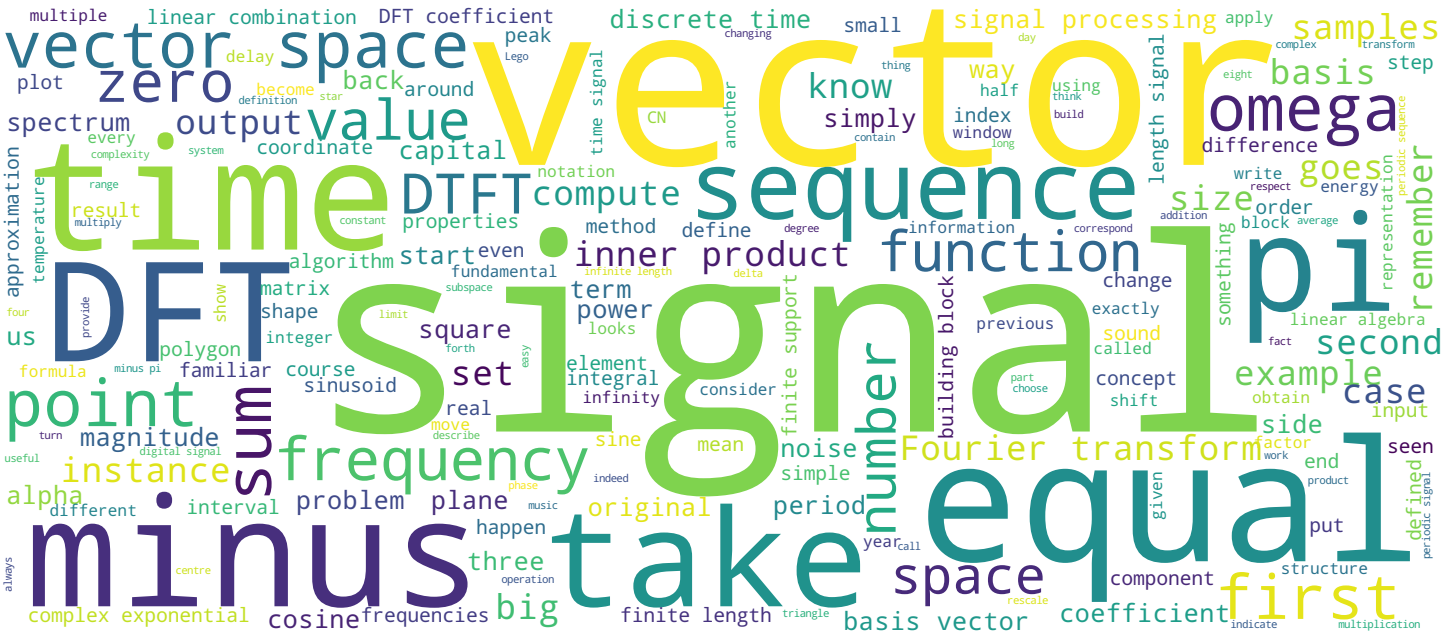
Prerequisite Warning!

$$\begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,N-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M-1,0} & a_{M-1,1} & \cdots & a_{M-1,N-1} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{M-1} \end{bmatrix} = \mathbf{Ax}$$



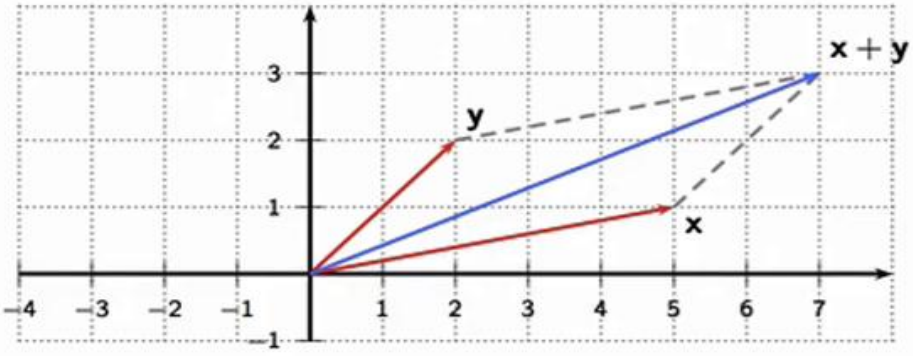
Search MOOC

Video

EPFL

Prerequisite Warning!

$$\mathbf{x}+\mathbf{y}=\begin{bmatrix}x_0+y_0 & x_1+y_1\end{bmatrix}^T$$



2

From now on, we're going to get progressively more technical. So here's another prerequisite warning. This is a matrix vector multiplication in explicit form and in compact form. If you're not familiar with this concept, if you're not comfortable with this notation, perhaps it's time to revise a bit your linear algebra textbooks. Similarly, this is the standard parallelogram rule for vector addition in the plane. Again, if you're not familiar with this concept, a brief overview of geometry and linear algebra will certainly help in what's to follow.

Notes

Summary

0m 00s

A generic discrete time signal...

$$x[n] = \ldots, 1.2390, -0.7372, 0.8987, 0.1798, -1.1501, -0.2642 \ldots$$

3

A generic discrete time signal can be indicated by this expression here. We have a set of numbers ordered in a sequence.

Notes

Summary

0m 35s

What about the details?

- finite length?
- infinite length?
- periodic?
- finite support?

We need a common framework: vector space

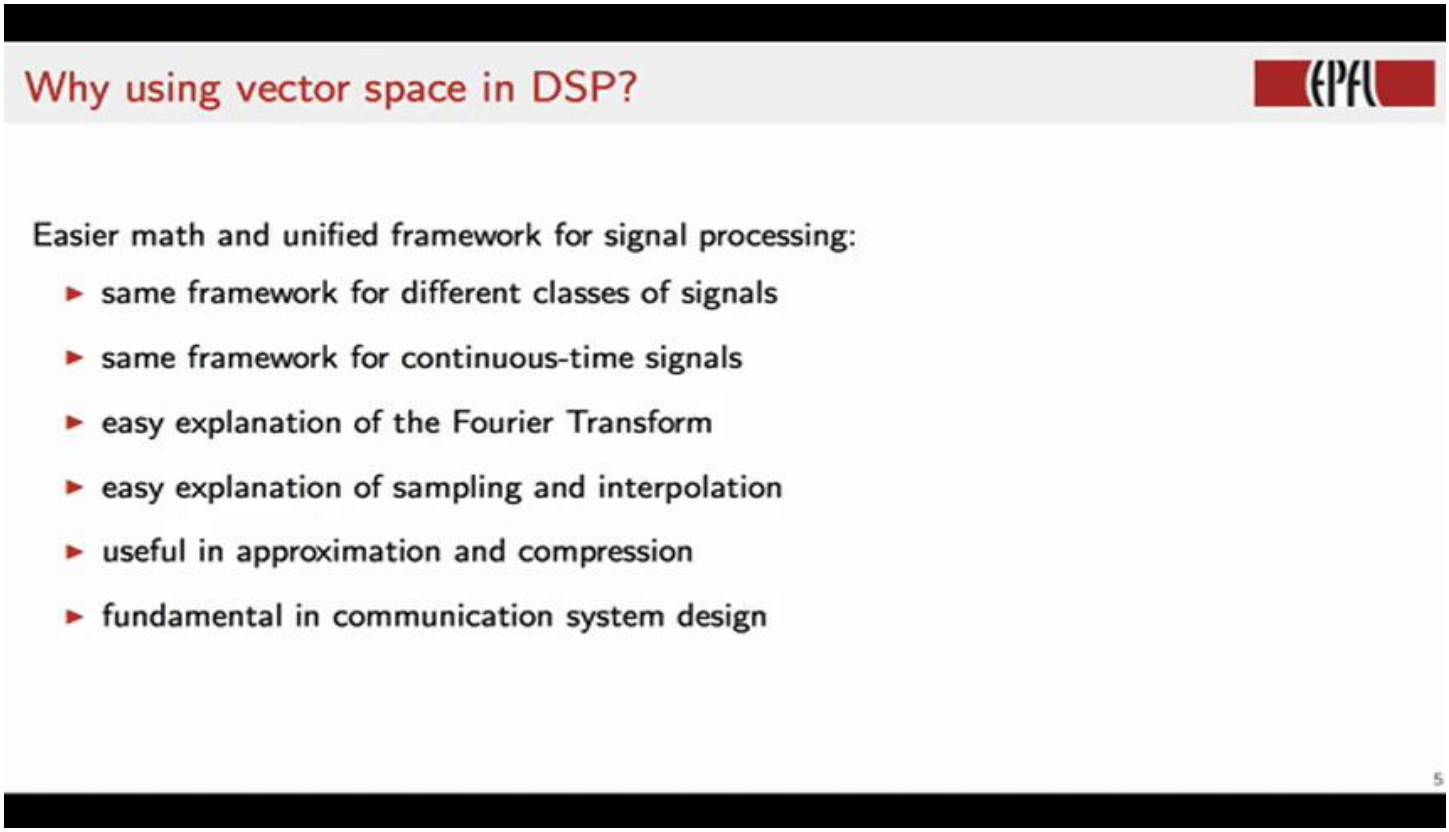
4

But we have seen that we have already four different categories of signals: finite length, infinite length, periodic, finite support. It would be very complicated to develop the whole of signal processing theory if every time we had to stop and specialise what we're saying with respect to the four categories that we see here. So we need a common framework in which we can talk about signals without worrying about which category they belong to.

Notes

Summary

0m 43s

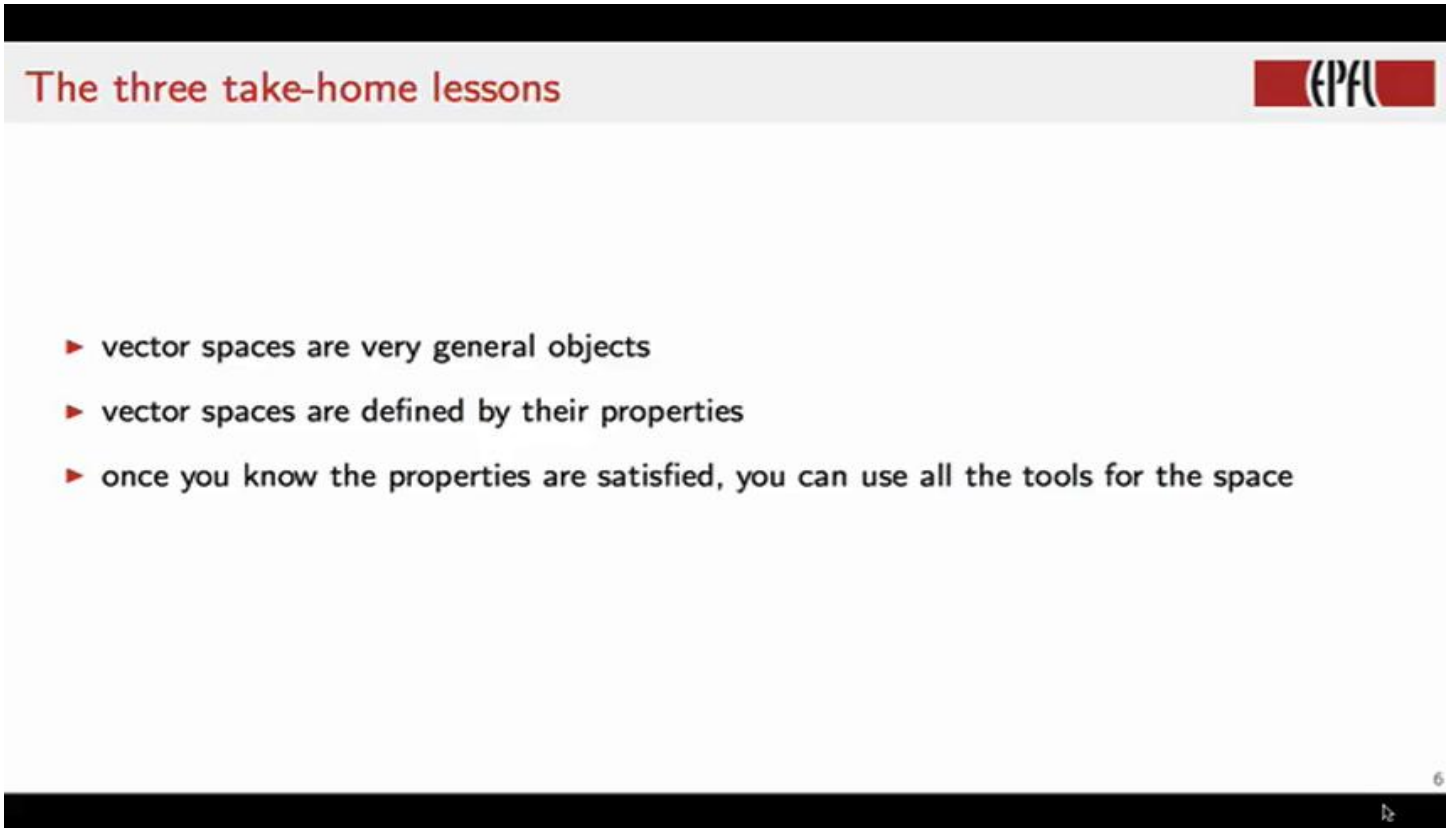


This common framework is provided by vector space and linear algebra. It is very convenient because it provides the same framework for different classes of signals and also for continuous-time signals. It will provide an easy explanation of the Fourier transform, an easy explanation of the sampling theorem and the interpolation theorem. It will be useful in approximation and compression applications, and it's fundamental in designing communication systems. So all these advantages form too long a list not to use vector space in signal processing.

Notes

Summary

1m 10s



The three take-home lessons that I would like you to remember from the next modules is that vector spaces are very general objects and vector spaces are defined by their properties, not by the shape of the vectors that they contain. Once you know that the properties of vector space are satisfied, then you can use all the tools for the space in all spaces. That's really the power of generalisation of vector space that will help us in dealing with different categories of signals.

Notes

Summary

1m 44s

Analogy #1: OOP

```
class Polygon(object):
    def __init__(self, num_sides, side_len=1, x=0, y=0):
        self.num_sides = num_sides
        self.side_len = side_len
        self.center = (x, y)

    def resize(self, factor):
        self.side_len *= factor

    def translate(self, x, y):
        self.center[0] += x
        self.center[1] += y

    def plot(self):
        ...
```

7

If you are familiar with object-oriented programming, perhaps you can think of vector space as of an abstract interface class. Suppose you define a class called Polygon and every object derived from this class will have to have a number of sides, a length for the side, and the coordinate of the centre of the polygon. Then you can define some methods that describe how you can manipulate these polygons. For instance, you can resize a polygon by changing the size of the side, or you can translate the polygon on the plane by changing the coordinates of the centre. These methods will apply to all derived objects independently on the number of sides.

Notes

Summary

2m 12s

Analogy #1: OOP

```
class Triangle(Polygon):
    def __init__(self):
        super(Triangle, self).__init__(3)

    ...


class Square(Polygon):
    def __init__(self):
        super(Square, self).__init__(4)

    ...
```

8

So when you derive objects like triangles, all you need to do is instantiate the number of sides for a triangle, but the methods will remain the same. A square will have four sides and so on and so forth. Just like in the interface paradigm, a vector space dictates how you can combine vectors together, how you can rescale them, but does not pose any limit on the internals of the vectors, on the implementation of the vectors. A vector space, however, does more than just specifying methods.

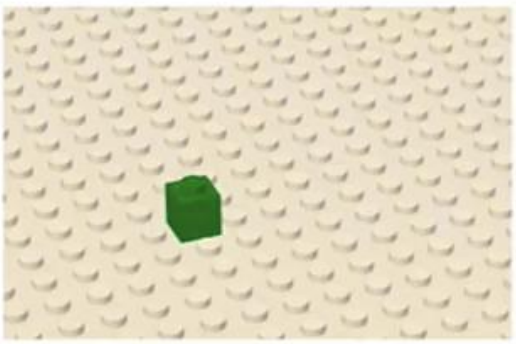
Notes

Summary

2m 54s

Analogy #2: LEGO

basic building block:



9

It imposes a structure on the space. Here, perhaps another tenuous analogy is with Lego. In Lego, you have a basic building block, and what you can do is rescale this building block, so have blocks that are of a different size, and then you can combine these blocks together according to the rules of the game.

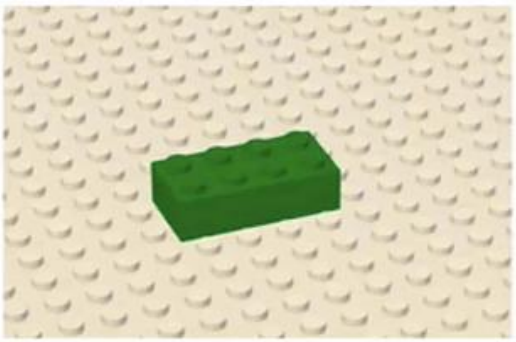
Notes

Summary

3m 25s

Analogy #2: LEGO

scaling (4x2):



10

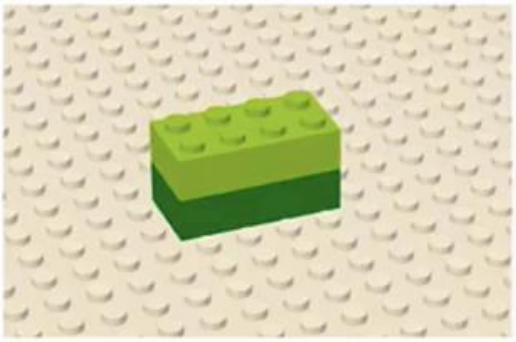
Notes

Summary

3m 37s

Analogy #2: LEGO

adding:



11

We will see that in vector space, the concept of basis will be the equivalent of the Lego minimal building block. By decomposing an arbitrarily complex vector into a linear combination of basis vectors, we will be able to gain profound insight on the properties of the vectors and of the space.

Notes

Summary

3m 44s