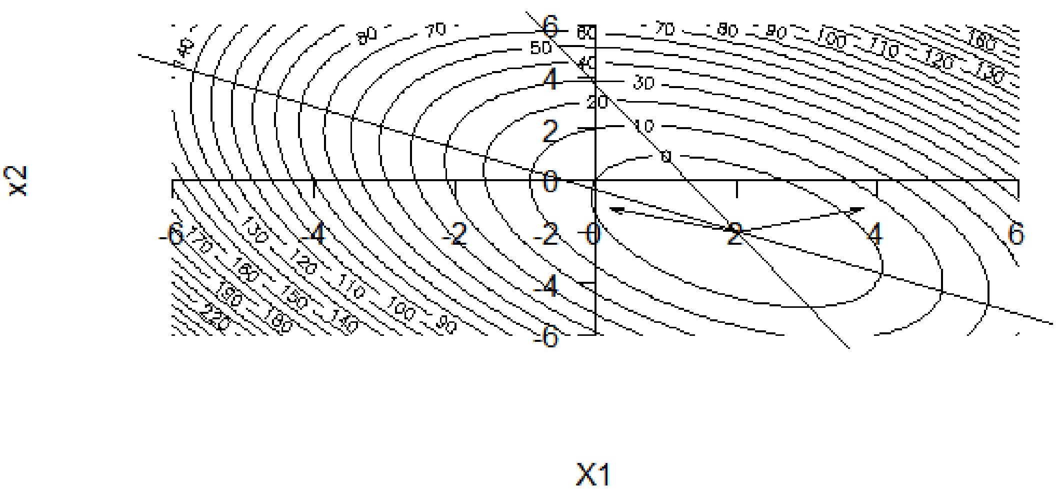


Codigo

```
A= matrix(c(3,2,2,6), nrow=2)
vectors=eigen(A)$vectors
x1 = seq(-6, 6, 0.1)
x2= seq(-6, 6, 0.1)

z <- outer(x1, x2, function(x1,x2) (3/2)*(x2)^2 +2*x1*x2+3*(x1)^2+2*x2-8*x1)
contour(x1,x2,z,nlevels=30, axes= FALSE,xlab = "X1", ylab = "x2")
axis(1, pos=0,at = c(-6,-4,-2,0,2,4,6))
axis(2, pos=0,at = c(-6,-4,-2,0,2,4,6),las=1)
jacobi <- function(A,b,c,x,max.iterations,minError)
{
  xi=x

  for (iteration in 1:max.iterations) {
    sol=solve(A)%*%b
    error=xi-sol
    if (t(error)%*%error < minError)
    {break()}
```

```
   D=diag(as.array(diag(A)))
   E=A-D
   Dinv=solve(D)
   B=(-1)*Dinv%*%E
   z=Dinv%*%b
   oldxi=xi
   t=B%*%xi
   xi[1]=t[1]+z[1]
   xi[2]=t[2]+z[2]

   segments(oldxi[1], oldxi[2], xi[1], xi[2])
  }


}
A= matrix(c(3,2,2,6), nrow=2)
b=c(2,-8)
x=c(-2,-2)
jacobi(A=A,b=b,x=x,max.iterations=10,minError=0)
```

Direcciones de más lenta convergencia del método steepest descent



Codigo:

```
A= matrix(c(3,2,2,6), nrow=2)

vectors=eigen(A)$vectors

x1 = seq(-6, 6, 0.1)

x2= seq(-6, 6, 0.1)


z <- outer(x1, x2, function(x1,x2) (3/2)*(x2)^2 +2*x1*x2+3*(x1)^2+2*x2-8*x1)


contour(x1,x2,z,nlevels=30, axes= FALSE,xlab = "X1", ylab = "x2")

axis(1, pos=0,at = c(-6,-4,-2,0,2,4,6))

axis(2, pos=0,at = c(-6,-4,-2,0,2,4,6),las=1)

v1=eigen(A)$vectors[,1]

v2=eigen(A)$vectors[,2]

k=v1[1]/v1[2]

arrows(x0=2, y0=-2, x1 = 2+1.8, y1 = -2+(-k)*(-1.8),length = 0.10, angle = 15)

arrows(x0=2, y0=-2, x1 = 2-(1.8), y1 = -2+(k)*(1.8),length = 0.10, angle = 15)

#abline(a=0, b=-1)



p=max(eigen(A)$values)/min(eigen(A)$values)


cos=sqrt(1/(1+k^2))

sen=sqrt(k/(1+k^2))

abline(b=(p*cos+sen)/(cos-p*sen),a=(-2)*((p*cos+sen)/(cos-p*sen))-2)


p=(-1)*max(eigen(A)$values)/min(eigen(A)$values)

abline(b=(p*cos+sen)/(cos-p*sen),a=(-2)*((p*cos+sen)/(cos-p*sen))-2)
```