# *9d.* Slice Views to Reduce Allocations

Martin Alfaro
PhD in Economics

---

## INTRODUCTION

In our previous discussion on Slices and Views, we defined the concept of a **slice** as a subvector of the parent vector `x`. Typical examples of slices are expressions such as `x[1:2]` or `x[x .> 0]`. By default, slices create a copy of the data and therefore incurs memory allocation, with the only exception of slices comprising a single object.

Next, we present an approach to avoiding the overhead of memory allocation. This is based on the concept of **views**, which bypass the need for a copy by directly referencing the parent object. This method is particularly effective when slices are indexed through ranges. However, it's not suitable for slices that employ Boolean indexing, in which case allocations will still occur.

Finally, we demonstrate that **copying data could be faster than using views**, despite the additional memory allocation involved. This seeming paradox arises because creating a vector ensures that elements are stored in a contiguous block in memory, which facilitates more efficient access to them.

## VIEWS OF SLICES

We start showing that views don't allocate memory *if the slice is indexed by a range*. This property can lead to performance improvements over regular slices, which create a copy by default.

**SLICE AS A COPY**

```
x = [1, 2, 3]

foo(x) = sum(x[1:2])           # it allocates ONE vector -> the slice 'x[1:2]'
```

```
julia> @btime foo($x)
  15.015 ns (1 allocation: 80 bytes)
```

**SLICE AS A VIEW**

```
x = [1, 2, 3]

foo(x) = sum(@view(x[1:2]))    # it doesn't allocate
```

```
julia> @btime foo($x)
  1.200 ns (0 allocations: 0 bytes)
```

However, **views under Boolean indexing won't reduce memory allocations or be more performant**. Therefore, don't rely on views of these objects to speed up computations. This fact is illustrated below.

**BOOLEAN INDEX (COPY)**

```
x = rand(1_000)

foo(x) = sum(x[x .> 0.5])
```

```
julia> @btime foo($x)
  662.500 ns (4 allocations: 8.34 KiB)
```

**BOOLEAN INDEX (VIEW)**

```
x = rand(1_000)

foo(x) = @views sum(x[x .> 0.5])
```

```
julia> @btime foo($x)
  759.770 ns (4 allocations: 8.34 KiB)
```

## COPYING DATA MAY BE FASTER

Although views can reduce memory allocations, there are scenarios where copying data can be the faster approach. This is due to an inherent trade-off between memory allocation and data access patterns. On the one hand, newly created vectors store data in contiguous blocks of memory, enabling more efficient CPU access. On the other hand, while views avoid allocation, they require accessing data scattered throughout memory.

In certain cases, the overhead of creating a copy may be outweighed by the benefits of contiguous memory access, making copying the more efficient choice. This possibility is illustrated below.

**COPY**

```
x = rand(100_000)

foo(x) = max.(x[1:2:length(x)], 0.5)
```

```
julia> @btime foo($x)
  30.100 μs (4 allocations: 781.34 KiB)
```

**VIEW**

```
x = rand(100_000)

foo(x) = max.(@view(x[1:2:length(x)]), 0.5)
```

```
julia> @btime foo($x)
  151.700 μs (2 allocations: 390.67 KiB)
```