Kernels

Kernels in Set language

Given Ω-modules V and W and a linear map $\varphi : V \to W$,

$$\ker \varphi = \{v \in V \mid \varphi(v) = 0\}.$$

This definition is concise because it's meaning is outsourced to the concept of a set. Let us see how far this gets us. For example, suppose we are given φ by a matrix

$$\Phi = \begin{bmatrix} 1 & 1 & 0 & 3 \\ 1 & 0 & 2 & 5 \\ 1 & 1 & 0 & 3 \end{bmatrix}$$

So what is in $\ker \varphi$? The set offers no answer.

Kernels in Computer Algebra Systems

Let us input a linear mapping (as a matrix) into a computer algebra system.

In [1]
```
Phi = [ 1 1 1; 1 0 1; 0 2 0; 3 5 3 ]
print Phi
```

Out [1]
```
[ 1 1 0 3 ]
[ 1 0 2 5 ]
[ 1 1 0 3 ]
```

Research shows humans make 3-6 errors per hour, no matter what the task is. Why waste any of them on miscalculating? Let us ask a computer for the kernel.

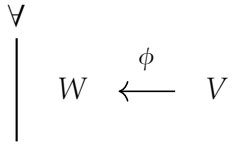
In [2]
```
K = Kernel(Phi)
print K
```

Out [2]
```
[ -2 -5 ]
[  2  2 ]
[  1  0 ]
[  0  1 ]
```

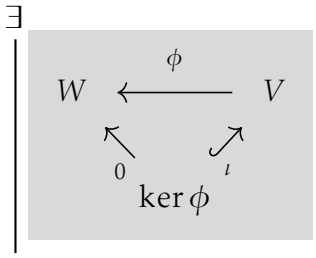
This may be a surprise. This is a matrix, not a set. Why?

Kernels in Diagram language

Let us expand on a possible alternative definition of kernels using diagrams. For all linear maps we can draw a diagram as follows.

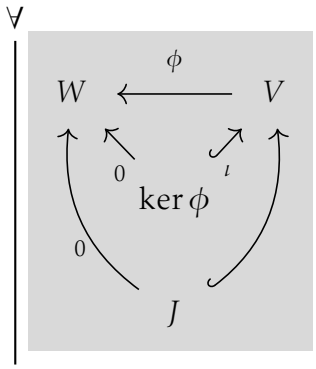


Kernels exist for every linear map, which must be connected to the above diagram, requiring two knew arrows. One arrow is just 0, and the second ι is injective. Notice the arrows ι feeds into the arrow φ which indicates we can compose these two functions $\varphi \circ \iota$. This should equal the other arrow reaching the same point, namely 0. So $\varphi \circ \iota = 0$, or rather $\varphi(\iota(k)) = 0$ for $k \in K$. We say the diagram *commutes* and we indicate this by shading the background.

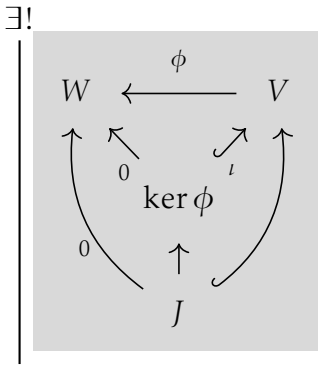


The ι in this diagram is a new linear map, and so it could be given by a matrix. That matrix will in fact be the matrix output by our above code.

But many spaces J and functions $\kappa : J \to V$ could play the role of $\ker\varphi$ shown in the above diagram, including $J = \{0\}$. So this cannot be a complete understanding of kernels. We need $\ker\varphi$ to be a big as possible. To add that to the diagram let us add such a hypothetical J to the diagram.



Now that we have two candidates for kernels we can insist that the first, $\ker\varphi$ is largest by insisting that all others map into it. This completes our diagram.



It is important to see all four diagrams as individually contributing to the concept of a kernel, much like a graphic novel.

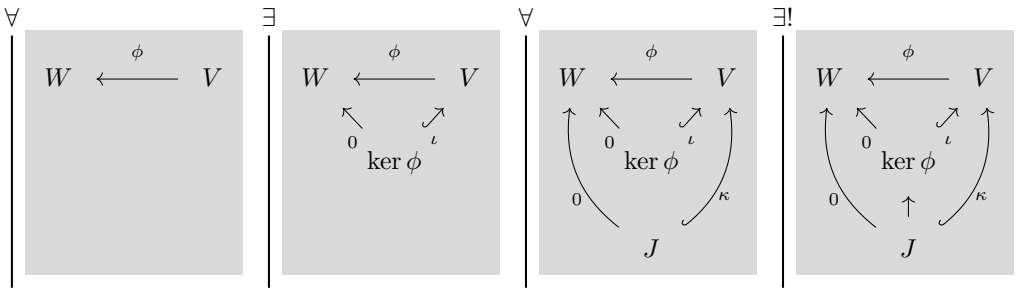


Figure 0.1: The diagram description of kernels.

Kernels in Typed language

The diagram language clarifies how kernels can be thought of as functions and functions that have a maximal quality. Yet, programs do not think in pictures, we do. So we need to translate the same ideas to a syntax we can turn into a program. Here is how.

First the question depends on space V and W and a linear map $\varphi : V \to W$ known first from context, denote ctx or Γ. Using the notation $P \vdash Q$ to say "P leads to Q", also denoted $\frac{P}{Q}$, then we can state this foundation assumptions as follows.

$$\frac{\mathsf{ctx} \vdash V, W : {}_\Omega\mathsf{Mod} \qquad \varphi : \mathsf{Lin}_\Omega(V, W)}{\mathsf{ctx} \vdash \ker \phi : \mathsf{Type}} \qquad (\mathsf{F}_{\ker})$$

The label $\mathsf{F}_{\ker}$ stands for *formation* as we are forming the new type of data $\ker \varphi$. Programs write this in many different forms but one example is as the following.

```
using V,W:Mod[Omega], Phi:Lin[V,W] from ctx
```

Next the diagram above captured the high-level movement of that data without ever considering the actual data. The programs will certainly need these data. The premise from the diagram is that any data $j : J$ which can produce values $\kappa(j) : V$ where $\varphi(\kappa(j)) = 0$ (see the diagrams above). Any such data $j : J$ is meant to produce data in the kernel, because the kernel is the largest such structure. So we include such a rule.

$$\frac{j : J \qquad \kappa : J \to V \qquad pf : \varphi(\kappa(j)) =_W 0}{\mathsf{null}(\kappa(j)) : \ker \varphi} \qquad (\mathrm{I}_{\ker})$$

The $\mathrm{I}_{\ker}$ here is for *introduction* because data is being introduced of the desired type. Programers may recognize the use $pf : \varphi(\kappa(j)) =_W 0$ as what is known as a *guard* or *rail*. It is a type of documentation added to a program to let the programming language enforce that data is used in restricted ways. In this case, no one can introduce a term in the kernel without proof. In code this can be captured in a number of ways, here is one option.

```
class Ker[Phi] {
```

```
    def null(j:J,kappa:J->V,
        require Phi(kappa(j)) == 0)
}
```

Now it is time to use data in the kernel. It is clear how this should proceed, anything in the kernel can be mapped to 0 in W or to a value in V which will map to 0 under φ. The rules are therefore as follows.

$$\frac{x : \ker \phi}{0 : W} \qquad \frac{x : \ker \phi}{\iota(x) : V} \tag{$\mathsf{E}_{\ker}$}$$

In code this might be done as follows.

```
class Ker[Phi] {
    def iota:V
    def zero:W
}
```

Such rules are known as *eliminations* as they eliminate the type annotation.

Finally we need to do some computing somewhere and we learn what to compute by inspecting the condition of "commutative diagrams".

$$\frac{j : J \qquad \kappa : J \to V \qquad pf : \varphi(\kappa(j)) =_W 0}{\iota(x) := \kappa(j)} \tag{$\mathsf{C}_{\ker}$}$$

All together this may look like the following.

```
using V,W:Mod[Omega], Phi:Lin[V,W] from ctx
class Ker[Phi] {
    private v:V

    def null(j:J, kappa:J->V,
        require Phi(kappa(j)) == 0) {
        v = kappa(j)
    }
    def iota:V = v
    def zero:W = 0
}
```

Computing a kernel

Algorithms appearing in China around 2000 years ago appear to depict a version of computing kernels. A thousand years later in Irag and Iran the inventers of algebra were solving systems of linear equations which would necessitate the ability to solve kernels as well. In the 1800's Gauss's many contributions to math included a systematic algorithm to solve for kernels. Once you have learned Gauss's method it becomes impossible to think of any other likely solution. For that reason the historic examples are usually conjectured to be the same algorithm.

Let us consider a matrix in which a subset of the columns are an identity matrix. That is, up to possibly permuting the columns the matrix has the form

$$\begin{bmatrix} I_r & M \\ 0 & 0 \end{bmatrix} \in \Delta^{m \times n}$$

Then an answer would be written down with formula requiring no computation:

$$\begin{bmatrix} I_r & M \\ 0 & 0 \end{bmatrix} \begin{bmatrix} -M^\dagger \\ I_{n-r} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\Phi = \begin{bmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -2 & -2 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Uniqueness questions