# *10a.* Overview and Goals

Martin Alfaro
PhD in Economics

---

## INTRODUCTION

The previous chapters started our study of techniques for improving performance. The focus was in particular on type stability and memory allocations, which are not only critical for achieving optimal performance, but also universally applicable. However, specific applications can often benefit from more specialized strategies. This chapter takes a step in this direction by introducing some of these techniques. In particular, we offer two key insights that extend beyond any particular application, and therefore can be applied broadly.

First, any of the new techniques we'll introduce **inherently involve trade-offs**. This occurs because, after applying fundamental optimizations to operate at the performance frontier, any further gains can only be achieved at the expense of precision, safety, or generality. This stands in contrast to fundamental optimizations like type stability and reduced memory allocations, which may hinder readability but don't entail compromises in other respects. The presence of these trade-offs also explains why the techniques aren't part of Julia's default implementation, which consistently prioritizes correctness and safety over speed.

The second important takeaway is related to the concept of **code transformation via macros**. This represents a general strategy that allows developers to implement sophisticated computational algorithms, without requiring users to grasp the underlying complexities for their applications. Macros are particularly well-suited suited for this purpose, as they essentially take expressions in a code and modifies it before compilation. This makes it possible, for example, to identify all operations within a for-loop, subsequently adapting the algorithm for a more efficient computation.