

Notes 3: Types and dispatch

Chris Paciorek

2025-01-28

Table of contents

Introduction

This document is the third of a set of notes, this document focusing on types and method dispatch based on types. The notes are not meant to be particularly complete in terms of useful functions (Google and LLMs can now provide that quite well), but rather to introduce the language and consider key programming concepts in the context of Julia.

Given that, the document heavily relies on demos, with interpretation in some cases left to the reader.

Composite types (structs)

A struct is a collection of named fields, useful for holding information of a particular structure.

Here's a struct meant to contain information about a Gaussian process, with each element in the type also having its own declared type (though that is not required).

Objects of the type are constructed by calling the name of the type with the field values, in order.

```
struct GaussianProcess
    x::Vector
    covFun::Function
    params
end

function expCov(dists, params)
    return params[2]*exp.(-dists / params[1])
end

myGP = GaussianProcess([0:0.01:1;], expCov, (1, .5));
myGP.params
```

(1, 0.5)

It appears one has to know the exact order of the inputs to the constructor and give them by position rather than by name.

Struct constructors

We can create an explicit (*outer*) constructor that gives some flexibility in terms of what inputs the user can provide.

```
function ExpGaussianProcess(x, params)
    GaussianProcess(x, expCov, params)
end

myGP = ExpGaussianProcess([0:0.01:1;], (1, .5));
```

We can use an *inner* constructor to check inputs. This is a bad example because by defining types for the variables (as seen in the original definition of the `GaussianProcess` struct), this can be handled automatically by Julia.

```
struct GaussianProcess2
    x
    covFun
    params
    ## Define the constructor using function creation shorthand and if-else shorthand:
    GaussianProcess2(x, covFun, params) = isa(covFun, Function) ?
        new(x, covFun, params) : error("covFun needs to be a function")
end


myGP = GaussianProcess2([0:0.01:1;], expCov, (1, .5));

myGP = GaussianProcess2([0:0.01:1;], 3, (1, .5))
```

LoadError: covFun needs to be a function
covFun needs to be a function

Stacktrace:

```
[1] error(s::String)
  @ Base ./error.jl:35
[2] GaussianProcess2(x::Vector{Float64}, covFun::Int64, params::Tuple{Int64, Float64})
  @ Main ./In[4]:6
[3] top-level scope
  @ In[4]:12
```

 Types can't be redefined

We can't redefine a struct (hence my use of `GaussianProcess2` above). I think this has to do with the fact that this would break methods that have been specialized to the type.