**Deep Learning Book Series · 2.7 Eigendecomposition**

**26-03-2018  |  HADRIENJ**[**FOLLOW @\_HADRIENJ**](https://twitter.com/_hadrienj?ref_src=twsrc%5Etfw)**|**[**LINEAR-ALGEBRA**](https://hadrienj.github.io/tags#linear-algebra)[**PYTHON**](https://hadrienj.github.io/tags#python)[**NUMPY**](https://hadrienj.github.io/tags#numpy)[**DEEP-LEARNING-BOOK**](https://hadrienj.github.io/tags#deep-learning-book)

[](https://github.com/hadrienj/deepLearningBook-Notes/blob/master/2.7%20Eigendecomposition/2.7%20Eigendecomposition.ipynb)

*Last update: 21/03/2020*

**Introduction**

We will see some major concepts of linear algebra in this chapter. We will start with defining eigenvectors and eigenvalues. We will develop on the idea that a matrix can be seen as a linear transformation and that applying a matrix on its eigenvectors gives new vectors that have the same direction. Then we will see how to express quadratic equations into matrix form. We will see that the eigendecomposition of the matrix corresponding to a quadratic equation can be used to find the minimum and maximum of this function. As a bonus, we will also see how to visualize linear transformations in Python!

Top of Form

**More Math for Data Science? Don't miss new articles.  
And receive a 30-day trial code for the  platform**

****

****

**Get access to the early release of my book**[**"Essential Math for Data Science"**](https://hadrienj.github.io/posts/Essential-Math-for-Data-Science/)**(and to all O'Reilly books 📚). Feedback welcome :)**

Bottom of Form

**2.7 Eigendecomposition**

The eigendecomposition is one form of matrix decomposition. Decomposing a matrix means that we want to find a product of matrices that is equal to the initial matrix. In the case of eigendecomposition, we decompose the initial matrix into the product of its eigenvectors and eigenvalues. Before all, let’s see the link between matrices and linear transformation. Then, you’ll learn what are eigenvectors and eigenvalues.

**Matrices as linear transformations**

As we have seen in [2.3](https://hadrienj.github.io/posts/Deep-Learning-Book-Series-2.3-Identity-and-Inverse-Matrices/) with the example of the identity matrix, you can think of matrices as linear transformations. Some matrices will rotate your space, others will rescale it. When we apply a matrix to a vector, we end up with a transformed version of the vector. When we say that we *apply* the matrix to the vector, it means that we calculate the dot product of the matrix with the vector. We will start with a basic example of this kind of transformation.

**Example 1.**

A = np.array([[-1, 3], [2, -2]])

A

array([[-1, 3],

[ 2, -2]])

v = np.array([[2], [1]])

v

array([[2],

[1]])

First, let’s create a function plotVectors() to easily plot vectors and visualize transformations:

def plotVectors(vecs, cols, alpha=1):

"""

Plot set of vectors.

Parameters

----------

vecs : array-like

Coordinates of the vectors to plot. Each vectors is in an array. For

instance: [[1, 3], [2, 2]] can be used to plot 2 vectors.

cols : array-like

Colors of the vectors. For instance: ['red', 'blue'] will display the

first vector in red and the second in blue.

alpha : float

Opacity of vectors

Returns:

fig : instance of matplotlib.figure.Figure

The figure of the vectors

"""

plt.figure()

plt.axvline(x=0, color='#A9A9A9', zorder=0)

plt.axhline(y=0, color='#A9A9A9', zorder=0)

for i in range(len(vecs)):

x = np.concatenate([[0,0],vecs[i]])

plt.quiver([x[0]],

[x[1]],

[x[2]],

[x[3]],

angles='xy', scale\_units='xy', scale=1, color=cols[i],

alpha=alpha)

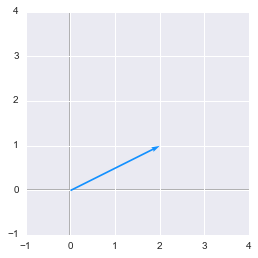
We can now use this function to plot our vector vv:

plotVectors([v.flatten()], cols=['#1190FF'])

plt.ylim(-1, 4)

plt.xlim(-1, 4)

(-1, 4)

***A simple vector***

Now, let’s apply the matrix AA to this vector. We’ll plot the old vector vv (light blue) and the new one (orange):

Av = A.dot(v)

print(Av)

plotVectors([v.flatten(), Av.flatten()], cols=['#1190FF', '#FF9A13'])

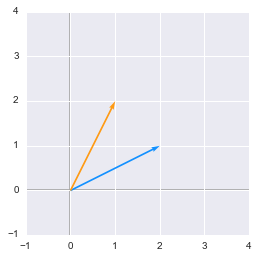
plt.ylim(-1, 4)

plt.xlim(-1, 4)

[[1]

[2]]

(-1, 4)

***The vector***vv***(blue) and its transformed version (orange)***

We can see that applying the matrix AA has the effect of modifying the vector.

You can think of matrices as linear transformation recipes. Let’s see the case of a very special type of vector: the eigenvectors.

**Eigenvectors and eigenvalues**

We have seen an example of a vector transformed by a matrix. Now imagine that the transformation of the initial vector gives us a new vector that has the exact same direction. The scale can be different but the direction is the same. Applying the matrix doesn’t change the direction of the vector. This special vector is called an *eigenvector* of the matrix. We will see that finding the eigenvectors of a matrix can be very useful.

***Imagine that the transformation of the initial vector by the matrix gives a new vector with the exact same direction. This vector is called an eigenvector of***AA***.***

This means that vv is a eigenvector of AA if vv and AvAv (the transformed vector) are in the same direction. The output vector is just a scaled version of the input vector. This scalling factor is λλ which is called the *eigenvalue* of AA.

Mathematically, we have the following equation:

Av=λvAv=λv

The vector AvAv is the vector vv transformed by the matrix AA. This transformed vector is a scaled version (scaled by the value λλ) of the initial vector vv.

**Example 2.**

Let’s AA be the following matrix:

A=[5313]A=[5133]

We know that one eigenvector of AA is:

v=⎡⎣⎢11⎤⎦⎥v=[11]

We can check that Av=λvAv=λv:

[5313][11]=[66][5133][11]=[66]

We can see that:

6×⎡⎣⎢11⎤⎦⎥=⎡⎣⎢66⎤⎦⎥6×[11]=[66]

which means that vv is well an eigenvector of AA. Also, the corresponding eigenvalue is λ=6λ=6.

We can represent vv and AvAv to check if their directions are the same:

A = np.array([[5, 1], [3, 3]])

A

array([[5, 1],

[3, 3]])

v = np.array([[1], [1]])

v

array([[1],

[1]])

Av = A.dot(v)

orange = '#FF9A13'

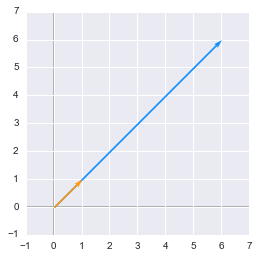
blue = '#1190FF'

plotVectors([Av.flatten(), v.flatten()], cols=[blue, orange])

plt.ylim(-1, 7)

plt.xlim(-1, 7)

(-1, 7)

***An eigenvector of a matrix doesn’t change direction when we apply this matrix***

We can see that their directions are the same!

Another eigenvector of AA is

v=⎡⎣⎢1−3⎤⎦⎥v=[1−3]

because

[5313][1−3]=[2−6][5133][1−3]=[2−6]

and

2×⎡⎣⎢1−3⎤⎦⎥=⎡⎣⎢2−6⎤⎦⎥2×[1−3]=[2−6]

So the corresponding eigenvalue is λ=2λ=2.

v = np.array([[1], [-3]])

v

array([[ 1],

[-3]])

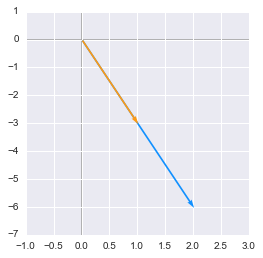
Av = A.dot(v)

plotVectors([Av.flatten(), v.flatten()], cols=[blue, orange])

plt.ylim(-7, 1)

plt.xlim(-1, 3)

(-1, 3)

***Another eigenvector and its transformation***

This example shows that the eigenvectors vv are vectors that change only in scale when we apply the matrix AA to them. Here the scales were 6 for the first eigenvector and 2 to the second but λλ can take any real or even complex value.

**Find eigenvalues and eigenvectors in Python**

Numpy provides a function returning eigenvectors and eigenvalues (the first array corresponds to the eigenvalues and the second to the eigenvectors concatenated in columns):

(array([ 6., 2.]), array([[ 0.70710678, -0.31622777],

[ 0.70710678, 0.9486833 ]]))

Here a demonstration with the preceding example.

A = np.array([[5, 1], [3, 3]])

A

array([[5, 1],

[3, 3]])

np.linalg.eig(A)

(array([ 6., 2.]), array([[ 0.70710678, -0.31622777],

[ 0.70710678, 0.9486833 ]]))

We can see that the eigenvalues are the same than the ones we used before: 6 and 2 (first array).

The eigenvectors correspond to the columns of the second array. This means that the eigenvector corresponding to λ=6λ=6 is:

⎡⎣⎢0.707106780.70710678⎤⎦⎥[0.707106780.70710678]

The eigenvector corresponding to λ=2λ=2 is:

⎡⎣⎢−0.316227770.9486833⎤⎦⎥[−0.316227770.9486833]

The eigenvectors look different because they have not necessarly the same scaling than the ones we gave in the example. We can easily see that the first corresponds to a scaled version of our [1 1][1 1]. But the same property stands. We still have Av=λvAv=λv:

[5313][0.707106780.70710678]=[4.242640694.24264069][5133][0.707106780.70710678]=[4.242640694.24264069]

With 0.70710678×6=4.242640690.70710678×6=4.24264069. So there are an infinite number of eigenvectors corresponding to the eigenvalue 66. They are equivalent because we are interested by their directions.

For the second eigenvector we can check that it corresponds to a scaled version of [1 −3][1 −3]. We can draw these vectors and see if they have the same direction.

v = np.array([[1], [-3]])

Av = A.dot(v)

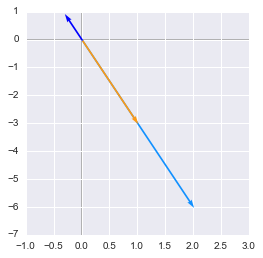
v\_np = [-0.31622777, 0.9486833]

plotVectors([Av.flatten(), v.flatten(), v\_np], cols=[blue, orange, 'blue'])

plt.ylim(-7, 1)

plt.xlim(-1, 3)

(-1, 3)

***Eigenvectors found in numpy have identical directions***

We can see that the vector found with Numpy (in dark blue) is a scaled version of our preceding [1 −3][1 −3].

**Rescaled vectors**

As we saw it with numpy, if vv is an eigenvector of AA, then any rescaled vector svsv is also an eigenvector of AA. The eigenvalue of the rescaled vector is the same.

Let’s try to rescale

v=⎡⎣⎢1−3⎤⎦⎥v=[1−3]

from our preceding example.

For instance,

3v=⎡⎣⎢3−9⎤⎦⎥3v=[3−9]

So we have:

[5313][3−9]=[6−18]=2×[3−9][5133][3−9]=[6−18]=2×[3−9]

This corresponds to A×3v=λ×3vA×3v=λ×3v and the eigenvalue is still λ=2λ=2.

**Concatenating eigenvalues and eigenvectors**

Now that we have an idea of what eigenvectors and eigenvalues are, we can see how it can be used to decompose a matrix. All eigenvectors of a matrix AA can be concatenated in a matrix with each column corresponding to each eigenvector (like in the second array return by np.linalg.eig(A)):

V=[111−3]V=[111−3]

The first column

⎡⎣⎢11⎤⎦⎥[11]

corresponds to λ=6λ=6 and the second

⎡⎣⎢1−3⎤⎦⎥[1−3]

to λ=2λ=2.

The vector λλ can be created from all eigenvalues:

λ=⎡⎣⎢62⎤⎦⎥λ=[62]

Then the eigendecomposition is given by

A=V⋅diag(λ)⋅V−1A=V⋅diag(λ)⋅V−1

***We can decompose the matrix***AA***with eigenvectors and eigenvalues. It is done with:***A=V⋅diag(λ)⋅V−1A=V⋅diag(λ)⋅V−1

diag(v)diag(v) is a diagonal matrix (see [2.6](https://hadrienj.github.io/posts/Deep-Learning-Book-Series-2.6-Special-Kinds-of-Matrices-and-Vectors/)) containing all the eigenvalues. Continuing with our example we have

V=[111−3]V=[111−3]

The diagonal matrix is all zeros except the diagonal that is our vector λλ.

diag(v)=[6002]diag(v)=[6002]

The inverse matrix of VV can be calculated with numpy:

V = np.array([[1, 1], [1, -3]])

V

array([[ 1, 1],

[ 1, -3]])

V\_inv = np.linalg.inv(V)

V\_inv

array([[ 0.75, 0.25],

[ 0.25, -0.25]])

So let’s plug

V−1=[0.750.250.25−0.25]V−1=[0.750.250.25−0.25]

into our equation:

V⋅diag(λ)⋅V−1=[111−3][6002][0.750.250.25−0.25]V⋅diag(λ)⋅V−1=[111−3][6002][0.750.250.25−0.25]

If we do the dot product of the first two matrices we have:

[111−3][6002]=[662−6][111−3][6002]=[626−6]

So if we replace into the equation:

[662−6][0.750.250.25−0.25]=[5313]=A[626−6][0.750.250.25−0.25]=[5133]=A

Let’s check our result with Python:

lambdas = np.diag([6, 2])

lambdas

array([[6, 0],

[0, 2]])

V.dot(lambdas).dot(V\_inv)

array([[ 5., 1.],

[ 3., 3.]])

That confirms our previous calculation.

**Real symmetric matrix**

In the case of real symmetric matrices (more details about symmetric matrices in [2.6](https://hadrienj.github.io/posts/Deep-Learning-Book-Series-2.6-Special-Kinds-of-Matrices-and-Vectors/)), the eigendecomposition can be expressed as

A=QΛQTA=QΛQT

where QQ is the matrix with eigenvectors as columns and ΛΛ is diag(λ)diag(λ).

**Example 3.**

A=[6223]A=[6223]

This matrix is symmetric because A=ATA=AT. Its eigenvectors are:

Q=[0.894427190.4472136−0.44721360.89442719]Q=[0.89442719−0.44721360.44721360.89442719]

and its eigenvalues put in a diagonal matrix gives:

Λ=[7002]Λ=[7002]

So let’s begin with QΛQΛ:

QΛ=[0.894427190.4472136−0.44721360.89442719][7002]=[0.89442719×70.4472136×7−0.4472136×20.89442719×2]=[6.260990333.1304952−0.89442721.78885438]QΛ=[0.89442719−0.44721360.44721360.89442719][7002]=[0.89442719×7−0.4472136×20.4472136×70.89442719×2]=[6.26099033−0.89442723.13049521.78885438]

with:

QT=[0.89442719−0.44721360.44721360.89442719]QT=[0.894427190.4472136−0.44721360.89442719]

So we have:

QΛQT=[6.260990333.1304952−0.89442721.78885438]⋅[0.89442719−0.44721360.44721360.89442719]=[6223]QΛQT=[6.26099033−0.89442723.13049521.78885438]⋅[0.894427190.4472136−0.44721360.89442719]=[6223]

It works! For that reason, it can useful to use symmetric matrices! Let’s do it now with linalg from Numpy:

A = np.array([[6, 2], [2, 3]])

A

array([[6, 2],

[2, 3]])

eigVals, eigVecs = np.linalg.eig(A)

eigVecs

array([[ 0.89442719, -0.4472136 ],

[ 0.4472136 , 0.89442719]])

eigVals = np.diag(eigVals)

eigVals

array([[ 7., 0.],

[ 0., 2.]])

eigVecs.dot(eigVals).dot(eigVecs.T)

array([[ 6., 2.],

[ 2., 3.]])

We can see that the result corresponds to our initial matrix.

**Quadratic form to matrix form**

Eigendecomposition can be used to optimize quadratic functions. We will see that when xx takes the values of an eigenvector, f(x)f(x) takes the value of its corresponding eigenvalue.

***When***xx***takes the values of an eigenvector,***f(x)f(x)***takes the value of its corresponding eigenvalue.***

We will see in the following points how we can show that with different methods.

Let’s have the following quadratic equation:

f(x)=ax21+(b+c)x1x2+dx22f(x)=ax12+(b+c)x1x2+dx22

These quadratic forms can be generated by matrices:

f(x)=[x1x2][acbd][x1x2]=xTAxf(x)=[x1x2][abcd][x1x2]=xTAx

with:

x=[x1x2]x=[x1x2]

and

A=[acbd]A=[abcd]

We call them matrix forms. This form is useful to do various things on the quadratic equation like constrained optimization (see bellow).

***Quadratic equations can be expressed under the matrix form***

If you look at the relation between these forms you can see that aa gives you the number of x21x12, (b+c)(b+c) the number of x1x2x1x2 and dd the number of x22x22. This means that the same quadratic form can be obtained from infinite number of matrices AA by changing bb and cc while preserving their sum.

**Example 4.**

x=⎡⎣⎢x1x2⎤⎦⎥x=[x1x2]

and

A=[2245]A=[2425]

gives the following quadratic form:

2x21+(4+2)x1x2+5x22=2x21+6x1x2+5x222x12+(4+2)x1x2+5x22=2x12+6x1x2+5x22

but if:

A=[29−35]A=[2−395]

we still have the quadratic same form:

2x21+(−3+9)x1x2+5x22=2x21+6x1x2+5x222x12+(−3+9)x1x2+5x22=2x12+6x1x2+5x22

**Example 5**

For this example, we will go from the matrix form to the quadratic form using a symmetric matrix AA. Let’s use the matrix of the example 3.

x=[x1x2]x=[x1x2]

and

A=[6223]A=[6223]

xTAx=[x1x2][6223][x1x2]=[x1x2][6x1+2x22x1+3x2]=x1(6x1+2x2)+x2(2x1+3x2)=6x21+4x1x2+3x22xTAx=[x1x2][6223][x1x2]=[x1x2][6x1+2x22x1+3x2]=x1(6x1+2x2)+x2(2x1+3x2)=6x12+4x1x2+3x22

Our quadratic equation is thus:

6x21+4x1x2+3x226x12+4x1x2+3x22

**Note**

If AA is a diagonal matrix (all 0 except the diagonal), the quadratic form of xTAxxTAx will have no cross term. Take the following matrix form:

A=[acbd]A=[abcd]

If AA is diagonal, then bb and cc are 0 and since f(x)=ax21+(b+c)x1x2+dx22f(x)=ax12+(b+c)x1x2+dx22 there is no cross term. A quadratic form without cross term is called diagonal form since it comes from a diagonal matrix.

**Change of variable**

A change of variable (or linear substitution) simply means that we replace a variable by another one. We will see that it can be used to remove the cross terms in our quadratic equation. Without the cross term, it will then be easier to characterize the function and eventually optimize it (i.e finding its maximum or minimum).

**With the quadratic form**

**Example 6.**

Let’s take again our previous quadratic form:

xTAx=6x21+4x1x2+3x22xTAx=6x12+4x1x2+3x22

The change of variable will concern x1x1 and x2x2. We can replace x1x1 with any combination of y1y1 and y2y2 and x2x2 with any combination y1y1 and y2y2. We will of course end up with a new equation. The nice thing is that we can find a specific substitution that will lead to a simplification of our statement. Specifically, it can be used to get rid of the cross term (in our example: 4x1x24x1x2). We will see later why it is interesting.

Actually, the right substitution is given by the eigenvectors of the matrix used to generate the quadratic form. Let’s recall that the matrix form of our equation is:

x=⎡⎣⎢x1x2⎤⎦⎥x=[x1x2]

and

A=[6223]A=[6223]

and that the eigenvectors of AA are:

[0.894427190.4472136−0.44721360.89442719][0.89442719−0.44721360.44721360.89442719]

With the purpose of simplification, we can replace these values with:

⎡⎣25√15√−15√25√⎤⎦=15√[21−12][25−151525]=15[2−112]

So our first eigenvector is:

15√[21]15[21]

and our second eigenvector is:

15√⎡⎣⎢−12⎤⎦⎥15[−12]

The change of variable will lead to:

[x1x2]=15–√[21−12][y1y2]=15–√[2y1−y2y1+2y2][x1x2]=15[2−112][y1y2]=15[2y1−y2y1+2y2]

so we have

⎧⎩⎨⎪⎪⎪⎪x1=15√(2y1−y2)x2=15√(y1+2y2){x1=15(2y1−y2)x2=15(y1+2y2)

So far so good! Let’s replace that in our example:

xTAx=6x21+4x1x2+3x22=6[15–√(2y1−y2)]2+4[15–√(2y1−y2)15–√(y1+2y2)]+3[15–√(y1+2y2)]2=15[6(2y1−y2)2+4(2y1−y2)(y1+2y2)+3(y1+2y2)2]=15[6(4y21−4y1y2+y22)+4(2y21+4y1y2−y1y2−2y22)+3(y21+4y1y2+4y22)]=15(24y21−24y1y2+6y22+8y21+16y1y2−4y1y2−8y22+3y21+12y1y2+12y22)=15(35y21+10y22)=7y21+2y22xTAx=6x12+4x1x2+3x22=6[15(2y1−y2)]2+4[15(2y1−y2)15(y1+2y2)]+3[15(y1+2y2)]2=15[6(2y1−y2)2+4(2y1−y2)(y1+2y2)+3(y1+2y2)2]=15[6(4y12−4y1y2+y22)+4(2y12+4y1y2−y1y2−2y22)+3(y12+4y1y2+4y22)]=15(24y12−24y1y2+6y22+8y12+16y1y2−4y1y2−8y22+3y12+12y1y2+12y22)=15(35y12+10y22)=7y12+2y22

That’s great! Our new equation doesn’t have any cross terms!

**With the Principal Axes Theorem**

Actually there is a simpler way to do the change of variable. We can stay in the matrix form. Recall that we start with the form:

f(x)=xTAxf(x)=xTAx

The linear substitution can be wrote in these terms. We want replace the variables xx by yy that relates by:

x=Pyx=Py

We want to find PP such as our new equation (after the change of variable) doesn’t contain the cross terms. The first step is to replace that in the first equation:

xTAx=(Py)TA(Py)=yT(PTAP)yxTAx=(Py)TA(Py)=yT(PTAP)y

Can you see the how to transform the left hand side (xx) into the right hand side (yy)? The substitution is done by replacing AA with PTAPPTAP. We also know that AA is symmetric and thus that there is a diagonal matrix DD containing the eigenvectors of AA and such as D=PTAPD=PTAP. We thus end up with:

xTAx=yTDyxTAx=yTDy

All of this implies that we can use DD to simplify our quadratic equation and remove the cross terms. If you remember from example 2 we know that the eigenvalues of AA are:

D=[7002]D=[7002]

xTAx=yTDy=yT[7002]y=[y1y2][7002][y1y2]=[7y1+0y20y1+2y2][y1y2]=7y21+2y22xTAx=yTDy=yT[7002]y=[y1y2][7002][y1y2]=[7y1+0y20y1+2y2][y1y2]=7y12+2y22

That’s nice! If you look back to the change of variable that we have done in the quadratic form, you will see that we have found the same values!

This form (without cross-term) is called the **principal axes form**.

**Summary**

To summarise, the principal axes form can be found with

xTAx=λ1y21+λ2y22xTAx=λ1y12+λ2y22

where λ1λ1 is the eigenvalue corresponding to the first eigenvector and λ2λ2 the eigenvalue corresponding to the second eigenvector (second column of xx).

**Finding f(x) with eigendecomposition**

We will see that there is a way to find f(x)f(x) with eigenvectors and eigenvalues when xx is a unit vector.

Let’s start from:

f(x)=xTAxf(x)=xTAx

We know that if xx is an eigenvector of AA and λλ the corresponding eigenvalue, then Ax=λxAx=λx. By replacing the term in the last equation we have:

f(x)=xTλx=xTxλf(x)=xTλx=xTxλ

Since xx is a unit vector, ∥x∥2=1‖x‖2=1 and xTx=1xTx=1 (cf. [2.5](https://hadrienj.github.io/posts/Deep-Learning-Book-Series-2.5-Norms/) Norms). We end up with

f(x)=λf(x)=λ

This is a usefull property. If xx is an eigenvector of AA, f(x)=xTAxf(x)=xTAx will take the value of the corresponding eigenvalue. We can see that this is working only if the euclidean norm of xx is 1 (i.e xx is a unit vector).

**Example 7**

This example will show that f(x)=λf(x)=λ. Let’s take again the last example, the eigenvectors of AA were

Q=[0.894427190.4472136−0.44721360.89442719]Q=[0.89442719−0.44721360.44721360.89442719]

and the eigenvalues

Λ=[7002]Λ=[7002]

So if:

x=[0.894427190.4472136]x=[0.894427190.4472136]

f(x)f(x) should be equal to 7. Let’s check that’s true.

f(x)=6x21+4x1x2+3x22=6×0.894427192+4×0.89442719×0.4472136+3×0.44721362=7f(x)=6x12+4x1x2+3x22=6×0.894427192+4×0.89442719×0.4472136+3×0.44721362=7

In the same way, if x=[−0.44721360.89442719]x=[−0.44721360.89442719], f(x)f(x) should be equal to 2.

f(x)=6x21+4x1x2+3x22=6×−0.44721362+4×−0.4472136×0.89442719+3×0.894427192=2f(x)=6x12+4x1x2+3x22=6×−0.44721362+4×−0.4472136×0.89442719+3×0.894427192=2

**Quadratic form optimization**

Depending to the context, optimizing a function means finding its maximum or its minimum. It is for instance widely used to minimize the error of cost functions in machine learning.

Here we will see how eigendecomposition can be used to optimize quadratic functions and why this can be done easily without cross terms. The difficulty is that we want a constrained optimization, that is to find the minimum or the maximum of the function for f(x)f(x) being a unit vector.

**Example 8.**

We want to optimize:

f(x)=xTAx subject to ||x||2=1f(x)=xTAx subject to ||x||2=1

In our last example we ended up with:

f(x)=7y21+2y22f(x)=7y12+2y22

And the constraint of xx being a unit vector imply:

||x||2=1⇔x21+x22=1||x||2=1⇔x12+x22=1

We can also show that yy has to be a unit vector if it is the case for xx. Recall first that x=Pyx=Py:

||x||2=xTx=(Py)T(Py)=yTPTPy=yTy=||y||2||x||2=xTx=(Py)T(Py)=yTPTPy=yTy=||y||2

So ∥x∥2=∥y∥2=1‖x‖2=‖y‖2=1 and thus y21+y22=1y12+y22=1

Since y21y12 and y22y22 cannot be negative because they are squared values, we can be sure that 2y22≤7y222y22≤7y22. Hence:

f(x)=7y21+2y22≤7y21+7y22≤7(y21+y22)≤7f(x)=7y12+2y22≤7y12+7y22≤7(y12+y22)≤7

This means that the maximum value of f(x)f(x) is 7.

The same way can lead to find the minimum of f(x)f(x). 7y21≥2y217y12≥2y12 and:

f(x)=7y21+2y22≥2y21+2y22≥2(y21+y22)≥2f(x)=7y12+2y22≥2y12+2y22≥2(y12+y22)≥2

And the minimum of f(x)f(x) is 2.

**Summary**

We can note that the minimum of f(x)f(x) is the minimum eigenvalue of the corresponding matrix AA. Another useful fact is that this value is obtained when xx takes the value of the corresponding eigenvector (check back the preceding paragraph). In that way, f(x)=7f(x)=7 when x=[0.894427190.4472136]x=[0.894427190.4472136]. This shows how useful are the eigenvalues and eigenvector in this kind of constrained optimization.

**Graphical views**

We saw that the quadratic functions f(x)=ax21+2bx1x2+cx22f(x)=ax12+2bx1x2+cx22 can be represented by the symmetric matrix AA:

A=[abbc]A=[abbc]

Graphically, these functions can take one of three general shapes (click on the links to go to the Surface Plotter and move the shapes):

| **1.**[**Positive-definite form**](https://academo.org/demos/3d-surface-plotter/?expression=x*x%2By*y&xRange=-50%2C+50&yRange=-50%2C+50&resolution=49) | **2.**[**Negative-definite form**](https://academo.org/demos/3d-surface-plotter/?expression=-x*x-y*y&xRange=-50%2C+50&yRange=-50%2C+50&resolution=25) | **3.**[**Indefinite form**](https://academo.org/demos/3d-surface-plotter/?expression=x*x-y*y&xRange=-50%2C+50&yRange=-50%2C+50&resolution=49) |
| --- | --- | --- |
| Quadratic function with a positive definite form | Quadratic function with a negative definite form | Quadratic function with a indefinite form |

With the constraints that xx is a unit vector, the minimum of the function f(x)f(x) corresponds to the smallest eigenvalue and is obtained with its corresponding eigenvector. The maximum corresponds to the biggest eigenvalue and is obtained with its corresponding eigenvector.

**Conclusion**

We have seen a lot of things in this chapter. We saw that linear algebra can be used to solve a variety of mathematical problems and more specifically that eigendecomposition is a powerful tool! However, it cannot be used for non square matrices. In the next chapter, we will see the Singular Value Decomposition (SVD) which is another way of decomposing matrices. The advantage of the SVD is that you can use it also with non-square matrices.

**BONUS: visualizing linear transformations**

We can see the effect of eigenvectors and eigenvalues in linear transformation. We will see first how linear transformation works. Linear transformation is a mapping between an input vector and an output vector. Different operations like projection or rotation are linear transformations. Every linear transformations can be though as applying a matrix on the input vector. We will see the meaning of this graphically. For that purpose, let’s start by drawing the set of unit vectors (they are all vectors with a norm of 1).

t = np.linspace(0, 2\*np.pi, 100)

x = np.cos(t)

y = np.sin(t)

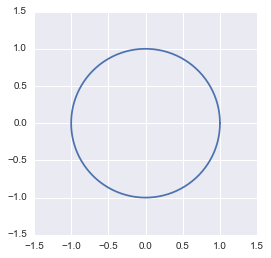
plt.figure()

plt.plot(x, y)

plt.xlim(-1.5, 1.5)

plt.ylim(-1.5, 1.5)

plt.show()

***Unit circle***

Then, we will transform each of these points by applying a matrix AA. This is the goal of the function bellow that takes a matrix as input and will draw

* the origin set of unit vectors
* the transformed set of unit vectors
* the eigenvectors
* the eigenvectors scalled by their eigenvalues

def linearTransformation(transformMatrix):

orange = '#FF9A13'

blue = '#1190FF'

# Create original set of unit vectors

t = np.linspace(0, 2\*np.pi, 100)

x = np.cos(t)

y = np.sin(t)

# Calculate eigenvectors and eigenvalues

eigVecs = np.linalg.eig(transformMatrix)[1]

eigVals = np.diag(np.linalg.eig(transformMatrix)[0])

# Create vectors of 0 to store new transformed values

newX = np.zeros(len(x))

newY = np.zeros(len(x))

for i in range(len(x)):

unitVector\_i = np.array([x[i], y[i]])

# Apply the matrix to the vector

newXY = transformMatrix.dot(unitVector\_i)

newX[i] = newXY[0]

newY[i] = newXY[1]

plotVectors([eigVecs[:,0], eigVecs[:,1]],

cols=[blue, blue])

plt.plot(x, y)

plotVectors([eigVals[0,0]\*eigVecs[:,0], eigVals[1,1]\*eigVecs[:,1]],

cols=[orange, orange])

plt.plot(newX, newY)

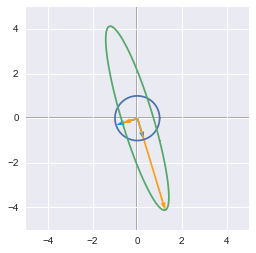
plt.xlim(-5, 5)

plt.ylim(-5, 5)

plt.show()

A = np.array([[1,-1], [-1, 4]])

linearTransformation(A)

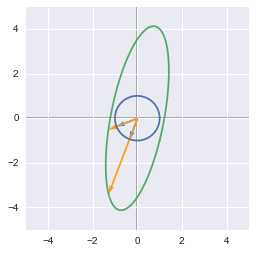
***The unit circle and its transformation by the matrix A. The vectors are the eigenvectors of A.***

We can see the unit circle in dark blue, the non scaled eigenvectors in light blue, the transformed unit circle in green and the scaled eigenvectors in yellow.

It is worth noting that the eigenvectors are orthogonal here because the matrix is symmetric. Let’s try with a non-symmetric matrix:

A = np.array([[1,1], [-1, 4]])

linearTransformation(A)

***The unit circle and its transformation by the matrix A. The vectors are the eigenvectors of A (with A non symmetric).***

In this case, the eigenvectors are not orthogonal!