Epode: Reference Manual

Martin Jay McKee
martinjaymckee@gmail.com

May 12, 2018

Contents

Part I

Just the Basics

Chapter 1

Getting Started

1.1 Overview

Epode is a library for numeric integration of first-order systems of ordinary differential equations for the solution of inital value problems. That is, given a system of equations, an integration range, and an intial value of the form,

$$\dot{\vec{x}} = f(t, \vec{x}), t = [t_0, t_1], \vec{x}_0 = ... \tag{1.1}$$

Epode can calculate an estimate of the final value of the system variables $\vec{x}$. Some problems of this type can be solved analytically. The vast majority of this type are, however, analytically intractible and, as such require some form of numeric solution. This is where an ode solver like **Epode** comes in.

1.1.1 What's with the name anyway?

Epode is a C++14 compatible library for integration of ordinary differential equations (odes). The connection is simple, an ode is a lyric poem written to address a particular subject. This library was written to address the need for a clean, easy to use and efficient library for the solution of ordinary differental equations. The final link (as simply naming the library "ode" would be too boring) is that the word epode is virtually synonomous to ode and refers to a specific form of lyric poem typified by couplets composed of lines with, typically, varying length (not that that matters in any way to this library). The Epode library is an ode to the new features of C++ 11/14 which make such a library much easier to create and use. It doesn't hurt that it's a short name and contains the letters 'o', 'd', 'e', either!

1.2 Installation

1.3 A Boring Example

```
using namespace epode;
using solver_t = epode::integrator::Euler<double, 3>;
using state_t = typename solver_t::state_t;
auto f = [](auto v, auto y){
  return {
    y(0) + v*y(1),
    -y(1),
    y(0) * y(2)
  }
} -> state_t;
auto y0 = state_t{1, 2, 3};
auto solver = Solver(0.01); // Step integration variable by 0.01
solver(f, 0, {0.5, 1, 1.5, 2}, y0);
```

Listing 1: This is how to create and run a solver

Chapter 2

User API

TALK ABOUT THE PARTS OF THE LIBRARY... INTRODUCE THE SECTIONS

2.1 The "solve" Function

The function `solve(...)` is syntactic sugar for a number of steps to develop an Epode solver and run it. Still, for simple uses of **Epode**, it should be fully sufficient.

2.2 Integrator Engine

The integrator engine is the object that contains the implementation of the method "stepping", as well as storage of the integration results.

2.3 Methods

Epode provides a number of integration methods, all the way from the 1^{st} order Euler's method up to the ?? method with adaptive stepping. These methods can be selected explicitly by the user either when using the basic "solve" function or when constructing an integration flow directly.

2.4 Trigger Objects

In the **Epode** library, Several functions of the integration are controlled by - so called - trigger objects.

IF THE "END" VALUE(S) (IS/ARE) BEFORE THE "START" VALUE, CONFIGURE THE TRIGGER OBJECT, LIMITER AND INTEGRATOR TO RUN BACKWARDS IN TIME.

Part II

The Library in Depth

Chapter 3

Solver Methods

3.1 Solver API

3.1.1 End/Save Syntax

There are multiple ways to denote the points where a solver either ends integration or saves the calculated values

3.1.2 Save Transformers

ADD A WAY TO "TRANSFORM" THE STATE BEFORE IT IS OUTPUT. THIS SHOULD ALLOW FOR CHANGING THE SIZE OF THE STATE AS WELL AS DOING SOME NUMBER OF OPERATIONS ON THE STATE BEFORE SAVING IT.

SYNTAX – transform(stats, y_n, $\dot{v}_n$, v_n, y_{n-1})

3.1.3 Results Type

IT WOULD BE NICE TO MAKE IT POSSIBLE FOR THE RESULTS PACKAGE TO BE OF FIXED SIZE (I.E. NOT A STD::VECTOR) SO THAT DYNAMIC ALLOCATIONS CAN BE AVOIDED). THIS WILL REQUIRE DEFINING THE RETURN TYPE OF THE INTEGRATOR OPERATOR () BASED ON THE TYPE OF THE STORE TRIGGER. ONCE THAT WORKS, AS LONG AS THE RETURN TYPES ALL HAVE A SIMILAR API, IT COULD BE A SINGLE VALUE, ARRAY OR VECTOR.

3.2 Explicit Methods

EACH OF THE METHODS SHOULD HAVE A DESCRIPTION OF THE METHOD, THE BUTCHERS TABLEAU, AND ANY REFERENCES. ADDITIONALLY, THE FILE THAT CONTAINS THE IMPLEMENTATION SHOULD BE STATED.

3.2.1 Forward Euler

The Forward Euler method of integration is by far the simplest. It uses only the current state and the derivitive of the state at the current point to extrapolate the state at the next point. While this makes the method simple to implement and easy to understand, it does lead to an integration method which cannot be both accurate and fast. If the timestep is made so small that some level of accuracy is attained, the runtime becomes prohibitive. In any case, the low order of Euler's method makes truly accurate calculations impossible in any case. The butcher's tableau for Forward Euler is:

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

Forward Euler is provided mostly because it can be used as an exceptionally quick to run comparison for a more usable method.

Available with: **include** ⟨*Euler*⟩

3.2.2 RKF1(2)

RKF 1(2) is a first-order accurate, adaptive step-size Runge-Kutta method, introduced by Fehlberg in [1]. The method requires an amortized three function evaluations per step with a butcher's tableau of:

$$\begin{array}{c|ccc} 0 & & & \\ 1/2 & 1/2 & & \\ 1 & 1/256 & 255/256 & \\ \hline & 1/256 & 255/256 & 0 \\ & 1/512 & 255/256 & 1/512 \end{array}$$

Epode implements the adaptive loop by removing the first evaluation from the loop, as it does not depend upon the timestep.

Available with: **include** ⟨*RKF*⟩

3.2.3 Generic 2nd-Order Runge-Kutta

IS THERE A GOOD WAY TO MAKE SUCH A PARAMETRIC METHOD?? THIS COULD USE A VALUE PASSED TO THE METHOD'S CONSTRUCTOR

TODO: WRITE THE GENERIC 2ND ORDER AND THEN IMPLEMENT HEUN'S, EXPLICIT MIDPOINT AND RALSTON'S USING IT.

$$\begin{array}{c|cc} 0 & & \\ \eta & \eta & \\ \hline & 1-1/2\eta & 1/2\eta \end{array}$$

Available with: **include** ⟨*RK2*⟩

3.2.4 Heun

$$
\begin{array}{c|cc}
0 & & \\
1 & 1 & \\
\hline
 & 1/2 & 1/2
\end{array}
$$

Available with: **include** ⟨*RK2*⟩

3.2.5 Midpoint

$$
\begin{array}{c|cc}
0 & & \\
1/2 & 1/2 & \\
\hline
 & 0 & 1
\end{array}
$$

Available with: **include** ⟨*RK2*⟩

3.2.6 Ralston's

$$
\begin{array}{c|cc}
0 & & \\
2/3 & 2/3 & \\
\hline
 & 1/4 & 3/4
\end{array}
$$

Available with: **include** ⟨*RK2*⟩

3.2.7 Euler/Heun

Available with: **include** ⟨*Euler*⟩

3.2.8 RK3

Available with: **include** ⟨*RK3*⟩

3.2.9 Bogacki–Shampine 3(2)

$$
\begin{array}{c|cccc}
0 & & & & \\
1/2 & 1/2 & & & \\
3/4 & 0 & 3/4 & & \\
1 & 2/9 & 1/3 & 4/9 & \\
\hline
 & 2/9 & 1/3 & 4/9 & 0 \\
 & 7/24 & 1/4 & 1/3 & 1/8
\end{array}
$$

Available with: **include** ⟨*BS32*⟩

3.2.10 RK4

Available with: **include** ⟨*RK4*⟩

3.2.11 RKF4(5)

Introduced by Fehlberg in [1], the RKF 4(5) method has been used as the default method[1] in a variety of ODE solvers since it's introduction in the late sixties. The method has an amortiezed cost of six function evaluations per step and an error of order four. The method is described by the butcher's tableau,

$$\begin{array}{c|cccc}
0 & & & & \\
1/2 & 1/2 & & & \\
3/4 & 0 & 3/4 & & \\
1 & 2/9 & 1/3 & 4/9 & \\
\hline
 & 2/9 & 1/3 & 4/9 & 0 \\
 & 7/24 & 1/4 & 1/3 & 1/8
\end{array}$$

Because it does not depend upon the timestep, the first evaluation is removed from the adaptation loop and, therefore, additional adaptation steps require only five function evaluations.

Available with: **include** ⟨*RKF*⟩

3.2.12 Butcher's 5th

Available with: **include** ⟨*Butchers*⟩

[1]In some cases, related $4^{th}/5^{th}$ order methods like the Cash-Karp 4(5) or the Dormand-Prince 4(5) method has been used. In any case, the RKF 4(5) method is important as representing a very good blend of accuracy and speed in problem-space of ode integration.

Chapter 4

The Integrator Process

4.1 Triggers

4.2 Logging

WHILE LOGGING WOULD BE A NICE FEATURE, IT CURRENTLY ISN'T IMPLEMENTED. FIGURE OUT HOW TO DO THAT AND THEN ADD IT

Chapter 5

Extension API

Part III

Usage Examples

Chapter 6

Toy Problems

SHOULD THE TOY PROBLEMS ALL BE SOLVABLE ANALYTICALLY?

6.1 Capacitor Discharge

THIS IS EASY TO SOLVE ANALYTICALLY – USE END VOLTAGE TRIGGER??? (CUSTOM TRIGGER)

6.2 A Random Complex Valued IVP

JUST SOMETHING SIMPLE LIKE $\dot{z} = (z - t)^2$

6.3 Van der Pol Oscillator

THIS CAN SHOW A SIMPLE TWO VARIABLE, MODERATELY STIFF SYSTEM

Chapter 7

Physical Simulation

7.1 Ballistic Modeling

Some of the earliest mechanical and electronic computers were created to assist in the calculation of ballistics. Why not do the same here?

IT WOULD BE NICE TO DO A BALLISTIC MODEL THAT ALLOWS FOR TESTING WITH AND WITHOUT DRAG. IS THERE A WAY TO DO THIS IN THE ODE SOLVER, OR IS THAT PART OF THE SYSTEM DESCRIPTION?

BASE THIS ON [2].

7.2 Pendulum

Here we will solve a non-linear second-order equation which defines a pendulum with friction[1]. The equation,

$$m\ddot{\theta} + \lambda\dot{\theta} + \frac{mg}{L}sin(\theta) = 0 \tag{7.1}$$

defines how the angle of the pendulum, θ, changes with time given the angle (θ), a frictional factor (λ), the pendulum arm length (L), pendulum bob mass (m) and gravitational acceleration (g). We first need to convert this second-order equation to a first-order system so that it can be processed by **Epode**. We can use a change of variables to define the state variables as $y_0 = \theta$ and $y_1 = \dot{\theta}$. By substitution the equation becomes,

$$m\dot{y_1} + \lambda y_1 + \frac{mg}{L}sin(y_0) = 0 \tag{7.2}$$

and following a second-order to first order transformation,

$$\dot{y_0} = y_1 \tag{7.3}$$

[1]the `https://nrich.maths.org/content/id/6478/Paul-not so simple pendulum 2.pdf`

$$\dot{y_1} = -\frac{\lambda y_1 + \frac{mg}{L} sin(y_0)}{m} \tag{7.4}$$

This is the system that we will implement.

`https://nrich.maths.org/6478`

Another Option[3]

7.3 Predator/Prey

On the island of ?? there are ??s and ??s.

Chapter 8

Chaotic Attractors

8.1 Lorenz System

8.2 Rossler Attractor

8.3 Chua's Circuit

Chua's circuit was intended to show that a physical electronic circuit could demonstrate chaotic behavior, as documented by Chua himself in [4]. The circuit, like the Lorenz system and the Rossler system, is described by an ODE with three state variables. For this example, we shall use the implementation of the circuit described in [5].

Part IV

Appendicies

Chapter 9

Basics of ODEs

Chapter 10

Analysis of Numeric Methods

Chapter 11

Troubleshooting

LIST POSSIBLE ERRORS WITH THEIR CAUSE AND SOLUTION, ETC.

Bibliography

[1] Erwin Fehlberg. Low-order classical runge-kutta formulas with stepsize control and their application to some heat transfer problems. 1969.

[2] Amanda Wade. Going ballistic: bullet trajectories. *Undergraduate Journal of Mathematical Modeling: One+ Two*, 4(1):5, 2011.

[3] Robert A Nelson and MG Olsson. The pendulum—rich physics from a simple system. *American Journal of Physics*, 54(2):112–121, 1986.

[4] Leon O Chua. *The genesis of Chua's circuit*.

[5] Michael Peter Kennedy. Robust op amp realization of chua's circuit. *Frequenz*, 46(3-4):66–80, 1992.

Index