Native Julia Solvers for Ordinary Differential Equations Boundary Value Problem: A GSoC proposal

Yingbo Ma

April 1, 2017

Contents

1 Synopsis

2 The Project

2.1 Basic concepts for nonexperts

An ordinary differential equation (ODE) is an equality relationship between a function $y(x)$ and its derivatives, and an nth order ODE can be written as

$$F(x, y, y', \cdots, y^{(n)}) = 0$$

.

Physics and engineering often present ODE problems. Many physics phenomena in physics can be reduced into an ODE. Newton's second law of motion, namely $F = ma$ can be rewritten into an ODE as $m\frac{d^2x}{dt^2} = F(x)$, since a can depend on time. ODEs can be very difficult to solve. There are many case in which an ODE does not have an analytical solution. Therefore, numerical methods need to be used to solve ODEs, by approximating the solution. There are

two kinds of problems in ODE. One is initial value problem (IVP) and the other is boundary value problem (BVP). For instance, the ODE

$$m\frac{d^2x}{dt^2} = -kx(t)$$

describes the motion for a harmonic oscillator. If the initial position x_0 and initial velocity $\frac{dx_0}{dt}$ is known, then it is an IVP problem. If the condition at the "boundary" is know, for instance, initial position x_0 and final position x_1, then it is a BVP problem. The solvers that I am going to work on with solvers for BVP problem.

2.1.1 Introduction

The project that I propose to work on in Google's Summer of Code project is the native Julia implementation of some BVP solving methods for ODE, namely, collocation method and shooting method.

2.2 Project Goals

2.2.1 Goal 1: Implement BVP related data structure

A data structure to describe the BVP problem, namely, "BVProblem". It contains the information

$$F(x, y, y', \cdots, y^{(n)}) = 0$$
domin: $x \in [a, b]$
boundary condition: Dirichlet, Neumann, Robin.

It can be defined by

$$\text{prob} = \text{BVProblem}(f, domin, bc)$$

.

```
abstract AbstractBVProblem{dType,bType,isinplace,F} <: DEProblem

type BVProblem{dType,bType,initType,F} <: AbstractBVProblem{dType,bType,F}
  f::F
  domin::dType
  bc::bType
  init::initType
end

function BVProblem(f,domin,bc,init=nothing)
  BVProblem{eltype(domin),eltype(bc),eltype(init),typeof(f)}(f,domin,bc,init)
end
```

2.2.2 Goal 2: Implement shooting method

The shooting method is a method that convert a BVP problem into an IVP problem and a root finding problem. Generally, the shooting method is efficient in simple problems, because it does not need a discretization matrix. This is no memory overhead. The drawback is that even if the BVP problem is well-conditioned, the root finding problem that the BVP converted to can be ill-conditioned. Therefore, a more robust method like the collocation method is also need, despite shooting method is easy to implement.

```
function solve(prob::BVProblem; OptSolver=LBFGS())
  bc = prob.bc
  u0 = bc[1]
  len = length(bc[1])
  probIt = ODEProblem(prob.f, u0, prob.domin)
  function loss(minimizer)
    probIt.u0 = minimizer
    sol = DifferentialEquations.solve(probIt)
    norm(sol[end]−bc[2])
  end
  opt = optimize(loss, u0, OptSolver)
  probIt.u0 = opt.minimizer
  @show opt.minimum
  DifferentialEquations.solve(probIt)
end
```

2.2.3 Goal 3: Implement collocation method

2.3 Stretch Goals and Future Directions

2.4 Timeline

2.5 Potential Hurdles

2.6 Mentor

2.7 Julia Coding Demo

2.8 About me

2.9 Contact Information

3 Summer Logistics

References