Básico de Álgebra Linear

Guilherme Garcia

09 de Maio de 2017

Objetivos

O objetivo deste tutorial é introduzir alguns conceitos básicos de álgebra linear. Esta disciplina possui um número muito grande de aplicações, que vão da computação gráfica até a previsão do tempo. No contexto deste curso, alguns conceitos centrais em genética quantitativa são expressos através de vetores e matrizes, os objetos centrais desta introdução.

Vetores

Existem três concepções básicas a respeito do que são vetores. Talvez, a concepção mais próxima de nós biólogos é a de um vetor como uma coleção de valores numéricos. Imagine que você toma algumas medidas lineares de uma coleção de organismos: a largura da cabeça, o comprimento da cabeça e o comprimento da cauda, por exemplo. Para cada indivíduo de sua amostra, você tem uma coleção de três valores, e é razoável pensar que cada indivíduo é representado por uma concatenação destes três valores,

$$\boldsymbol{a} = (x, y, z); \quad \boldsymbol{b} = (x', y', z')...$$

em que $\boldsymbol{a}$ e $\boldsymbol{b}$ se referem a dois indivíduos distintos na amostra. Claro que não precisamos nos ater a apenas três mensurações, de modo que esse vetor pode representar quantas medidas gostaríamos de tomar, mas é melhor manter as ideias em dimensionalidade baixa, por enquanto.

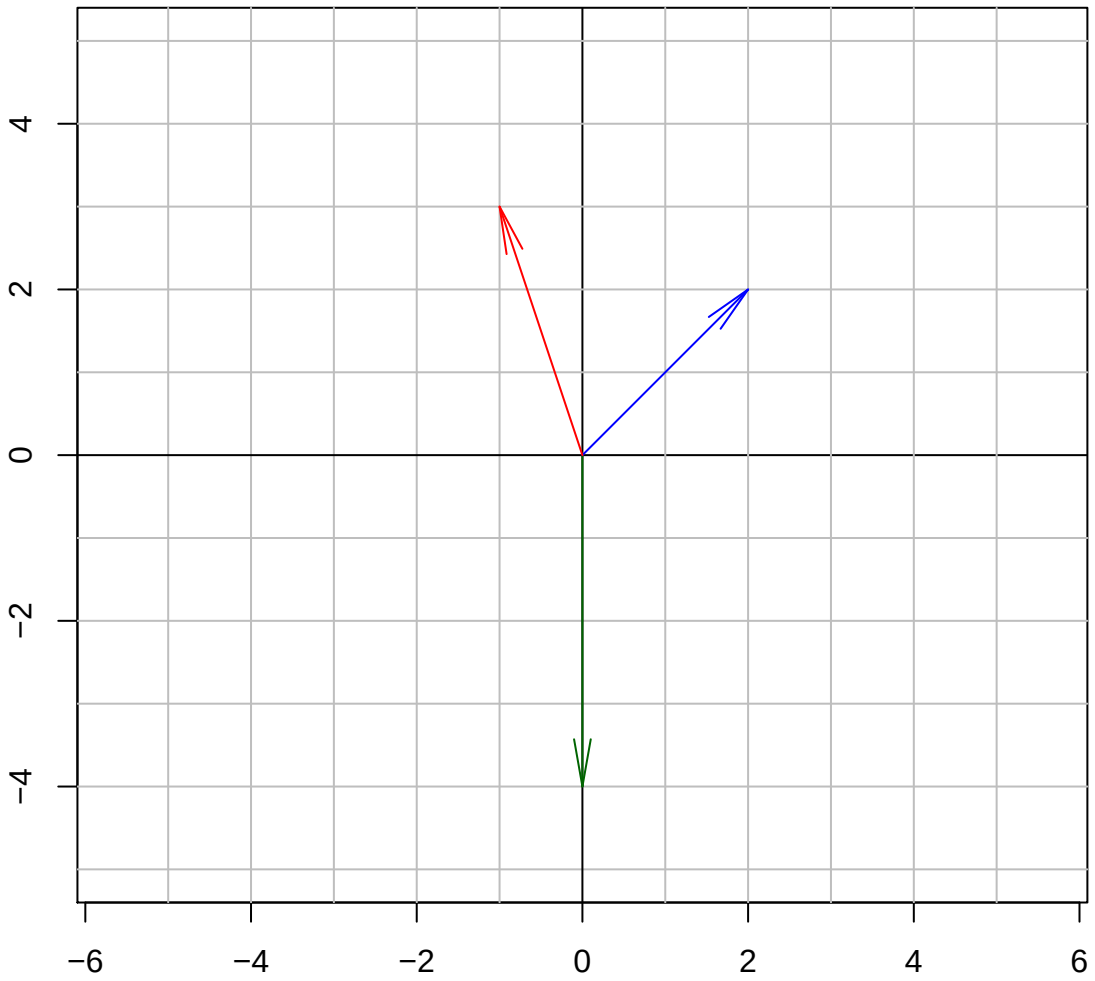
Uma outra representação para vetores, talvez mais próxima da realidade dos físicos é imaginar um vetor como uma seta. Para eles, várias quantidades de interesse são naturalmente representadas como vetores, como por exemplo a velocidade de uma partícula em uma trajetória ou o campo gravitacional da Terra. Cada valor na concatenação acima pode ser concebido como a coordenada deste vetor em um eixo, e podemos representar

graficamente cada vetor como uma seta que sai da origem destas coordenadas. Por exemplo, os vetores

$$\vec{u} = (0, -4); \quad \vec{v} = (2, 2); \quad \vec{w} = (-1, 3)$$

podem ser representados graficamente no R usando o seguinte código:

```
# definindo alguns vetores
u <- c(0, -4)
v <- c(2, 2)
w <- c(-1, 3)

# essa pequena função facilita bastante representar vetores na janela gráfica
plotVector <- function(v, color = 'black', width = 1)
    arrows(x0 = 0, y0 = 0, x1 = v[1], y1 = v[2],
           col = color, lwd = width, angle = 10)

# vamos apenas inicializar uma região gráfica vazia;
# xlim e ylim se referem aos limites desta janela;
# xlab e ylab são os nomes de cada eixo (nada, no caso).
plot(NA, xlim = c(-5, 5), ylim = c(-5, 5), xlab = '', ylab = '', asp = 1)

# vamos também desenhar linhas que representam o sistema de coordenadas
for(i in (-5):5)
    {
        abline(h = i, col = ifelse(i != 0, 'grey', 'black'))
        abline(v = i, col = ifelse(i != 0, 'grey', 'black'))
    }

# agora, podemos usar a função que desenha vetores
plotVector(u, 'darkgreen')
plotVector(v, 'blue')
plotVector(w, 'red')
```



Importante: Idealmente, para todo código neste tutorial, seria bom você colá-lo em um *script* no RStudio e executá-lo mudando algumas coisas só para ver o que acontece. No caso deste código acima, você pode plotar outros vetores, por exemplo, ou talvez mudar as cores.

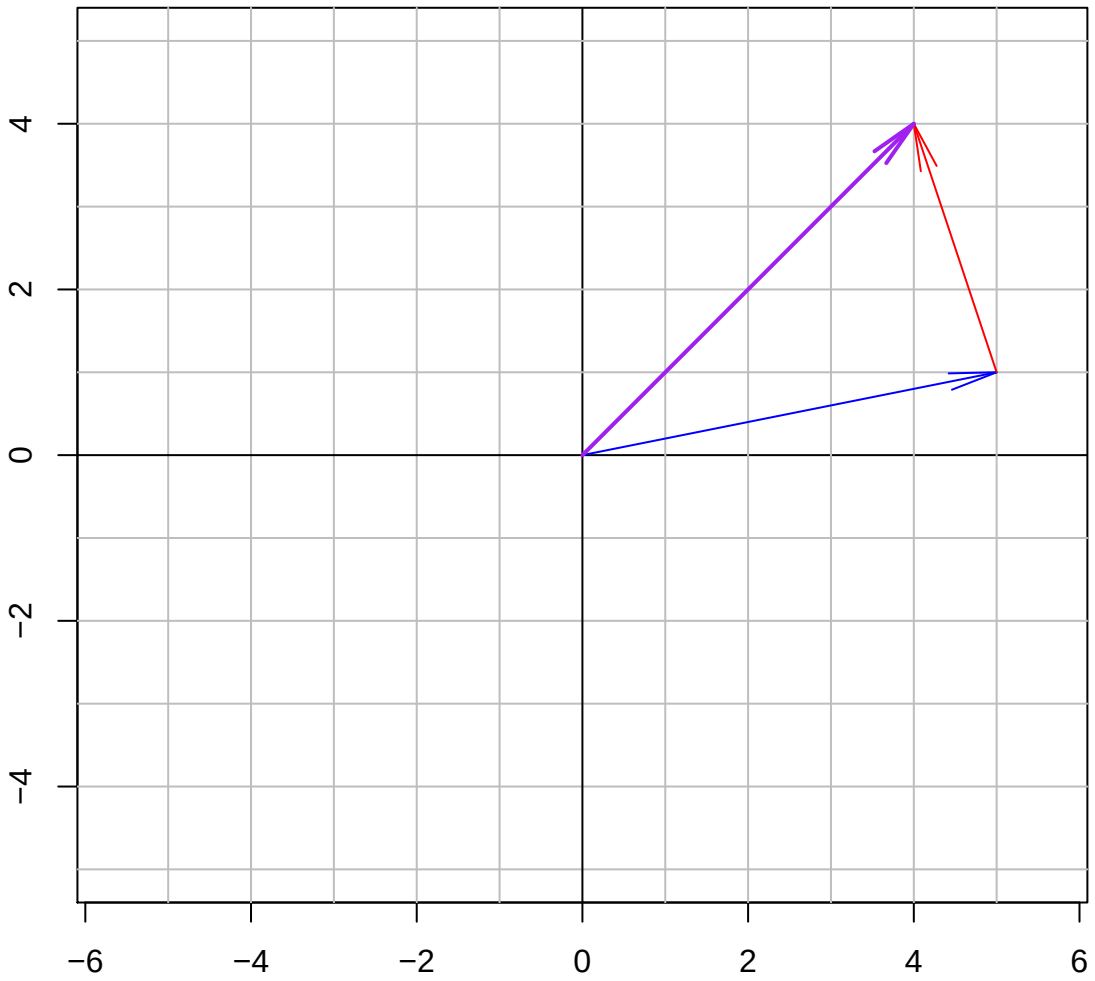
Uma terceira concepção é mais próxima da realidade dos matemáticos, que em geral pensam as coisas na forma de conjuntos, as operações que podemos fazer com elementos neste conjunto, e as propriedades destas operações. Nesta concepção, a representação mais adequada para um vetor é aquela que facilita sua compreensão do que está acontecendo. No entanto, é importante notar que, qualquer que seja a operação feita com um vetor ou vetores, haverá duas formas de imaginar esta operação: uma relacionada a representá-los sob a forma de listas de números, outra relacionada a representá-los como setas.

Inicialmente, podemos fazer duas coisas com vetores: somá-los e multiplicá-los por números. Para somar dois vetores, a forma mais simples de pensar a princípio é simplesmente somando suas entradas correspondentes individualmente:

$$\vec{u} = (u_1, u_2); \vec{v} = (v_1, v_2) \Rightarrow \vec{u} + \vec{v} = (u_1 + v_1, u_2 + v_2)$$

Além disso, podemos pensar nesta soma como uma concatenação de ações. Neste código, mostramos que a soma de dois vetores, quando representados sob a forma de setas, corresponde a andar na direção e distância dada pelo primeiro vetor, depois andar a direção e distância dada pelo segundo vetor.

```
u <- c(5, 1)
v <- c(-1, 3)

# esta função desenha um vetor 'v' a partir da ponta de outro vetor 'u'
plotDislocatedVector <- function(v, u, color = 'black', width = 1)
    arrows(x0 = u[1], y0 = u[2], x1 = u[1] + v[1], y1 = u[2] + v[2],
           col = color, lwd = width, angle = 10)

# novamente, vamos inicializar um gráfico vazio
plot(NA, xlim = c(-5, 5), ylim = c(-5, 5), xlab = '', ylab = '', asp = 1)

for(i in (-5):5)
    {
        abline(h = i, col = ifelse(i != 0, 'grey', 'black'))
        abline(v = i, col = ifelse(i != 0, 'grey', 'black'))
    }

plotVector(u, 'blue')
plotDislocatedVector(v, u, 'red')
plotVector(u + v, 'purple', width = 2)
```



Multiplicar um vetor por um número é também bastante direto, basta apenas multiplicar suas entradas individuais pelo número:

$$\vec{u} = (u_1, u_2) \Rightarrow \alpha \cdot \vec{u} = (\alpha \cdot u_1, \alpha \cdot u_2)$$

Neste contexto, chamamos de *escalar* este valor α que multiplica o vetor $\vec{u}$.

Exercício 1

Tente modificar os pedaços de código utilizados anteriormente para representar graficamente o produto entre vetores e escalares. Depois, responda as seguintes perguntas:

1. O que acontece com um vetor após esta operação?
2. O que acontece se você multiplicar um vetor por um número negativo?

Norma e Direção

Podemos caracterizar um vetor por sua **norma**, ou seja, o comprimento da seta que representa o vetor, e por sua **direção**, ou seja, para onde o vetor aponta. A norma de um vetor pode ser calculada em duas dimensões pela relação de Pitágoras:

$$\|\vec{u}\| = \sqrt{u_1^2 + u_2^2}$$

mas, independente do número de dimensões representadas no vetor, a norma de um vetor é obtida como sendo a raiz quadrada da soma de todas as coordenadas ao quadrado.

Uma operação que fazemos comumente é *normalizar* um vetor, isto é, dividir os elementos do vetor por sua norma, de modo que a norma deste novo vetor será unitária.

Exercício 2

Leia atentamente e execute o código a seguir:

```
require(evolqg)

# o pacote evolqg possui uma função que normaliza um vetor
?Normalize

# vamos gerar alguns vetores aleatórios em duas dimensões

# primeiro, vamos gerar alguns números advindos de uma distribuição normal padrão
rand.vec <- rnorm(100)

# agora, vamos organizar esses números como vetores em uma tabela
rand.vec <- matrix(rand.vec, nrow = 50, ncol = 2)

# cada vetor aleatório corresponde a uma linha desta tabela
# quero normalizar cada vetor, utilizando a função aaply
?aaply
norm.vec <- aaply(rand.vec, 1, Normalize)
```

```r
# agora, vamos plotar estes vetores
# aqui, não vou me preocupar em representá-los como setas
# apenas o ponto no plot entre suas coordenadas é suficiente
plot(norm.vec, asp = 1, pch = 20)
```

1. Os pontos parecem desenhar uma forma em particular? Que forma é essa?
2. Aumente o número de vetores gerados para mil.
3. Você consegue elaborar uma razão para esta forma estar sendo gerada?

Bases

Bases são um conceito bastante simples, mas que possui muitas consequencias importantes. De maneira simples, posso escrever um vetor qualquer no espaço bidimensional da seguinte forma:

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \Rightarrow \vec{u} = u_1 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} + u_2 \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

introduzindo também a notação de vetores como matrizes-coluna. É importante notar que qualquer vetor bidimensional pode ser escrito desta forma, como uma *combinação linear* dos vetores $(1,0)$ e $(0,1)$. Basta substituir os valores u_1 e u_2 por quaisquer pares de números, e eu obtenho um vetor no espaço bidimensional. Estes dois vetores são importantes o bastante para receber nomes especiais:

$$\hat{\imath} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \hat{\jmath} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Juntos, $\hat{\imath}$ e $\hat{\jmath}$ constituem o que chamamos de *base canônica* do espaço bidimensional (que, pra economizar palavras, vamos chamar de $\mathbb{R}^2$ a partir de agora). Daí podemos escrever $\vec{u}$ como

$$\vec{u} = u_1\hat{\imath} + u_2\hat{\jmath}$$

que é mais uma forma de representar um vetor. Como $\hat{\imath}$ e $\hat{\jmath}$ podem ser combinados de modo a produzir qualquer vetor no $\mathbb{R}^2$, dizemos que estes vetores *geram* o espaço.

Mas, somente estes dois vetores são capazes de gerar o espaço bidimensional?

Existem pares de vetores que não são capazes de fazê-lo?

Exercício 3

1. Escreva o vetor $(5, 1)$ como uma combinação linear dos vetores $(-2, 1)$ e $(1, 3)$.

- *Dica*: Vamos escrever o problema:

$$\begin{bmatrix} 5 \\ 1 \end{bmatrix} = \alpha \begin{bmatrix} -2 \\ 1 \end{bmatrix} + \beta \begin{bmatrix} 1 \\ 3 \end{bmatrix}.$$

- É possível achar valores para α e β?

```
u <- c(-2, 1)

v <- c(1, 3)

w <- c(5, 1)


# aqui vão suas soluções

# descomente o código para executar essas linhas quando você achar as soluções

# alpha <-

# beta <-


plot(NA, xlim = c(-5, 5), ylim = c(-5, 5), xlab = '', ylab = '', asp = 1)


for(i in (-5):5)

    {

        abline(h = i, col = ifelse(i != 0, 'grey', 'black'))

        abline(v = i, col = ifelse(i != 0, 'grey', 'black'))

    }


# confira sua solução

# enquanto vc não achar alpha e beta e atribuir valores

# esse código não vai funcionar

plotVector(alpha * u, 'darkgreen')

plotDislocatedVector(beta * v, alpha * u, 'blue')

plotVector(w, 'red', width = 2)
```

```
# se estiver certo a ponta dos vetores vai coincidir
```

2. Escreva o mesmo vetor $(5, 1)$ como combinação linear dos vetores $(-2, 1)$ e $(2, -1)$.

```
# o mesmo código deve ajudar você a entender o que está acontecendo nesta situação
u <- c(-2, 1)
v <- c(2, -1)
w <- c(5, 1)

plot(NA, xlim = c(-5, 5), ylim = c(-5, 5), xlab = '', ylab = '', asp = 1)

for(i in (-5):5)
    {
        abline(h = i, col = ifelse(i != 0, 'grey', 'black'))
        abline(v = i, col = ifelse(i != 0, 'grey', 'black'))
    }

plotVector(u, 'darkgreen')
plotVector(v, 'blue')
plotVector(w, 'red', width = 2)
```

Matrizes

Intuitivamente, matrizes são tabelas de números, algo como

$$\mathrm{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}.$$

Neste tutorial, gostaria de enfatizar seu papel geométrico como transformação linear, ao invés de focar na álgebra, como fazem na Educação Básica. Neste contexto, o que são trasformações? Você pode pensar em uma função, como $f(x) = x^2$, que recebe um número como entrada e retorna um número como saída. Matrizes podem ser pensadas como funções quando observamos seu efeito sobre um vetor. Assim, definimos uma função $f(\vec{v}) = \mathrm{A}\vec{v}$, que recebe um vetor $\vec{v}$ e retorna um outro vetor $\vec{w} = f(\vec{v})$. Neste momento inicial, não importa como exatamente

computamos esta função. Por exemplo, considere a matriz

$$\mathrm{R} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}.$$

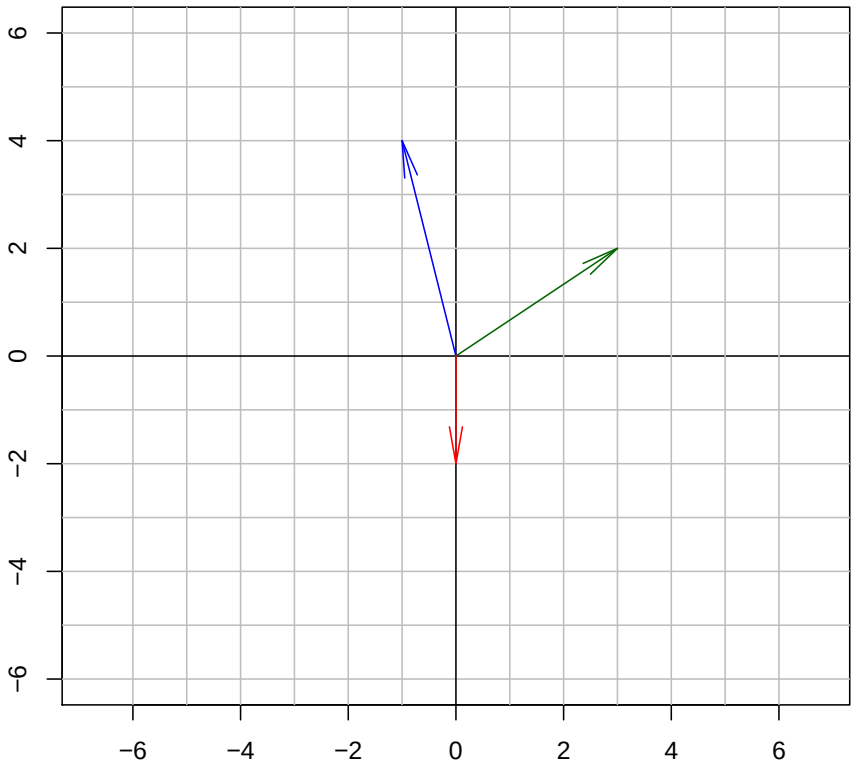
Vamos ver o que ela faz com um vetor? Ou com vários vetores.

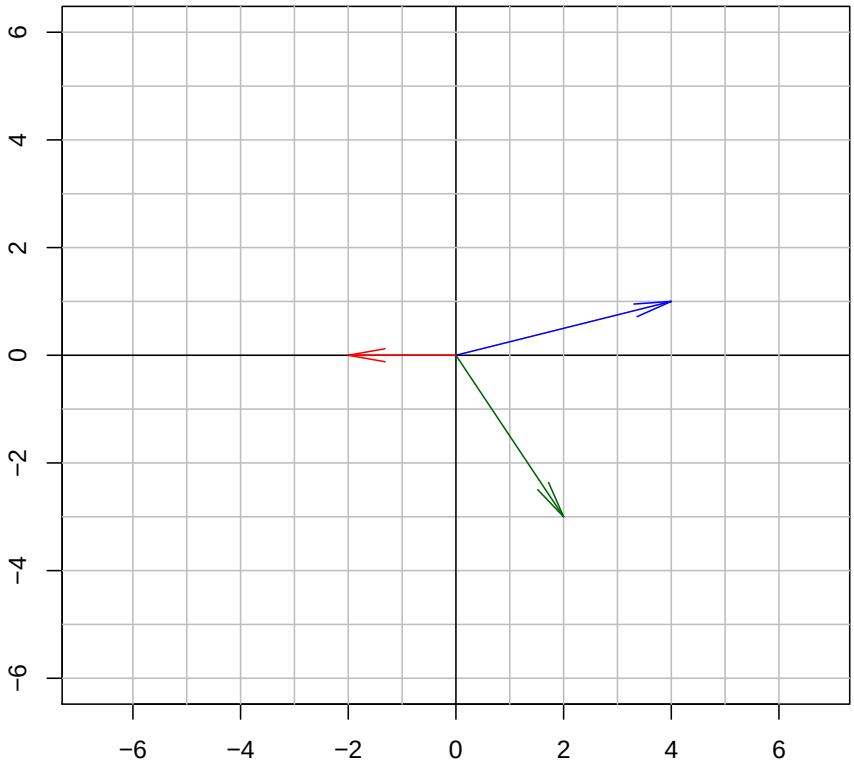
```
## primeiro vamos definir R
R <- matrix(c(0, 1,
              -1, 0), byrow = TRUE, nrow = 2)

## e dar uma olhada nesta matriz
R
```

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

```
## vamos definir alguns vetores
u <- c(3, 2)
v <- c(-1, 4)
w <- c(0, -2)

## e operar sobre eles com R
f.u <- R %*% u
f.v <- R %*% v
f.w <- R %*% w

## essa operação aí em cima representa a multiplicação de matrizes e vetores
## por isso eu disse que você não precisa se preocupar com isso

## vou usar esse código bastante, então vamos fazer uma pequena função
plotEmptyCanvas <- function()
    {
        plot(NA, xlim = c(-6, 6), ylim = c(-6, 6), xlab = '', ylab = '', asp = 1)
```

```
        for(i in (-6):6)
        {
            abline(h = i, col = ifelse(i != 0, 'grey', 'black'))
            abline(v = i, col = ifelse(i != 0, 'grey', 'black'))
        }
    }

par(mfrow = c(2, 1))

# plotar os vetores que entram
plotEmptyCanvas()
plotVector(u, 'darkgreen')
plotVector(v, 'blue')
plotVector(w, 'red')

# e os vetores que saem
plotEmptyCanvas()
plotVector(f.u, 'darkgreen')
plotVector(f.v, 'blue')
plotVector(f.w, 'red')
```


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

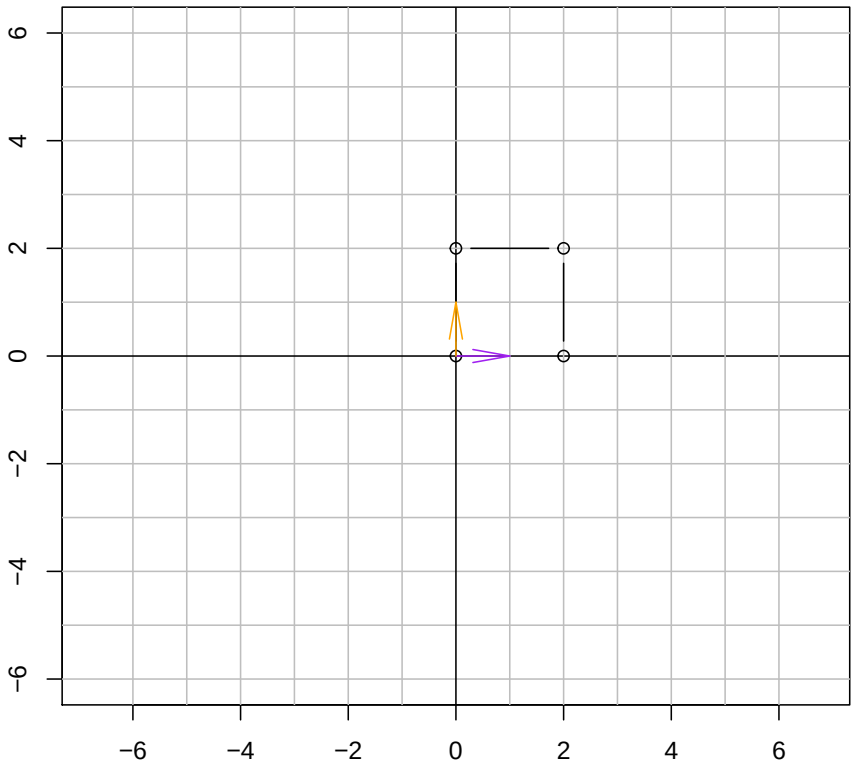
Esta matriz rotaciona cada vetor 90 graus em sentido horário. Não estende ou muda os ângulos entre os vetores. Naturalmente, R é uma *matriz de rotação*. O próximo exemplo deixa mais claro o que entendemos por transformação linear. Considere a matriz

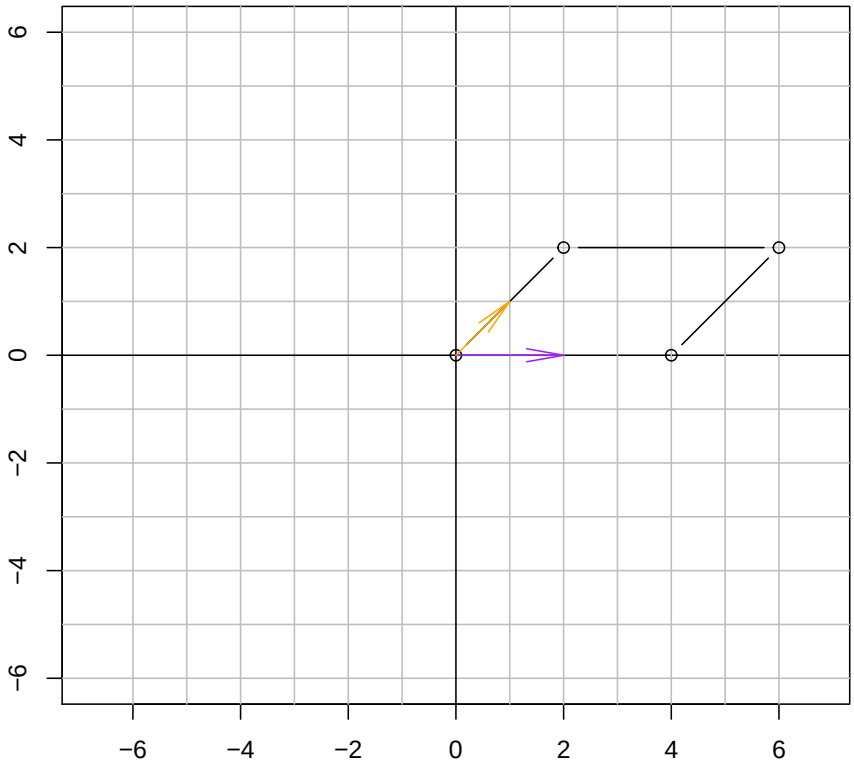
$$S = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}.$$

Vamos ver o que ela faz com um quadrado.

```
## vamos desenhar um quadrado
quadrado <- matrix(c(0, 0,
                     0, 2,
                     2, 2,
                     2, 0), byrow = TRUE, nrow = 4)

S <- matrix(c(2, 1,
              0, 1), byrow = TRUE, nrow = 2)

f.quadrado <- S %*% t(quadrado)
## esse função t() troca as linhas pelas colunas da matriz quadrado
## isso garante que cada aresta do quadrado seja um vetor coluna pra ser multiplicado

f.quadrado <- t(f.quadrado)
## cada aresta do quadrado transformado é uma linha agora

## vamos também olhar para o que acontece com os vetores que definem a base canônica do R2
ihat <- c(1, 0)
jhat <- c(0, 1)

f.ihat <- S %*% ihat
f.jhat <- S %*% jhat

par(mfrow = c(2, 1))

plotEmptyCanvas()
```

```
points(quadrado, type = 'b')
plotVector(ihat, 'purple')
plotVector(jhat, 'orange')


plotEmptyCanvas()
points(f.quadrado, type = 'b')
plotVector(f.ihat, 'purple')
plotVector(f.jhat, 'orange')
```


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

Essa matriz distorce o quadrado, o transformando em um paralelogramo, mas note que esta distorção preserva o paralelismo entre os lados da figura resultante. Esta propriedade é uma das condições que define uma transformação *linear*. Linhas retas na entrada se mantém retas na saída. A outra propriedade importante é que a origem deve ficar onde está após passar pela transformação, mas para estas funções definidas apenas como um um produto entre matriz e vetor, isto é garantido.

No exemplo, também aproveitamos e mostramos o destino dos vetores $\hat{\boldsymbol{\imath}}$ e $\hat{\boldsymbol{\jmath}}$ após a transformação. Estes vetores vão parar em

$$f(\hat{\boldsymbol{\imath}}) = \begin{bmatrix} 2 \\ 0 \end{bmatrix}; f(\hat{\boldsymbol{\jmath}}) = \begin{bmatrix} 1 \\ 1 \end{bmatrix};$$

que são exatamente as colunas da matriz S! De fato, podemos definir qualquer transformação linear a partir do que acontece com $\hat{\boldsymbol{\imath}}$ e $\hat{\boldsymbol{\jmath}}$, e escrever qualquer vetor no espaço transformado como sendo uma combinação linear de $f(\hat{\boldsymbol{\imath}})$ e $f(\hat{\boldsymbol{\jmath}})$:

$$\mathrm{S}\vec{\boldsymbol{v}} = v_1 \begin{bmatrix} 2 \\ 0 \end{bmatrix} + v_2 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

o que deixa muito mais claro o que fazer para multiplicar uma matriz por um vetor.

Note que a gente pode plotar tanto os vetores que entram nestas matrizes quanto os que saem porque ambas são matrizes com duas linhas e duas colunas. Estas matrizes quadradas são especiais porque o espaço de entrada e o espaço de saída possuem o mesmo número de dimensões.

Exercício 4

Vamos considerar o que acontece com composições entre matrizes.

1. Obtenha o produto entre a matriz R, definida no início desta seção, e os vetores $(2, 0)$ e $(1, 1)$. Escreva estes produtos na forma de combinações lineares entre as colunas da matriz R.
2. Leia o código abaixo:

```
### aqui a gente roda o paralelogramo
fg.quadrado <- t(R %*% t(f.quadrado))


plotEmptyCanvas()
points(fg.quadrado, type = 'b')
```

Este código aplica a matriz R sobre a saída de multiplicar as arestas do quadrado por S. Você consegue escrever um código que faça o oposto (pega o quadrado, aplica R, depois aplica S e mostra o resultado)?

3. As duas formas geradas são iguais?