# Native Julia solvers for ordinary differential equations and improvements to IVP testing suite: A GSoC proposal

Joseph Obiajulu

March 2016

## 1 Synopsis

ODE.jl is an ever increasing store house of numerical solvers of ordinary differential equations. While many solvers are currently found in ODE.jl, there are still many which are awaiting a native Julia implementation. In particular, there is a lack of implicit solvers implemented in ODE.jl (currently, there is only one implicit solver: `ode23s`). Such solvers are important because they are especially suited for solving stiff ODEs, which are very common. In light of this, the first goal of my proposed project is a native implementation of implicit solvers RADAU and MEBDFI. Further, in order to reliably use ODE.jl, a robust initial value problem (IVP) testing suite for ODE.jl solvers is a necessity. Work has has been started towards this end through the development of IVPTestSuite.jl. Though, there are many improvements to IVPTestSuite.jl which could be made, especially in improving the ODE solver performance tracking. Thus, the second goal of my proposed project is to advance IVPTestSuite.jl and its surrounding documentation.

## 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 $n$th order ODE can be written as

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

in its general form. They arise and find applications in mathematics, especially Analysis, and virtually all the hard sciences, especially in engineering and physics. In fact, many engineering and physic problems simple reduce to solving a ODE; for example, finding the oscillatory motion of a spring with spring constant $k$ reduces to solving Newton's second law $F = ma = m\frac{d^2x(t)}{dt^2} = -kx(t)$, an ODE. Numerical computing methods, which are able to arrive a approximate solutions to complicated systems of ordinary equations are an indispensable tool for engineers and scientists. One must give partial credit to these numerical methods for the advancements in science and engineering. In light of their importance to modern science, there is always a demand for ever better ODE solvers (where better can mean a number of things: more accurate, quicker, less memory, etc), or specific ODE solvers which better suite specific ODE problems. Over the years, certain numerical solving methods have risen to fame due to their good performance, and have been implemented in a number of languages. However, many very good solvers are still awaiting a native Julia implementation in ODE.jl, Julia's ODE package.

### 2.2 Introduction

The project I propose to complete through Google's Summer of Code program this summer is the native implementation of some well know solvers, namely, RADAU (an implicit Runge-Kutta solver), MEBDFI (a Modified BDF solver), as well a Adams method ODE solver as a warm-up solver. We will discuss specific benefits and attractions of each of these solvers in the "Project Goals" sections as to explain why we choose

these particular solvers, though it should be noted here that these solvers have been discussed and requested by various members of the Julia community and seem to be sincerely in need.

Also frequently requested by the Julia community is an robust Julia initial value problem testing suite, which tests a solver's performance using well-known IVP test problems. @mauro3 has made considerable progress in this endeavor with the creation of IVPTestingSuite.jl. However, development has recently slowed, mostly, I think, do to a lack of developers who can focus on the package. I would hope to help contribute and keep the project active, and to that end, the second large goal of the proposed project is to improve the existing IVP testing suite. The specific proposed improvements are detailed below, but they focus on making IVPTestSuite.jl more user-friendly and increasing the perform tracking functionality. Also, in my mind, a state of the art IVP testing suite is especially necessary to compete with other commercial ODE packages, which is another motivation to undergo this project.

And thus, paraphasing @mauro3, a natural progression of this proposed project would be to *Add new solvers to ODE.jl and improve IVPTestSuite.jl to validate their performance.*

Project deliverables are detailed below, but, in a nutshell, they are new solvers implemented in ODE.jl, enhancements to IVPTestSuite.jl, and new documentation relevant to both ODE.jl and IVPTestSuite. A timeline is presented with when I anticipate these deliverables will be finished and awaiting peer-review and merging.

## 2.3 Project Goals

### 2.3.1 Goal 1: Implement Adam-Bashforth methods

**What does it do and how does it work?** Adam-Bashforth (AB) methods are multistep explicit methods for solving first order ODEs. To show this solver works in general, we explain the theory behind the simplest AB method, the 2nd-order AB method (also called 2-step AB). Consider the ODE

$$y'(t) = f(t, y(t))$$

Our first step is to integrate from $t_{n+1}$ to $t_{n+2}$ to see that

$$y(t_{n+2}) - y(t_{n+1}) = \int_{t_{n+1}}^{t_{n+2}} f(\tau, y(\tau))d\tau$$

Next, instead of left or right endpoint rules, or the trapezoidal rule, we will use a Lagrange interpolating polynomial, $p(\tau)$, to approximate our function on $[t_{n+1}, t_{n+2}]$. The interpolating polynomial formula gives

$$p(\tau) = \frac{\tau - t_{n+1}}{t_n - t_{n+1}} f(t_n, y(t_n)) + \frac{\tau - t_n}{t_{n+1} - t_n} f(t_{n+1}, y(t_{n+1})) = \frac{t_{n+1} - \tau}{h} f(t_n, y(t_n)) + \frac{\tau - t_n}{h} f(t_{n+1}, y(t_{n+1}))$$

where $h = t_{n+1} - t_n$. Approximating our integral, we see

$$y(t_{n+2}) - y(t_{n+1}) = \int_{t_{n+1}}^{t_{n+2}} f(\tau, y(\tau))d\tau \tag{1}$$

$$\approx \int_{t_{n+1}}^{t_{n+2}} p(\tau)d\tau = \frac{3h}{2} f(t_{n+1}), y(t_{n+1}) - \frac{h}{2} f(t_n, y(t_n) \tag{2}$$

And thus, the above result leads to the numerical method of solving $y(t_{n+2}) = y(t_{n+1}) + \frac{3h}{2} f(t_{n+1}), y(t_{n+1}) - \frac{h}{2} f(t_n, y(t_n)$. We notice that 2-step AB requires two initial conditions, and in general the $k$-th step Adam-Bashforth method requires $k$ initial conditions. Runge-Kutta is typical used to achieve these necessary added initial condition. Then, the general numerical result is simply obtained from increasing the power of the Langrange interpolating polynomial via the added initial conditions. This is the general idea behind AB solves

**Implementation Details** I would implement AB2 and AB4 and aim to see these methods successfully merged in ODE.jl as a warm up to the harder solvers. I would give the user the option to either manually input additional initial conditions, or get them from Runge-Kutta. I have made a demo of AB2 and AB4, which can be found below in the coding demo section

**Why this solver?** AB2 and AB4 are canonical and classical solvers, which would be great to have included in ODE.jl. Also, it will serve as a warm-up solver for me, so that I can learn how best to integrate a new solver into ODE.jl while working with a fairly simple method.

#### 2.3.2 Goal 2: Implement RADAU methods

**What does it do and how does it work?** RADAU is a Radua IIA implicit Runge-Kutta method. It is written for problems taking the form $My' = f(t, y)$ where $M$ could be a singular matrix. RADAU5,9, and 13 are all $L$-stable, which make them especially suited for solving very stiff equations. How does it work? Well, I still have to do more research into how RADAU was specifically implemented; what we here instead discuss is how Radua IIA, and Runge-Kutta methods work in general. Runge-Kutta methods are used to solve $y' = f(t, y)$, where the solution takes the form

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i$$

where

$$k_i = f\Big(t_n + c_i h, y_n + h\sum_{j=1}^{s} a_{ij} k_j\Big)$$

Different Runge-Kutta methods have different Butcher tableaus, which specifies $c_i, a_{ij}$ and $b_i$ for $1 \leq i, j \leq s$. The tableu for the Radua IIA method specifically for the third-order, for example, is

| | | | | | |
|---|---|---|---|---|---|
| $c_1$ | $a11$ | $a_1 2$ | $1/3$ | $5/12$ | $-1/12$ |
| $c_2$ | $a21$ | $a_2 2$ | $1$ | $3/4$ | $1/4$ |
| | $b_1$ | $b_2$ | | $3/4$ | $1/4$ |

**Implementation Details** Code for RADAU5,9 and 13 was written by E.Hairer and G. Wanner (Universite de Geneve) coded in Fortran and is currently in the public domain, so most of the implementation would be intelligently reproducing this algorithm in Julia, in ways that most take advantage of the differences between the languages. Currently, I plan on implementing all three: RADAU5,9, and 13.
**Why this solver?** RADAU has been specifically requested by name by more than one from the Julia community, as a way to solve very stiff DAEs.

#### 2.3.3 Goal 3: Implement MEBDFI

**What does it do and how does it work?**

Normal backward differentiation formula methods are implicit linear multistep methods used for solving stiff ODEs and DAEs. The specific solver MEBDFI is a BDF-like method which was first implemented by T.J. Abdulla and J.R. Cash (Imperial College) and was built to solve systems of implicit differential algebraic equations

$$F(t, y, y') = 0, y = (y(1), y(2), \cdots, y(n))$$

We first briefly explain how BDF methods work, and then how MEBDFI varies from a normal BDF methods. Backward differentiation formulas approximate $y'(t_n)$ given $y(t_n)$ and other initial conditions. We demonstrate how one arrives at the first-order BDF method. Say we are given the DAE

$$F(y', y, t) = 0$$

Our approach will be to approximate $y'(t_n)$ in terms of $y(t_n)$ and $t_n$. In general, give the IVP

$$y' = f(t, y) \text{ and } y(t_n) = y_n, y(t_{n-1}) = y_{n-1}$$

we create the linear interpolating polynomial

$$y(t) \approx y(t_n) + (t - t_n)\frac{y(t_n) - y(t_{n-1})}{t_n - t_{n-1}}$$

which we take as equality (while acknowledge the existence of some error). Taking derivatives on both sides yields $y'(t) \approx \frac{y(t_n)-y(t_{n-1})}{t_n-t_{n-1}}$ near $t = t_n$. And so, we go back as solve $F(\frac{y(t_n)-y(t_{n-1})}{t_n-t_{n-1}}, y(t_n), t_n) = 0$. We generalize this approach for higher orders simply by using a high-degree interpolating polynomial, and yields the form

$$\sum_{k=0}^{s} a_k y_{n+k} = hbf(t_{n+s}, y_{n+s})$$

, where $h = \frac{t_n-t_0}{n}$ and $a_k, b$ are choose to achieve order $s$. The modified BDF works using iterations of BDF, where in each iteration a nonlinear set of equations is solved. The results of each previous step are used in the next iteration of BDF, for increasing precision. Of course, as a result, MEBDF is much more computationally heavy due to these iterations.
**Implementation Details** Code for MEBDFI written T.J. Abdulla and J.R. Cash (Imperial College) coded in Fortran is currently in the public domain, so most of the implementation would be intelligently reproducing this algorithm in Julia, in ways that most take advantage of the differences between the languages.
**Why this solver?**Once again, the stability of MEBDFI makes it very well suited for solving stiff equations.

#### 2.3.4 Goal 4: Finish Up ODE.jl PR #72

Currently, two Rosenbrock(-W) method solvers, Rodas3 and RA34PWW2, are being implemented by @mauro3 and @jedbrown. Early round tests presented by @mauro3 on PR # 72 show factors of 2 or 5 time increases when compared with the standard `ode23s` solver. So, this solver certainly shows promising results and is worthwhile to finish and merge. I would plan to talk with the developers of the project and see where they paused in development, and what parts of the solver need finishing. Like, RADAU, code for RODAS written by E.Hairer and G. Wanner (Universite de Geneve) coded in Fortran is currently in the public domain.

#### 2.3.5 Goal 5: Expand IVPTestSuite.jl

The importance of a extensive IVP test suite can not be underscored enough. Several members of the Julia community have noted the importance of utilizing and expanding IVPTestSuite.jl (see ODE.jl issues #56 and #91. and IVPTestSuite.jl issues #1 and #2), and their is a sincere need to see further development in this directon.

The project would focus on

- Integrating IVPTestSuite.jl with BenchmarkTrackers.jl in order to produce performance tracking reports. Maybe also more integration with ODE.jl
- Implementing new test-cases
- Implementing new solvers, namely, those I would have coded earlier in GSoC project
- Increasing documentation and reports of performance of different solvers, as to help Julia users decide which ODE solver would best suite their needs. Particularly, I here have in mind replicating something like the Test Set for IVP Solvers maintained by the Universita Degli Studi Di Bari Aldo Moro.

### 2.4 Stretch Goals and Future Directions

The stretch goals for the summer would be to add even more solvers, and I here list a few more

- Implement SDIRK
- Implement Adams- Moulton
- Implement IMEX RK
- Implement PRK (4-th order sympletic solver)

## 2.5 Timeline

**Week 0 (May 23-May 29):** This week is Google's official start week for GSoC. Unfortunately, I will still be in the midst of final exams during this week, so its variable how much time I will have to get things really going. Either (A) I will have a lot of time and essentially would shift the schedule for weeks 1-6 early to weeks 0-4, and take week 6 as time to work on stretch goals or (B) I will pressed for time and will spend the week looking into solving issues on ODE.jl repo and another light work for the GSoC project.

**Week 1 (May 30 - June 5):** I plan to warm up with implementing an explicit ODE solver, which will be easier to code. Will spend the week establishes good work-flows and get used to writing good documentation. I likely will also start work on week 2 goals. Deliverable: Implementation of Adam-Bashforth in ODE.jl

**Week 2 (June 6 - June 12):** The main goal this week is to finish the work started on the RODAS solver in ODE.jl. If I finish early, I would start to make progress on next week's tasks. Deliverable: Merged PR #72

**Week 3 (June 13 - June 19):** I aim to begin work on RADAU solvers. However, this week will be a busy week for me personally. I will be a camp conselor for a leadership training camp. I anticipate being able to spend only $\approx$ 20 hrs working this week, so I don't forsee having a solid deliverable RADAU coded by then, but will anticipate finishing it week 4.

**Week 4 (June 20 - June 26):** Continue working on RADAU solvers. Deliverable: Implementation of RADAU in ODE.jl

**Week 5 (June 27 - July 4):** Hopefully attend JuliaCon, and stay in Boston afterwards for week 6 also. Start Implementation of MEBDFI in ODE.jl

**Week 6 (July 5 - July 11):** The main goal of this week is to finish the implementation of MEBDFI started in week 5. This is also "midterm week". So, I would take stock, write evaluations, and make changes to schedule for following weeks if necessary. I would also spend this week focusing on writing very good documentation for the new solvers, and some of the existing ones as well, hosted on `readthedocs.org`. Deliverable: Implementation of MEBDFI in ODE.jl

**Week 7-8 (July 12 - July 25):** Would switch focus to working on the testing suite, as explained above. I would first focus on linking with BenchmarkTrackers.jl, as I forsee this being the hardest part of this goal. Once that is done, I would focus on adding the newest solvers. After these two subgoals are on steady ground, I would move on to focus on increasing documentation and performance reports. Deliverable: Enhanced IVPTestSuite.jl linked to BenchmarkTrackers.jl for performance tracking and accompanying documentation about each ODE solver and its performance.

**Week 9-10 (July 26 - August 8):** I plan on using this time as Stretch Goals or Buffer Space weeks. Which stretch goals I would work on would be hard to say from here, as some may become most relevant as the summer progresses. As these weeks near, I will run ideas past my mentor and we would determine which I should pursue. Though, if there is one thing I've learned in long-term projects like these, it build in buffer time. So, this could be two weeks of buffer space, just in case some project goals take longer than anticipated.

**Week 11 (August 9 - August 14):** Google's suggested "Pencil's down" week to scrub code, do more testing, and finish documentation. If I am able to do some stretch goal solvers in weeks 9-10, I would spend this week finishing up the corresponding documentation.

## 2.6 Potential Hurdles

The potential hurdles I see are mostly getting acquainted with the ODE.jl coding standards and reading and understanding old solver algorithms that were written in Fortran and which do not have much surrounding documentation. Thus, to anticipate this potential hurdles I am hoping to start getting used to the ODE.jl

package through using it to do some of my side projects between now and May (I am thinking of reproducing results relevant to the Hot Spot Conjecture, which I will be presenting in a seminar class of mine, and am currently planning to write the code in Julia!). Also, I am going to start learning Fortrran (even though a buddy of mine admonished me that Fortran is just needs to die already).

### 2.7 Potential Mentors

Currently @mauro3 has has shown interest in mentoring this project

## 3 Julia Coding Demo: AB2,AB4

See the below link to a Google Drive folder with a .ipynb file, where I demonstrate a native Julia implementation of AB2 and AB4: `https://drive.google.com/folderview?id=0B2oRp-nVYvzGQlBsdEZ6X01TcXc&usp=sharing`

## 4 About me

My name is Joseph Obiajulu and I am currently a junior studying Mathematics and Computer Science at Princeton Univesity. I'm curious, and love tinkering around with new things, opening up blackboxes and discovering the secrets to what make them go. My formal schooling has focused more on Mathematics, and on my side time I try to break more into the world of software development. I'm hoping to join the Julia team for this summer, and I forsee myself continuing to contribute to the community as a hobby moving forward.

### 4.1 Why Julia?

I've always been interested in mathematics and computer science, and have recently started to be fascinated with their intersection in numerical methods. I thought it would be exciting to spend the summer working in the area, to see if I might want to purpose this field in graduate studies. Also, the Julia community seems very passionate about what they do and welcoming to others - exactly the kind of community I was looking to join. Finally, the Julia Language is very much active and growing, as several scientist and engineers are starting to move to Julia as their base language. I thought it would be great to help Julia continue to grow.

### 4.2 Academic and Coding Details

**Academic Details**

- University:Princeton University
- Degree: Mathematics (Predicted: 2017)
- GPA: 3.85
- Relevant Coursework: Analysis I: Fourier Series and PDEs, Validated Numerics, Theory of Algorithms, Algorithms and Data Structures, Advanced Linear Algebra with Applications

**Coding Details** I've been coding for quite sometime now, mostly as part of coursework in my computer science courses. My languages of choice are Java and Python, and have recently been getting into Julia Language (see the Julia Coding Demo above). I'm really hoping to use this summer as a launching-pad to dive deep into the open source world, and to continue to contribute!

### 4.3 Contact Information

**email:** obiajulu@princeton.edu

**github** obiajulu

# 5 Summer Logistics

## 5.1 Work hours

I will anticipate being able to average ∼40 hours of work a week throughout the course of the summer, with some weeks surely being closer to 50 hours and some closer to 30 hours. There is one week (week 3 on the timeline) during which I will be a summer counselor for a leadership camp, and will likely only find the time for closer to 20 hours of work that week. My only other set engagement is a course in philosophy I would be taking from July 12 to August 22. Over all, even with these engagements, I anticipate being able to put 400-500 hours of work into this project this summer.

## 5.2 Would I be willing to relocate to Cambridge for part of GSoC?

I am willing too; I think it would be cool to meet the community I would be working with in person. For example, I think going to Boston for JuliaCon would be pretty cool. Also, I have an older brother who lives in the Cambridge area, and so I would be able to stay in his flat for free.

# 6

# References


[1] Github Repo ODE.jl aand IVPTestSuite.jl, especially those issues and PR threads referenced in proposal

[2] Test Set for IVP Solvers and solver reports, especially on MEBDFI and RADAU methods
`http://www.dm.uniba.it/~testset/testsetivpsolvers`
`http://www.dm.uniba.it/~testset/report/mebdfi.pdf`
`http://www.dm.uniba.it/~testset/report/radau.pdf`

[3] Multistep methods, detailing Adam's Methods
`http://www.math.iit.edu/~fass/478578_Chapter_2.pdf`

[4] Wikipedia's pages on ODE solver methods, especially Runge-Kutta methods. Also, used as a general resource for research
`https://en.wikipedia.org/wiki/List_of_Runge%E2%80%93Kutta_methods`