R Exercises in Linear Algebra 2

BIOS13 Modelling Biological Systems, Lund University, Lund, Sweden

Yun-Ting Jang

These exercises are in approximate increasing level of difficulty. Do as many as you please, or as you have time for.

1. Write a script that defines a vector **u** = [3,6,7] and **v** = [12,13,14] and calculates
    a. **u** + **v**
    b. **u**·**v** (the scalar product)
    c. |**u**|
    d. |**v**|
    e. |**u** + **v**|

The correct answers should be a) [15,19,21], b) 212, c) 9.6954, d) 22.5610, e) 32.0468

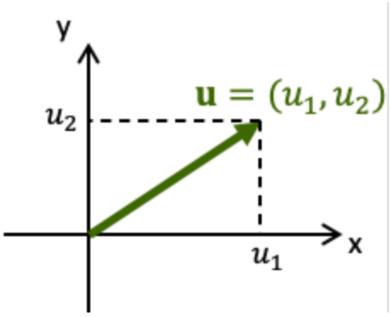
2. In the same script, also define a matrix $B = \begin{pmatrix} 1 & 0 & 4 \\ 2 & 0 & 5 \\ 3 & 0 & 6 \end{pmatrix}$ and calculate (assuming u and v are column vectors)
    a. $B\mathbf{u}$
    b. $B\mathbf{v}$
    c. $B(\mathbf{u} + \mathbf{v})$
    d. $BB\mathbf{u}$

The correct answers should be a) $\begin{pmatrix} 31 \\ 41 \\ 51 \end{pmatrix}$, b) $\begin{pmatrix} 68 \\ 94 \\ 120 \end{pmatrix}$, c) $\begin{pmatrix} 99 \\ 135 \\ 171 \end{pmatrix}$, d) $\begin{pmatrix} 235 \\ 317 \\ 399 \end{pmatrix}$

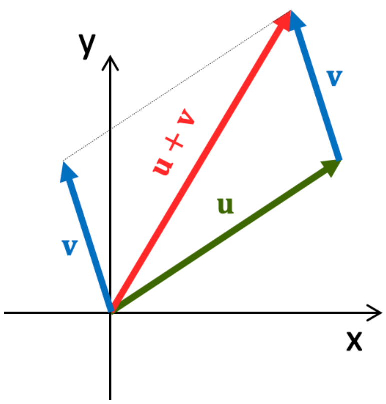
3. Write a new script that defines a matrix $X = \begin{pmatrix} 1 & 2 \\ 1 & 0 \end{pmatrix}$, and calculates
    a. The determinant, $\det(X)$
    b. The two eigenvalues of *X*, λ_1 and λ_2
    c. Can you confirm that the determinant is equal to the product of the two eigenvalues, i.e. that $\det(X) = \lambda_1\lambda_2$?
    d. What are the corresponding eigenvectors, $\mathbf{v}_1$ and $\mathbf{v}_2$?
    e. Can you confirm that $X\mathbf{v}_1 = \lambda_1\mathbf{v}_1$ and that $X\mathbf{v}_2 = \lambda_2\mathbf{v}_2$?

Enough warming up! We will now play with matrices as 'vector functions'. The multiplication of a matrix with a vector, such as $B\mathbf{u}$, results in a new vector. If we write $\mathbf{v} = B\mathbf{u}$, we can think of it as a mathematical function that maps the vector **u** to another vector **v**, just like the function $y = x^2$ maps a number *x* to another number *y*.

Before we get to the fun stuff, we need a geometric interpretation of vectors – as arrows in 2- or 3-dimensional space. The elements of a vector correspond to the coordinates of the tip of the arrow, with the base of the arrow at the origin (0,0):



Vector addition, for example, can be interpreted as stacking the arrows, putting one after the other, but preserving their directions:



A matrix multiplication, $B\mathbf{u}$, can represent different types of *geometric transforms* of vectors.

The simplest example is the *identity matrix*, *I*, which is a matrix with ones on the diagonal and zeros elsewhere. The 2x2 and 3x3 identity matrices look like this:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

The identity matrix doesn't change a vector at all: $I\mathbf{u} = \mathbf{u}$.

Exercise:

4. In R, create a 3x3 identity matrix and show that

$$I \begin{pmatrix} 8 \\ 2 \\ -6 \end{pmatrix} = \begin{pmatrix} 8 \\ 2 \\ -6 \end{pmatrix}.$$

5. Confirm, in R, that the eigenvalues of an identity matrix are all = 1.

Scaling a vector, i.e. changing its length but not its direction, can be implemented with the identity matrix times a scaling constant. For example, the matrix $0.5I$ will cut a vector in half, whereas $3I$ will stretch it to 3 times its former length.

6. In R, create a 2x2 identity matrix and show that

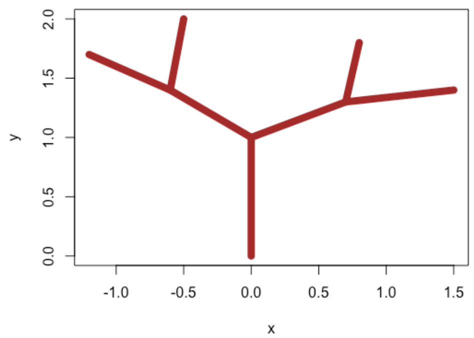
$$0.5I\begin{pmatrix}12\\8\end{pmatrix} = \begin{pmatrix}6\\4\end{pmatrix}.$$

7. Confirm, in R, that the eigen-values of a scaling matrix, such as $0.5I$, are all equal to the scaling constant (0.5 in the example).

Where's the fun??? Ok, lets plot a tree! The following code defines a function, which you can call using `draw_a_tree()` (no input parameters).

```
draw_a_tree <- function() {
  # x-coordinates of nodes in the tree
  # (NA values creates breaks in the line, 'lifting the pen'):
  x <- c(0, 0, 0.7, 1.5, NA, 0.7, 0.8, NA, 0, -0.6, -1.2, NA, -0.6, -
0.5)
  # y-coordinates:
  y <- c(0, 1, 1.3, 1.4, NA, 1.3, 1.8, NA, 1.0, 1.4, 1.7, NA, 1.4,
2.0)
  # Plot a brown tree, with thick branches
  plot(x, y, type='l', col='brown', lwd=8) # lwd sets the line
thickness
}
```

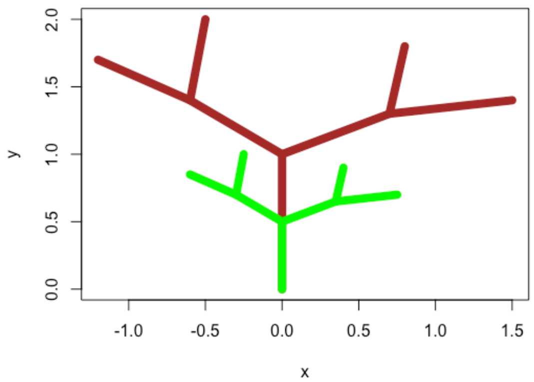
8. Create a script that defines the function above. Calling `draw_a_tree()` should result in something like:



Each node of the tree corresponds to coordinates defined in the x- and y-vectors in the function. For example, the right-most branch-tip has coordinates $(x = 1.5, y = 1.4)$ (check!). Can you find those coordinates in the code? Interpreting the same point as a vector, we get the vector $\mathbf{u} = \begin{pmatrix}1.5\\1.4\end{pmatrix}$. *Scaling* that vector by for example a factor 0.5 would give us a new vector $0.5\mathbf{u} = \begin{pmatrix}0.75\\0.7\end{pmatrix}$.

9. Write a new function `draw_a_small_tree()`, which draws the same tree as above, but half its size (all coordinates scaled by a factor 0.5). Use `lines` instead of `plot`. Make it green instead of brown.

10. Write a script which first calls `draw_a_tree()` and next `draw_a_small_tree()`. The result should be something like



There are many ways to write `draw_a_small_tree()`. One way (not the most efficient) is like this:

```
draw_a_small_tree <- function() {
  # x-coordinates of nodes in the tree
  # (NA values creates breaks in the line, 'lifting the pen'):
  x <- c(0, 0, 0.7, 1.5, NA, 0.7, 0.8, NA, 0, -0.6, -1.2, NA, -0.6, -
0.5)
  # y-coordinates:
  y <- c(0, 1, 1.3, 1.4, NA, 1.3, 1.8, NA, 1.0, 1.4, 1.7, NA, 1.4,
2.0)

  # 2x2 identity matrix:
  I2 <- matrix(c(1,0,0,1),2,2)

  # Scaling matrix:
  A <- 0.5*I2

  # For each vector u =(x[i], y[i]),
  # calculate the corresponding transformed vector v = A*u,
  # and put the new coordinates back into the vectors x, y.
  for (i in 1:length(x)) {
    # original coordinate vector:
    u <- c(x[i],y[i])
    # transformed vector:
    v <- A %*% u
    # replace the original coordinates with the transformed ones:
    x[i] <- v[1]
    y[i] <- v[2]
  }

  # Add a green tree to the current plot
  lines(x, y, type='l', col='green', lwd=8) # lwd sets the line
thickness
}
```

11. Make sure you understand how the code above works.

12. Change the plot-command in `draw_a_tree()` to leave some extra space in the original tree-plot:
    ```
    plot(x, y, type='l', col='brown', lwd=8, xlim=c(-2,2), ylim=c(-2,2))
    ```

13. Write a new function `draw_transformed_tree(A)`, which takes a 2x2 matrix `A` as input and uses that to transform the tree coordinates before plotting (using lines).

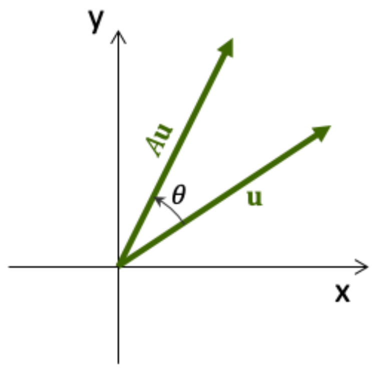
By now, you should be able to draw a big and a small tree from the console like this:

```
> draw_a_tree()
> A <- 0.5*matrix(c(1,0,0,1),2,2)
> draw_a_transformed_tree(A)
```

Time for play!

14. Draw an upside-down-tree:
    ```
    > A <- -1*matrix(c(1,0,0,1),2,2)
    > draw_a_transformed_tree(A)
    ```

An anti-clockwise *rotation* is represented by a matrix $A = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$, where θ is the angle in radians ($radians = \frac{degrees}{360} 2\pi$).



A 30° anti-clockwise rotated tree can be drawn like this:

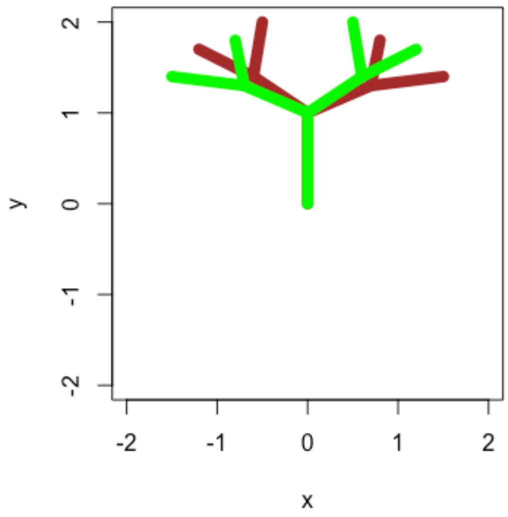
```
> draw_a_tree()
> theta <- 30/360*2*pi
> A <- matrix(c(cos(theta),sin(theta),-sin(theta),cos(theta)), 2, 2)
> draw_a_transformed_tree(A)
```

To rotate clock-wise, use the negative angle: `theta <- -30/360*2*pi`

15. Confirm, in R, that the eigenvalues and eigenvectors of a rotation matrix are complex (with a real, and imaginary part). Complex eigenvalues often correspond to rotations or cyclic dynamics, as we shall see later in the course.

Mirroring in the y-axis is achieved using $A = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$.

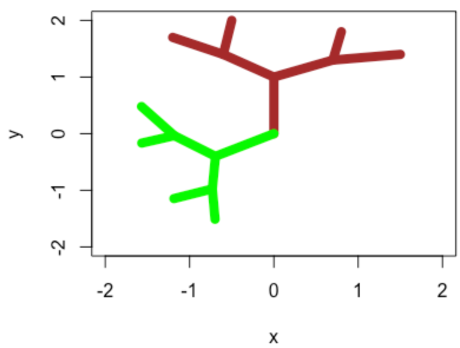
16. Try to make the plot below, where the green tree is a mirror image of the original:



Now you know how to *scale*, *rotate* and *mirror* coordinates using matrices. Each such *transform* can be combined with all the others. A combined transform is simply the matrix multiplication of the two transform matrices. For instance, a scaling by a factor 0.8 followed by a rotation 120° can be achieved like this:

```
> A_scaling <- 0.8*matrix(c(1,0,0,1),2,2)
> theta <- 120/360*2*pi
> A_rotation <- matrix(c(cos(theta),sin(theta),-
sin(theta),cos(theta)),2,2)
> A <- A_rotation %*% A_scaling
```

Notice the order in the matrix multiplication.



17. Free play! Change anything you like (tree shape, color, thickness, …), combine as many trees as you like in a single plot, play with the transforms. The prettiest tree-plot wins ☺.