

# Solving Equations in With Julia-Defined Types

Chris Rackauckas

June 27, 2019

One of the nice things about `DifferentialEquations.jl` is that it is designed with Julia's type system in mind. What this means is, if you have properly defined a `Number` type, you can use this number type in `DifferentialEquations.jl`'s algorithms! [Note that this is restricted to the native algorithms of `OrdinaryDiffEq.jl`. The other solvers such as `ODE.jl`, `Sundials.jl`, and `ODEInterface.jl` are not compatible with some number systems.]

`DifferentialEquations.jl` determines the numbers to use in its solvers via the types that are designated by `tspan` and the initial condition of the problem. It will keep the time values in the same type as `tspan`, and the solution values in the same type as the initial condition. [Note that adaptive timestepping requires that the time type is compatible with `sqrt` and `^` functions. Thus `dt` cannot be `Integer` or numbers like that if adaptive timestepping is chosen].

Let's solve the linear ODE first define an easy way to get `ODEProblems` for the linear ODE:

```
using DifferentialEquations
f = (u,p,t) -> (p*u)
prob_ode_linear = ODEProblem(f,1/2,(0.0,1.0),1.01);
```

First let's solve it using `Float64s`. To do so, we just need to set `u0` to a `Float64` (which is done by the default) and `dt` should be a float as well.

```
prob = prob_ode_linear
sol = solve(prob,Tsit5())
println(sol)
```

```
retcode: Success
Interpolation: specialized 4th order "free" interpolation
t: [0.0, 0.0996426, 0.345703, 0.677692, 1.0]
u: [0.5, 0.552939, 0.708938, 0.99136, 1.3728]
```

Notice that both the times and the solutions were saved as `Float64`. Let's change the time to use rational values. Rationals are not compatible with adaptive time stepping since they do not have an L2 norm (this can be worked around by defining `internallnorm`, but rationals already explode in size!). To account for this, let's turn off adaptivity as well:

```
prob = ODEProblem(f,1/2,(0//1,1//1),101//100);
```

```
sol = solve(prob,RK4(),dt=1//2^(6),adaptive=false)
println(sol)
```

retcode: Success

Interpolation: 3rd order Hermite

t: Rational{Int64}[0//1, 1//64, 1//32, 3//64, 1//16, 5//64, 3//32, 7//64, 1//8, 9//64, 5//32, 11//64, 3//16, 13//64, 7//32, 15//64, 1//4, 17//64, 9//32, 19//64, 5//16, 21//64, 11//32, 23//64, 3//8, 25//64, 13//32, 27//64, 7//16, 29//64, 15//32, 31//64, 1//2, 33//64, 17//32, 35//64, 9//16, 37//64, 19//32, 39//64, 5//8, 41//64, 21//32, 43//64, 11//16, 45//64, 23//32, 47//64, 3//4, 49//64, 25//32, 51//64, 13//16, 53//64, 27//32, 55//64, 7//8, 57//64, 29//32, 59//64, 15//16, 61//64, 31//32, 63//64, 1//1]

u: [0.5, 0.507953, 0.516033, 0.524241, 0.53258, 0.541051, 0.549658, 0.558401, 0.567283, 0.576306, 0.585473, 0.594786, 0.604247, 0.613858, 0.623623, 0.633542, 0.64362, 0.653857, 0.664258, 0.674824, 0.685558, 0.696463, 0.707541, 0.718795, 0.730229, 0.741844, 0.753644, 0.765632, 0.777811, 0.790183, 0.802752, 0.815521, 0.828493, 0.841671, 0.855059, 0.86866, 0.882477, 0.896514, 0.910775, 0.925262, 0.93998, 0.954931, 0.970121, 0.985552, 1.00123, 1.01715, 1.03333, 1.04977, 1.06647, 1.08343, 1.10067, 1.11817, 1.13596, 1.15403, 1.17239, 1.19103, 1.20998, 1.22923, 1.24878, 1.26864, 1.28882, 1.30932, 1.33015, 1.35131, 1.3728]

Now let's do something fun. Let's change the solution to use Rational{BigInt} and print out the value at the end of the simulation. To do so, simply change the definition of the initial condition.

```
prob = ODEProblem(f,BigInt(1)//BigInt(2),(0//1,1//1),101//100);
sol =solve(prob,RK4(),dt=1//2^(6),adaptive=false)
println(sol[end])
```

```
415403291938655888343294424838034348376204408921988582429386196369066828013
380062427154556444246064110042147806995712770513313913105317131993928991562
472219540324173687134074558951938783349315387199475055050716642476760417033
833225395963069751630544424879625010648869655282442577465289103178163815663
464066572670655356269579471636764679863656649012559514171272038086748586891
653145664881452891757769341753396504927956887980186316721217138912802907978
839488971277351483679854338427632656105429434285170828205087679096886906512
836058415177000071451519455149761416134211934766818795085616643778333812510
724294609438512646808081849075509246961483574876752196687093709017376892988
720208689912813268920171256693582145356856885176190731036088900945481923320
301926151164642204512204346142796306783141982263276125756548530824427611816
333393407861066935488564588880674178922907680658650707284447124975289884078
283531881659241492248450685643985785207092880524994430296917090030308304496
2139908567605824428891872081720287044135359380045755621121//302595526357001
916401850227786985339805854374596312639728370747077589271270423243703004392
074003302619884721642626495128918849830763359112247111187416392615737498981
461087857422550657171300852094084580555857942985570738231419687525783564788
285621871741725085612510228468354691202070954415518824737971685957295081128
193794470230767667945336581432859330595785427486755359414346047520148998708
472579747503225700773992946775819105236957926068135290787592745892648489231
54827578713239056475245050253159810279037690534441254912000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
```



```
using DecFP
prob_ode_decfpllinear =
    ODEProblem(f,Dec128(1)/Dec128(2),(Dec128(0.0),Dec128(1.0)),Dec128(1.01))
sol =solve(prob_ode_decfpllinear,Tsit5())
```

Error: StackOverflowError:

```
println(sol[end]); println(typeof(sol[end]))
```

```
1.372800440903808727789283182314
ArbNumerics.ArbFloat{128}
```

**Decimals.jl** Install with ]add Decimals.

```
using Decimals
prob_ode_decimallinear =
    ODEProblem(f,[decimal("1.0")]./[decimal("2.0")],(0//1,1//1),decimal(1.01))
sol =solve(prob_ode_decimallinear,RK4(),dt=1/2^(6)) #Fails
```

Error: MethodError: Decimals.Decimal(::Rational{Int64}) is ambiguous. Candidates:

```
(::Type{T})(x::Rational{S}) where {S, T<:AbstractFloat} in Base at rational.jl:92
```

```
Decimals.Decimal(num::Real) in Decimals at /home/alex/.julia/packages/Decimals/Qfcas/src/decimal.jl:13
```

Possible fix, define

```
Decimals.Decimal(::Rational{S})
```

```
println(sol[end]); println(typeof(sol[end]))
```

```
1.372800440903808727789283182314
ArbNumerics.ArbFloat{128}
```

At the time of writing this, Decimals are not compatible. This is not on DifferentialEquations.jl's end, it's on partly on Decimal's end since it is not a subtype of Number. Thus it's not recommended you use Decimals with DifferentialEquations.jl

### 0.3 Conclusion

As you can see, DifferentialEquations.jl can use arbitrary Julia-defined number systems in its arithmetic. If you need 128-bit floats, i.e. a bit more precision but not arbitrary, DoubleFloats.jl is a very good choice! For arbitrary precision, ArbNumerics are the most feature-complete and give great performance compared to BigFloats, and thus I recommend their

use when high-precision (less than 512-800 bits) is required. DecFP is a great library for high-performance decimal numbers and works well as well. Other number systems could use some modernization.

## 0.4 Appendix

This tutorial is part of the DiffEqTutorials.jl repository, found at: <https://github.com/JuliaDiffEq/DiffEqTutorials>

To locally run this tutorial, do the following commands:

```
using DiffEqTutorials
DiffEqTutorials.weave_file("type_handling", "01-number_types.jmd")
```

Computer Information:

```
Julia Version 1.1.1
Commit 55e36cc308 (2019-05-16 04:10 UTC)
Platform Info:
  OS: Linux (x86_64-pc-linux-gnu)
  CPU: Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-6.0.1 (ORCJIT, ivybridge)
```

Package Information:

```
Status `~/julia/environments/v1.1/Project.toml`
[7e558dbc-694d-5a72-987c-6f4ebed21442] ArbNumerics 0.5.4
[6e4b80f9-dd63-53aa-95a3-0cdb28fa8baf] BenchmarkTools 0.4.2
[be33ccc6-a3ff-5ff2-a52e-74243cff1e17] CUDAnative 2.2.0
[3a865a2d-5b23-5a0f-bc46-62713ec82fae] CuArrays 1.0.2
[55939f99-70c6-5e9b-8bb0-5071ed7d61fd] DecFP 0.4.8
[abce61dc-4473-55a0-ba07-351d65e31d42] Decimals 0.4.0
[ebbdde9d-f333-5424-9be2-dbf1e9acfb5e] DiffEqBayes 1.1.0
[eb300fae-53e8-50a0-950c-e21f52c2b7e0] DiffEqBiological 3.8.2
[459566f4-90b8-5000-8ac3-15dfb0a30def] DiffEqCallbacks 2.5.2
[f3b72e0c-5b89-59e1-b016-84e28bfd966d] DiffEqDevTools 2.9.0
[1130ab10-4a5a-5621-a13d-e4788d82bd4c] DiffEqParamEstim 1.6.0
[055956cb-9e8b-5191-98cc-73ae4a59e68a] DiffEqPhysics 3.1.0
[6d1b261a-3be8-11e9-3f2f-0b112a9a8436] DiffEqTutorials 0.1.0
[0c46a032-eb83-5123-abaf-570d42b7fbaa] DifferentialEquations 6.4.0
[31c24e10-a181-5473-b8eb-7969acd0382f] Distributions 0.20.0
[497a8b3b-efae-58df-a0af-a86822472b78] DoubleFloats 0.9.1
[f6369f11-7733-5829-9624-2563aa707210] ForwardDiff 0.10.3
[c91e804a-d5a3-530f-b6f0-dfbca275c004] Gadfly 1.0.1
[7073ff75-c697-5162-941a-fcdaad2a7d2a] IJulia 1.18.1
[4138dd39-2aa7-5051-a626-17a0bb65d9c8] JLD 0.9.1
```

[23fbe1c1-3f47-55db-b15f-69d7ec21a316] Latexify 0.8.2  
[eff96d63-e80a-5855-80a2-b1b0885c5ab7] Measurements 2.0.0  
[961ee093-0014-501f-94e3-6117800e7a78] ModelingToolkit 0.2.0  
[76087f3c-5699-56af-9a33-bf431cd00edd] NLOpt 0.5.1  
[2774e3e8-f4cf-5e23-947b-6d7e65073b56] NLSolve 4.0.0  
[429524aa-4258-5aef-a3af-852621145aeb] Optim 0.18.1  
[1dea7af3-3e70-54e6-95c3-0bf5283fa5ed] OrdinaryDiffEq 5.8.1  
[65888b18-ceab-5e60-b2b9-181511a3b968] ParameterizedFunctions 4.1.1  
[91a5bcdd-55d7-5caf-9e0b-520d859cae80] Plots 0.25.1  
[d330b81b-6aea-500a-939a-2ce795aea3ee] PyPlot 2.8.1  
[731186ca-8d62-57ce-b412-fbd966d074cd] RecursiveArrayTools 0.20.0  
[90137ffa-7385-5640-81b9-e52037218182] StaticArrays 0.11.0  
[f3b207a7-027a-5e70-b257-86293d7955fd] StatsPlots 0.11.0  
[c3572dad-4567-51f8-b174-8c6c989267f4] Sundials 3.6.1  
[1986cc42-f94f-5a68-af5c-568840ba703d] Unitful 0.15.0  
[44d3d7a6-8a23-5bf8-98c5-b353f8df5ec9] Weave 0.9.0  
[b77e0a4c-d291-57a0-90e8-8db25a27a240] InteractiveUtils  
[37e2e46d-f89d-539d-b4ee-838fcccc9c8e] LinearAlgebra  
[44cfe95a-1eb2-52ea-b672-e2afdf69b78f] Pkg