Rで行列・固有値分解：実固有値編

行列計算の復習

行列を作る

```
x <- 1
y <- 2
z <- 0.5
w <- 1.5
M <- matrix(c(x,y,z,w),2,2)
M
```

```
##      [,1] [,2]
## [1,]    1  0.5
## [2,]    2  1.5
```

```
eigen(M)
```

```
## eigen() decomposition
## $values
## [1] 2.2807764 0.2192236
##
## $vectors
##            [,1]       [,2]
## [1,] -0.3636591 -0.5392856
## [2,] -0.9315321  0.8421229
```

これがどういう行列だったかと言うと

- (1,0) -> (x,y)
- (0,1) -> (z,w)

確かめる

```
v1 <- c(1,0)
M %*% v1
```

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

```
v2 <- c(0,1)
M %*% v2
```

```
##      [,1]
## [1,]  0.5
## [2,]  1.5
```

この検算は一括して行える

```
v12 <- cbind(v1,v2)
v12
```

```
##      v1 v2
## [1,]  1  0
## [2,]  0  1
```

```
M %*% v12
```

```
##      v1  v2
## [1,]  1 0.5
## [2,]  2 1.5
```

v12 は単位行列

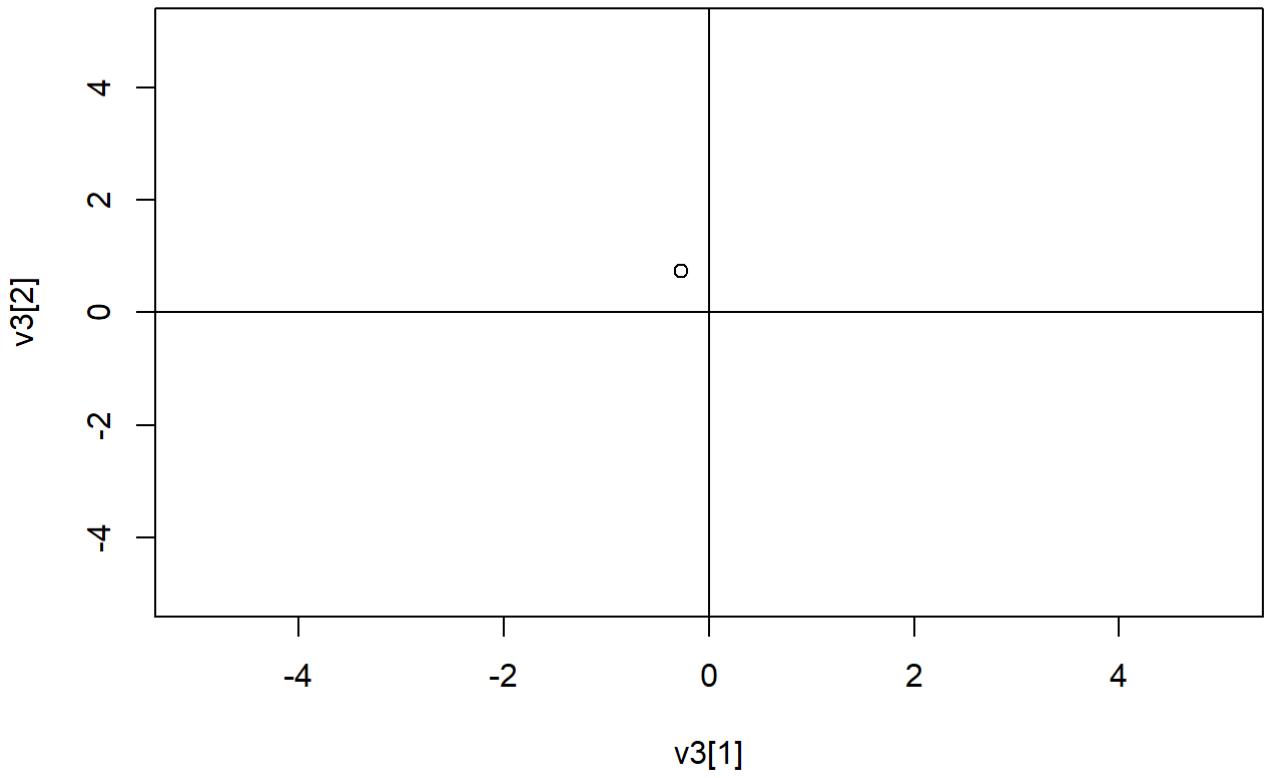
単位行列は簡単に作れる

```
I <- diag(c(1,1)) # 対角成分を引数で指定する
I
```

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

Mが起こす変化を図にしてみる

```
v3 <- rnorm(2) # 2つの乱数(正規分布に従う正規乱数)を発生させて、適当な点をとる
plot(v3[1],v3[2],xlim=c(-5,5),ylim=c(-5,5)) # xlim,ylimはx軸,y軸の範囲を指定する
abline(h=0) # y=0 の水平(horizontal)線を引く
abline(v=0) # x=0 の鉛直(vertical)線を引く
```



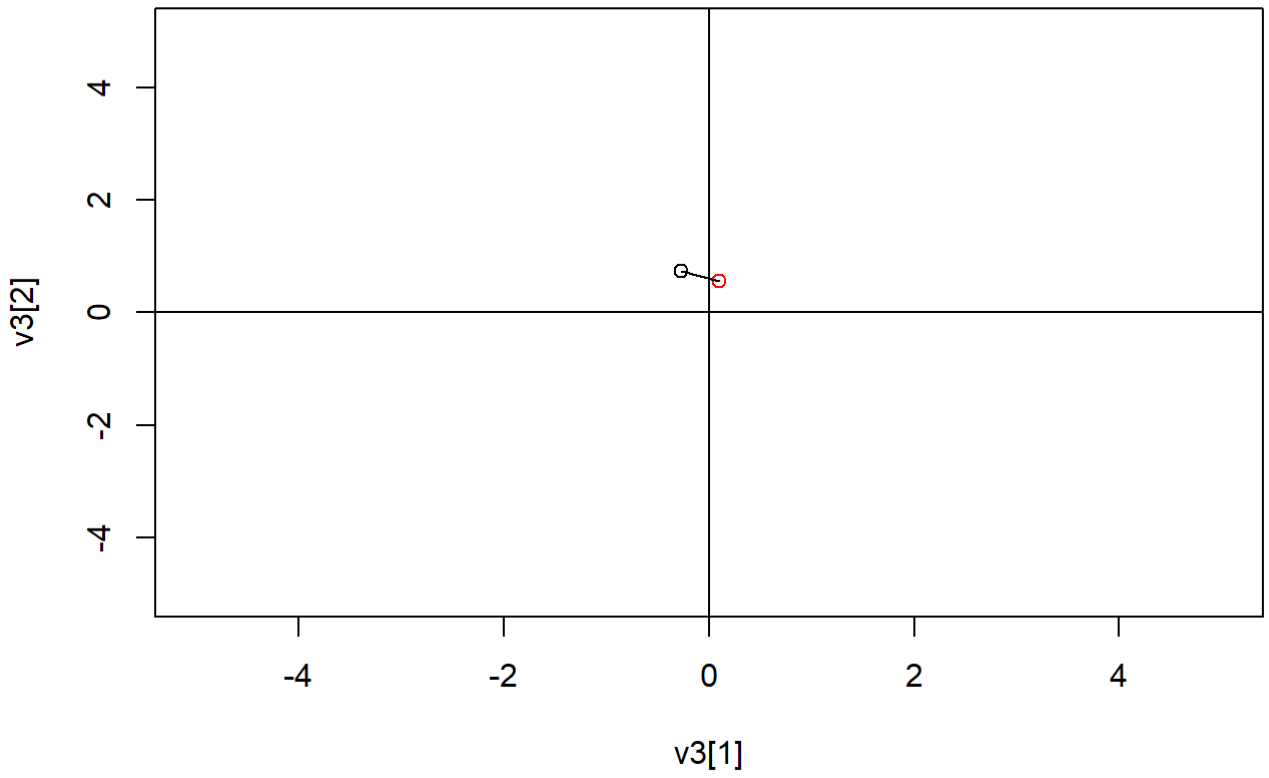
v3にMを作用させる(移動させる)

```
v3. <- M %*% v3
v3.
```

```
##             [,1]
## [1,] 0.09472439
## [2,] 0.55773787
```

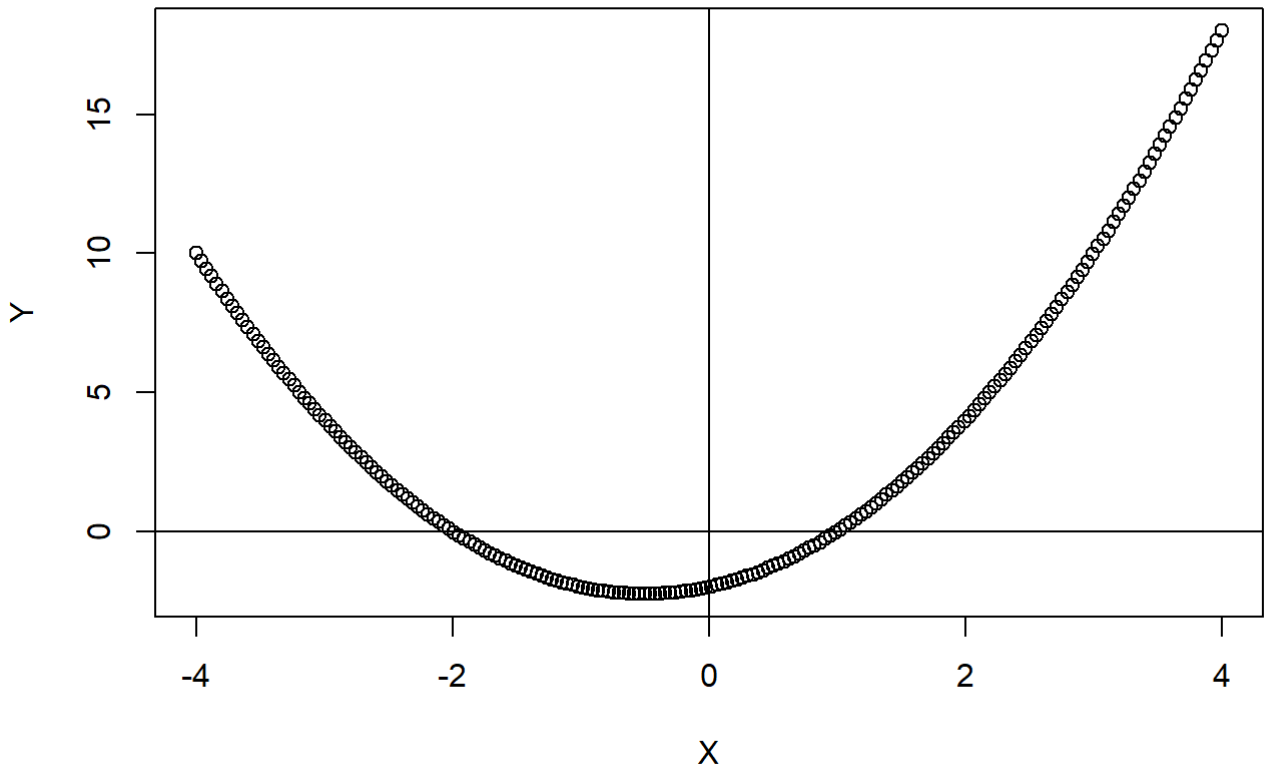
合わせてプロットする

```
plot(v3[1],v3[2],xlim=c(-5,5),ylim=c(-5,5)) # xlim,ylimはx軸,y軸の範囲を指定する
abline(h=0) # y=0 の水平(horizontal)線を引く
abline(v=0) # x=0 の鉛直(vertical)線を引く
points(v3.[1],v3.[2],col=2)
segments(v3[1],v3[2],v3.[1],v3.[2])
```



多数の点をまとめて移動させてみる

```
p.pt <- 200
X <- seq(from=-4,to=4,length=p.pt)
Y <- (X + 2) * (X - 1)
plot(X,Y,type="b")
abline(h=0)
abline(v=0)
```



まとめてMを作用させてみる

各列が座標ならよいから、Xを１行目(１段目)、Yを２行目(２段目)にしたXYを作る

```
XY <- rbind(X,Y)
XY[,1:5] # 初めの５点だけ見ておく
```

```
##   [,1]      [,2]      [,3]      [,4]      [,5]
## X   -4 -3.959799 -3.919598 -3.879397 -3.839196
## Y   10  9.720209  9.443650  9.170324  8.900230
```

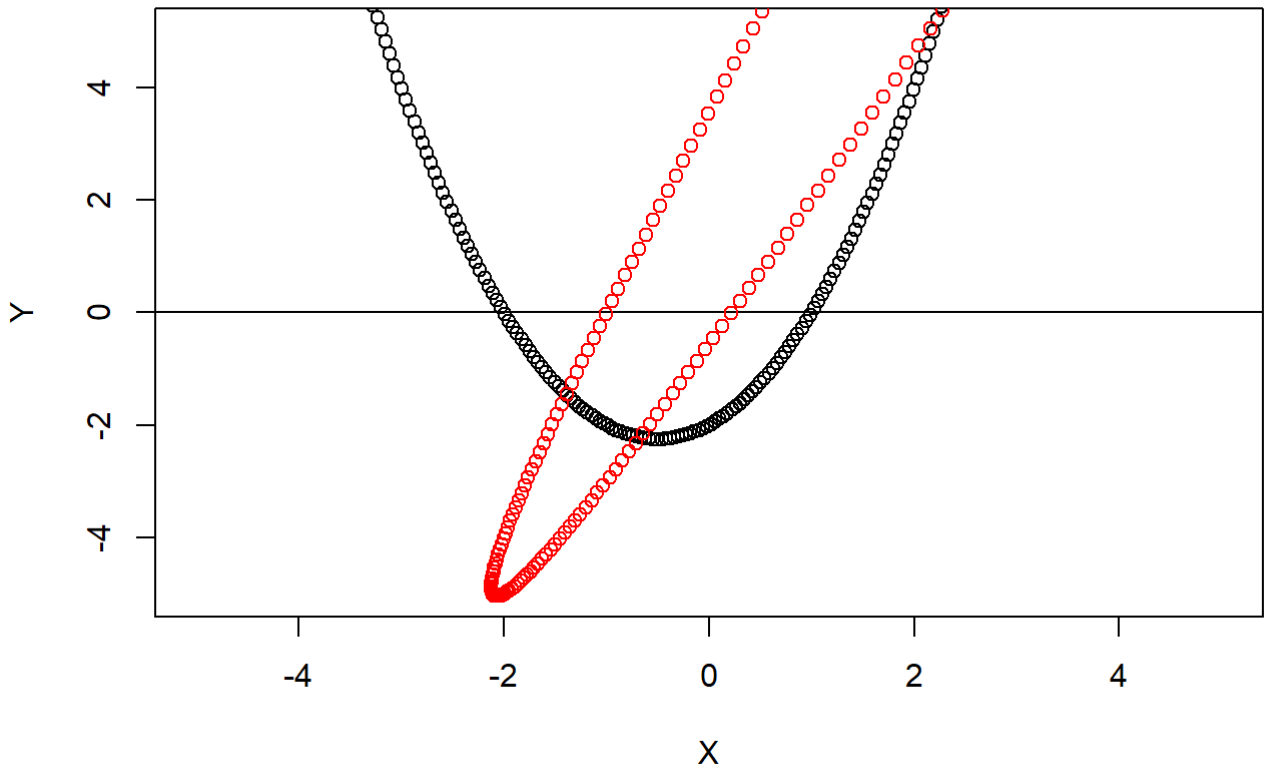
```
dim(XY)
```

```
## [1]   2 200
```

```
XY. <- M %*% XY
plot(X,Y,type="b",xlim=c(-5,5),ylim=c(-5,5))
abline(h=0)
abline(x=0)
```

```
## Warning in int_abline(a = a, b = b, h = h, v = v, untf = untf, ...): "x" は
## グラフィックスパラメータではありません
```

```
points(XY.[1,],XY.[2,],col=2,type="b")
```

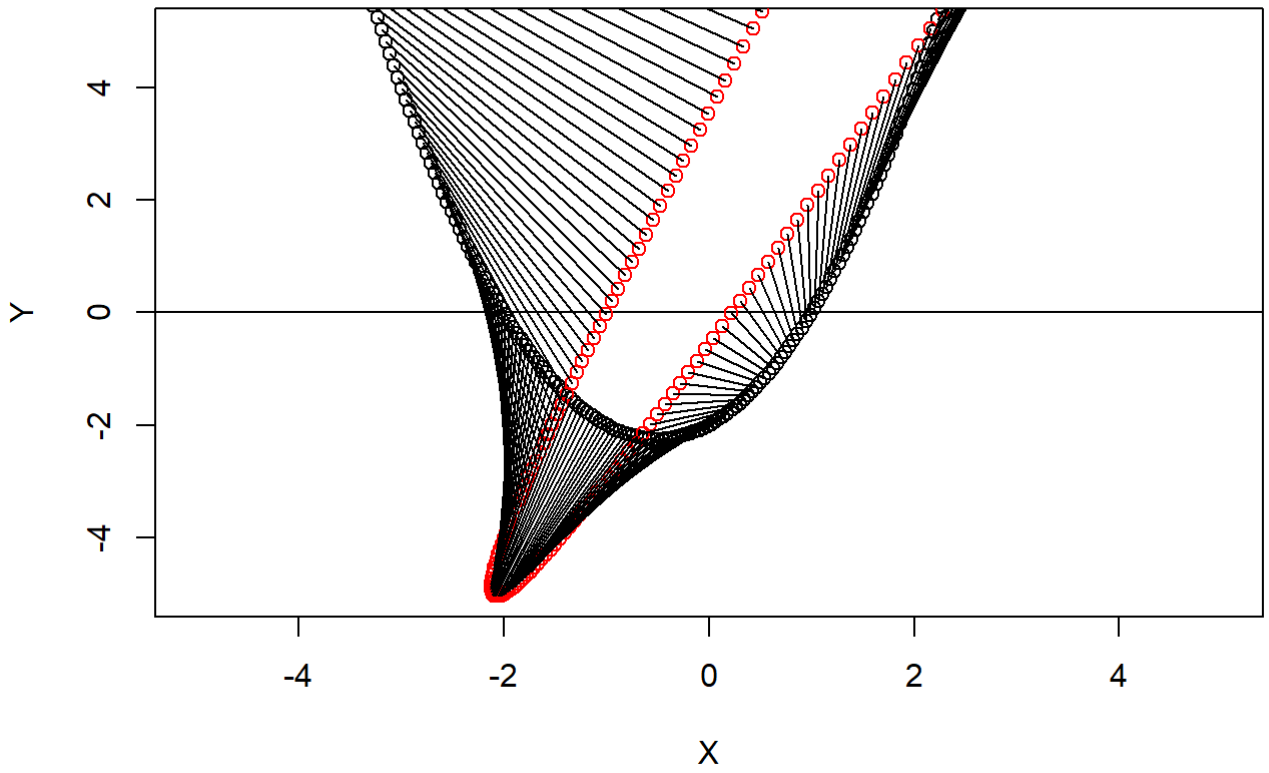


どの点がどの点に動いたのか示しておく

```
XY. <- M %*% XY
plot(X,Y,type="b",xlim=c(-5,5),ylim=c(-5,5))
abline(h=0)
abline(x=0)
```

```
## Warning in int_abline(a = a, b = b, h = h, v = v, untf = untf, ...): "x" は
## グラフィックスパラメータではありません
```

```
points(XY.[1,],XY.[2,],col=2,type="b")
segments(X,Y,XY.[1,],XY.[2,])
```



たくさん動いた点と、ほとんど動かない点とがある

固有値・固有ベクトルの復習

行列Mの固有値と固有ベクトルは、以下の式を満たすλとxのこと

$$Mx = \lambda x$$

Rでは以下のようにして、固有値と固有ベクトルを計算する

```
eigen.out <- eigen(M)
eigen.out
```

```
## eigen() decomposition
## $values
## [1] 2.2807764 0.2192236
##
## $vectors
##            [,1]       [,2]
## [1,] -0.3636591 -0.5392856
## [2,] -0.9315321  0.8421229
```

固有値は

```
eigen.out$values # またはeigen.out[[1]]
```

```
## [1] 2.2807764 0.2192236
```

固有ベクトルは

```
eigen.out$vectors # またはeigen.out[[2]]
```

```
##            [,1]       [,2]
## [1,] -0.3636591 -0.5392856
## [2,] -0.9315321  0.8421229
```

検算

$$Mx = \lambda x$$

検算してみる

```
M %*% eigen.out$vectors[,1]
```

```
##            [,1]
## [1,] -0.8294252
## [2,] -2.1246164
```

```
eigen.out$values[1] * eigen.out$vectors[,1]
```

```
## [1] -0.8294252 -2.1246164
```

```
M %*% eigen.out$vectors[,2]
```

```
##            [,1]
## [1,] -0.1182241
## [2,]  0.1846132
```

```
eigen.out$values[2] * eigen.out$vectors[,2]
```

```
## [1] -0.1182241  0.1846132
```

２つの固有値・２つの固有ベクトルを一括して検算することもできる

```
M %*% eigen.out$vectors
```

```
##            [,1]       [,2]
## [1,] -0.8294252 -0.1182241
## [2,] -2.1246164  0.1846132
```

```
eigen.out$values[1] * eigen.out$vectors[,1]
```

```
## [1] -0.8294252 -2.1246164
```

```
eigen.out$values[2] * eigen.out$vectors[,2]
```

```
## [1] -0.1182241  0.1846132
```

固有値を対角成分に持つ対角行列を使えばもっと簡単に検算できる

```
S <- diag(eigen.out$values)
S # 固有値の対角行列
```

```
##          [,1]      [,2]
## [1,] 2.280776 0.0000000
## [2,] 0.000000 0.2192236
```

```
S %*% eigen.out$vectors
```

```
##            [,1]       [,2]
## [1,] -0.8294252 -1.2299899
## [2,] -0.2042138  0.1846132
```

```
M %*% eigen.out$vectors
```

```
##            [,1]       [,2]
## [1,] -0.8294252 -0.1182241
## [2,] -2.1246164  0.1846132
```

固有ベクトルを図示する

固有ベクトルが意味するのは、「固有」な方向(直線)

その方向(直線)上の点は、固有値倍される

確かめる

```
k <- rnorm(1) # 適当な数
k
```

```
## [1] -0.9530269
```

```
M %*% (k * eigen.out$vectors[,1])
```

```
##           [,1]
## [1,] 0.7904645
## [2,] 2.0248166
```

```
k * eigen.out$values[1] * eigen.out$vectors[,1]
```

```
## [1] 0.7904645 2.0248166
```

固有ベクトルが意味する方向(直線)を図示する

その直線は、原点を通り、傾きが

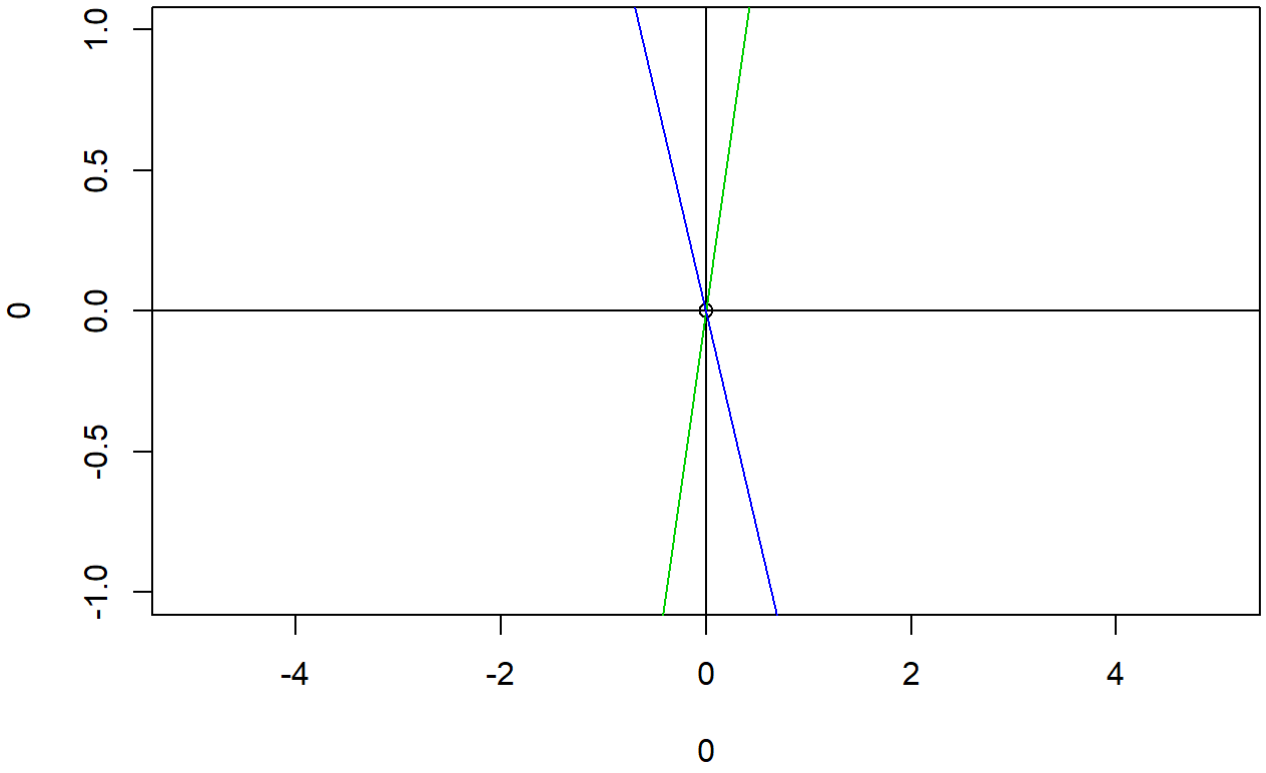
```
sl1 <- eigen.out$vectors[2,1]/eigen.out$vectors[1,1] # ベクトル(x,y)の傾きはy/x
sl2 <- eigen.out$vectors[2,2]/eigen.out$vectors[1,2]
# sl12 <- eigen.out$vectors[2,]/eigen.out$vectors[1,]として、２つの傾きを一回で計算することもできる
sl1
```

```
## [1] 2.561553
```

```
sl2
```

```
## [1] -1.561553
```

```
plot(0,0,xlim=c(-5,5),ylim=c(0,0))
abline(h=0)
abline(v=0)
abline(0,sl1,col=3) # y切片 = 0,傾きsl1の直線を色番号3で引く
abline(0,sl2,col=4)
```

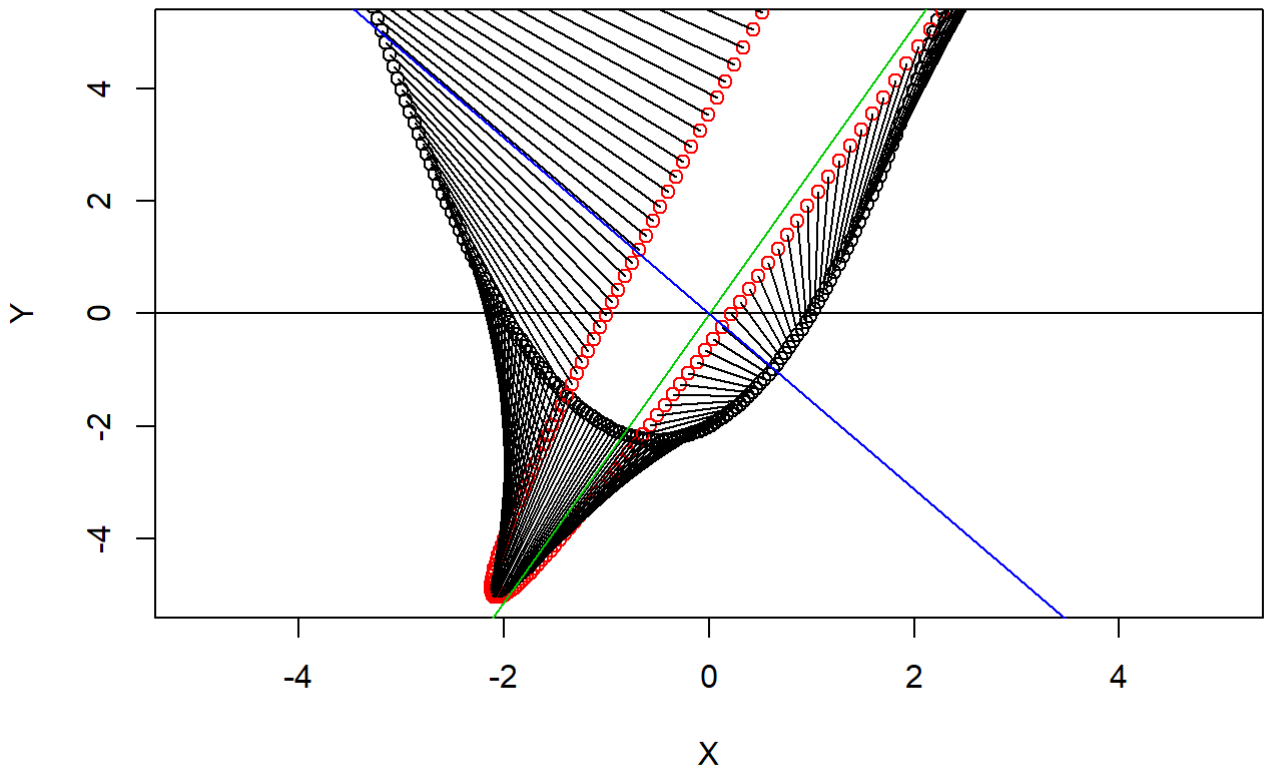


点の移動を表すプロットと、固有ベクトル方向直線を併せてプロットする

```
plot(X,Y,type="b",xlim=c(-5,5),ylim=c(-5,5))
abline(h=0)
abline(x=0)
```

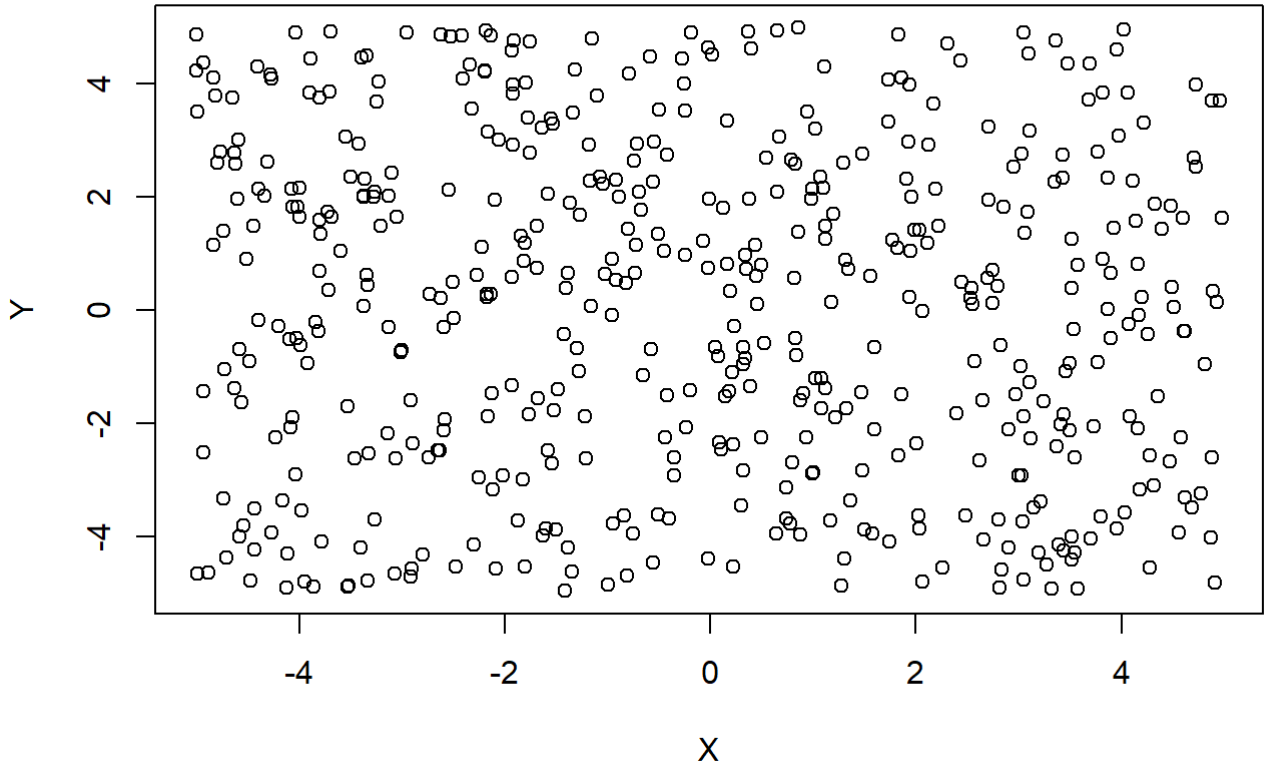
```
## Warning in int_abline(a = a, b = b, h = h, v = v, untf = untf, ...): "x" は
## グラフィックスパラメータではありません
```

```
points(XY.[1,],XY.[2,],col=2,type="b")
segments(X,Y,XY.[1,],XY.[2,])
abline(0,sl1,col=3)
abline(0,sl2,col=4)
```



点の移動の様子を、関数上の点に限らず図示する

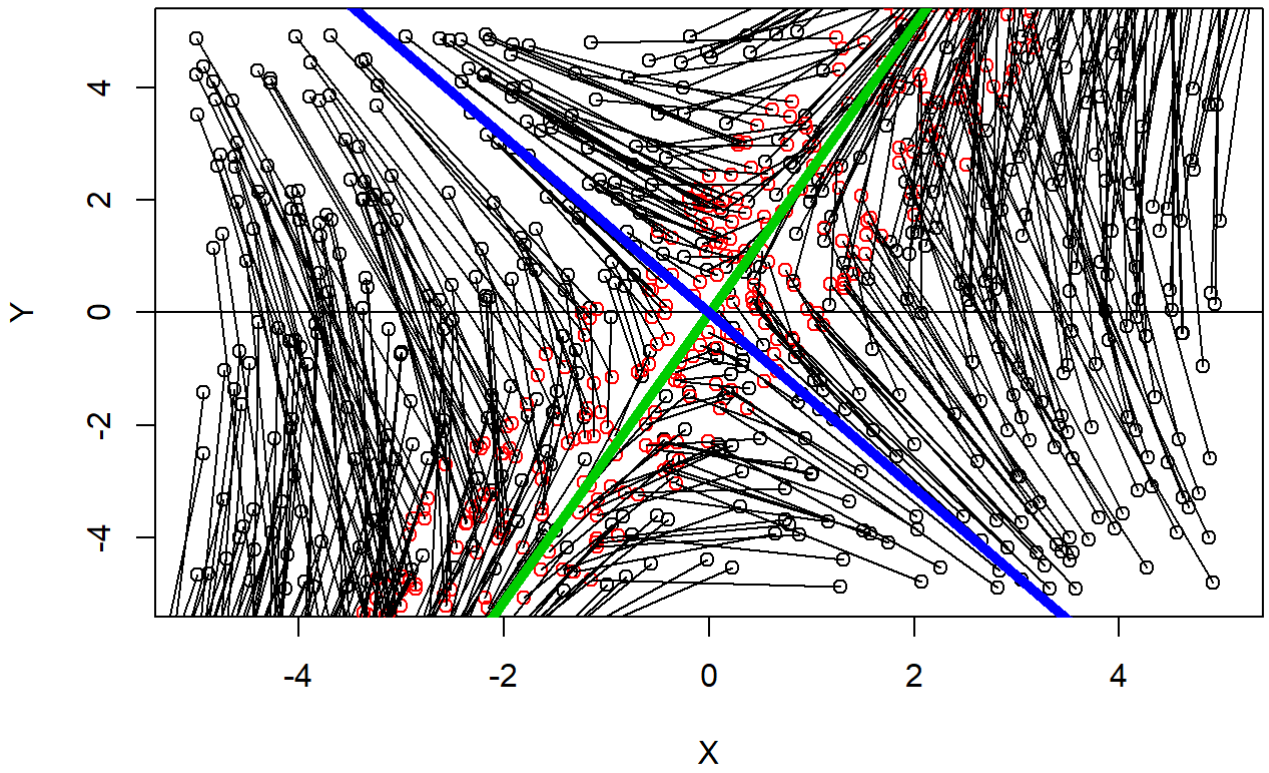
```
n.pt <- 500
X <- runif(n.pt,min=-5,max=5)
Y <- runif(n.pt,min=-5,max=5)
plot(X,Y)
```




```
XY <- rbind(X,Y)
XY. <- M %*% XY
plot(X,Y,xlim=c(-5,5),ylim=c(-5,5))
points(XY.[1,],XY.[2,],col=2)
segments(X,Y,XY.[1,],XY.[2,])
abline(h=0)
abline(x=0)
```

```
## Warning in int_abline(a = a, b = b, h = h, v = v, untf = untf, ...): "x" は
## グラフィックスパラメータではありません
```

```
abline(0,sl1,col=3,lw=5)
abline(0,sl2,col=4,lw=5)
```



固有値分解の「分解」という意味

```
V <- eigen.out$vectors
S <- diag(eigen.out$values)
S
```

```
##          [,1]      [,2]
## [1,] 2.280776 0.0000000
## [2,] 0.000000 0.2192236
```

```
V.inv <- solve(V)
V %*% V.inv
```

```
##      [,1]         [,2]
## [1,]    1 5.551115e-17
## [2,]    0 1.000000e+00
```

```
V.inv %*% V
```

```
##      [,1]          [,2]
## [1,]    1 -1.110223e-16
## [2,]    0  1.000000e+00
```

分解されている

```
V %*% S %*% V.inv
```

```
##      [,1] [,2]
## [1,]    1  0.5
## [2,]    2  1.5
```

```
M
```

```
##      [,1] [,2]
## [1,]    1  0.5
## [2,]    2  1.5
```

固有値はMの重要な情報

重要な情報は対角成分に集約される

```
sum(S) # 固有値分解で取り出した対角行列
```

```
## [1] 2.5
```

```
sum(eigen.out$values) # sum(S)と同じこと
```

```
## [1] 2.5
```

```
M
```

```
##      [,1] [,2]
## [1,]    1  0.5
## [2,]    2  1.5
```

```
diag(M)
```

```
## [1] 1.0 1.5
```

```
sum(diag(M)) # Mのトレース
```

```
## [1] 2.5
```

行列式も大事な情報

それは固有値の積

```
det(M)
```

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

```
prod(eigen.out$values)
```

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

これはたまたま起きることではない

```
d <- 100 # d x d 行列
M2 <- matrix(rnorm(d^2),d,d)
eigen.out2 <- eigen(M2)
sum(eigen.out2$values)
```

```
## [1] -18.25553+0i
```

```
sum(diag(M2))
```

```
## [1] -18.25553
```

```
prod(eigen.out2$values)
```

```
## [1] -1.320733e+78+0i
```

```
det(M2)
```

```
## [1] -1.320733e+78
```

固有値が複素数の場合

$$M = VSV^{-1}$$

ただし、Sは対角行列

というとき、Sの対角成分は複素数でもよい

Mの成分が実数であるためには、共役複素数である必要はある

```
lambda1 <- 0.5 + 1i * 0.7
lambda1
```

```
## [1] 0.5+0.7i
```

```
lambda2 <- Conj(lambda1) # 共役複素数
lambda2
```

```
## [1] 0.5-0.7i
```

```
S <- diag(c(lambda1,lambda2))
S
```

```
##          [,1]     [,2]
## [1,] 0.5+0.7i 0.0+0.0i
## [2,] 0.0+0.0i 0.5-0.7i
```

固有ベクトルも成分同士が共役にすると、Mcは実数行列になる

```
eigen.v1 <- c(rnorm(1)+1i*rnorm(1),rnorm(1)+1i*rnorm(1))
eigen.v2 <- Conj(eigen.v1)
eigen.v1
```

```
## [1] -1.0732814-0.3207619i -0.0271137-0.7925607i
```

```
eigen.v2
```

```
## [1] -1.0732814+0.3207619i -0.0271137+0.7925607i
```

```
V <- cbind(eigen.v1,eigen.v2)
V.inv <- solve(V)
V
```

```
##                   eigen.v1              eigen.v2
## [1,] -1.0732814-0.3207619i -1.0732814+0.3207619i
## [2,] -0.0271137-0.7925607i -0.0271137+0.7925607i
```

```
V.inv
```

```
##                         [,1]                 [,2]
## [1,] -0.4706733-0.0161018i 0.190489+0.6373831i
## [2,] -0.4706733+0.0161018i 0.190489-0.6373831i
```

```
V %*% V.inv
```

```
##                   [,1]                 [,2]
## [1,]  1.000000e+00+0i 0e+00-1.110223e-16i
## [2,] -2.775558e-17+0i 1e+00-0.000000e+00i
```

```
M <- V %*% S %*% V.inv
M
```

```
##               [,1]         [,2]
## [1,]  0.2644418+0i 1.0432703-0i
## [2,] -0.5228632+0i 0.7355582+0i
```

固有値分解しなおす

```
eigen.out <- eigen(M)
eigen.out
```

```
## eigen() decomposition
## $values
## [1] 0.5+0.7i 0.5-0.7i
##
## $vectors
##                       [,1]                  [,2]
## [1,] 0.8161764+0.0000000i 0.8161764+0.0000000i
## [2,] 0.1842831+0.5476275i 0.1842831-0.5476275i
```

固有値は指定したものそのもの

固有ベクトルは様子が違う

```
S
```

```
##          [,1]     [,2]
## [1,] 0.5+0.7i 0.0+0.0i
## [2,] 0.0+0.0i 0.5-0.7i
```

```
eigen.out$values
```

```
## [1] 0.5+0.7i 0.5-0.7i
```

```
V
```

```
##                    eigen.v1              eigen.v2
## [1,] -1.0732814-0.3207619i -1.0732814+0.3207619i
## [2,] -0.0271137-0.7925607i -0.0271137+0.7925607i
```

```
eigen.out$vectors
```

```
##                       [,1]                  [,2]
## [1,] 0.8161764+0.0000000i 0.8161764+0.0000000i
## [2,] 0.1842831+0.5476275i 0.1842831-0.5476275i
```

ただし、固有ベクトルは「定数倍」しても意味に変わりはないから、確認してみると、確かに、定数倍(ただし、複素数)になっている

```
eigen.out$vectors / V
```

```
##                  eigen.v1            eigen.v2
## [1,] -0.698097+0.208634i -0.698097-0.208634i
## [2,] -0.698097+0.208634i -0.698097-0.208634i
```