3.2 Algorithm Options

We can solve an ODE with many algorithms. The four selected libraries each provide many algorithms. We roughly classify available algorithms into four categories based on the type of algorithm they use. They are a family of Adams methods, a family of backward differentiation formula methods (BDF), a family of Runge-Kutta (RK) methods, and a "catch all" category of other methods. The commonality analysis we provide on available algorithms is a starting point. It is an incomplete approximation. Getting a complete commonality analysis will require help from domain experts in ODEs. Although the commonality is incomplete, the team still benefits from the current analysis. Not only can a future student quickly access information on which algorithm is available in each language, but also the analysis reminds us that we can increase the consistency of artifacts by providing one-to-one mapping for each algorithm in the four libraries. For example, if a user explicitly chooses a family of Adams methods as the targeted algorithm, all available libraries should use a family of Adams methods to solve the ODE. Unfortunately, not all libraries provide a family of Adams methods. Table 3.1 shows the availability of a family of algorithms in each library. The full details of each library's algorithm availability are shown in Appendix A.3.

3.3 Output an ODE

In the Drasil framework, we can generate modularized software. This modularized software will contain a controller module, an input module, a calculation module, and an output module. The controller module contains the main function~~, the~~ that starts ~~of~~

Algorithm \ Library	**Scipy-Python**	**ACM-Java**	**ODEINT-C++**	**OSLO-C#**
Family of Adams	• Implicit Adams	• Adams Bashforth • Adams Moulton	• Adams Bashforth Moulton	
Family of BDF	• BDF			• Gear's BDF
Family of RK	• Dormand Prince (4)5 • Dormand Prince 8(5,3)	• Explicit Euler • 2ed order • 4th order • Gill fourth order • 3/8 fourth order • Luther sixth order • Higham and Hall 5(4) • Dormand Prince 5(4) • Dormand Prince 8(5,3)	• Explicit Euler • Implicit Euler • Symplectic Euler • 4th order • Dormand Prince 5 • Fehlberg 78 • Controlled Error Stepper • Dense Output Stepper • Rosenbrock 4 • Symplectic RKN McLachlan 6	• Dormand Prince RK547M
Others		• Gragg Bulirsch Stoer	• Gragg Bulirsch Stoer	

Table 3.1: Algorithms support in external libraries

the software. The input module handles all input parameters and constraints. We manually create a ~~txt~~ text file that contains all input information. The input module will read this file and convert the information to its environment. The calculation module contains all the logic for solving the scientific problem. For example, in Double Pendulum, the calculation module contains all functions ~~of~~ for calculating the numerical solution. Lastly, the output module will output the solution. In all ODE case studies, the output module will write the return of the calculation module as a string in a ~~txt~~ text file.

With each module interacting with others, we would like to study the output of the calculation module in the ODE case studies. Currently, the calculation module will output a finite sequence of real numbers, $\mathbb{R}^{i}$ n. The i n is a natural number which depends on the start time, end time, and time step. We have the following specification for the calculation module:

i is usually used for an specific element of a sequence.

Module Name	**Input**	**Output**
Calculations	$\mathbb{R}^n$	$\mathbb{R}^i$ n

Table 3.2: Specification for Calculations Module Return s a Finite Sequence

superscript

The n is the length of the sequence. The $\mathbb{R}^n$ represents input values for calculating the ODE. In our study case, after running the generated program, it will create a file containing the numerical solution of the ODE from the start time to the end time. The numerical solution is written as a stream of real numbers in a ~~txt~~ text file. A finite sequence of real numbers only captures a partial solution, and we ideally want to capture a complete solution. Therefore, we would like to explore options to output a different type for the calculation module.

Most selected external libraries only provide numerical solutions in the form of a finite sequence of real numbers, $\mathbb{R}^{n}$. The C# OSLO library not only supports outputting a finite sequence of real numbers but also an infinite sequence of real numbers. In C# OSLO library, we can get an infinite numerical solution that contains all possible values of the dependent variable over time ($\mathbb{R}^j$). The j is the length of the sequence, and it is a natural number. The function `Ode.RK547M` returns an endless enumerable sequence of solution points. If we are interested in a partial solution ($\mathbb{R}^i$), we can filter it with parameters such as start time, end time, and time step. Code 3.1 shows the full details of how to solve Example 3.1.1 in the OSLO library.

```csharp
public static List<double> func_y_t(double K_d, double K_p, double r_t, double t_sim, double t_step) {
    List<double> y_t;
    Func<double, Vector, Vector> f = (t, y_t_vec) => {
        return new Vector(y_t_vec[1], -(1.0 + K_d) * y_t_vec[1] + -(20.0 + K_p) * y_t_vec[0] + r_t * K_p);
    };
    Options opts = new Options();
    opts.AbsoluteTolerance = Constants.AbsTol;
    opts.RelativeTolerance = Constants.RelTol;

    Vector initv = new Vector(new double[] {0.0, 0.0});
    IEnumerable<SolPoint> sol = Ode.RK547M(0.0, initv, f, opts);
    IEnumerable<SolPoint> points = sol.SolveFromToStep(0.0, t_sim, t_step);
    y_t = new List<double> {};
    foreach (SolPoint sp in points) {
        y_t.Add(sp.X[0]);
    }

    return y_t;
}
```

Code 3.1: Source code of solving PDController in OSLO

I've changed $\mathbb{R}^j$ to $\mathbb{R}^{\infty}$. $\mathbb{R}^i$ and $\mathbb{R}^j$ are basically the same type, with a different name for the type parameter. If we had values for i & j, they would be different, like $\mathbb{R}^3$ vs $\mathbb{R}^4$. I'm not sure what Jacques will think of the notation $\mathbb{R}^{\infty}$, but it captures your idea.

In Code 3.1, between line 3 and line 4, we encode the ODE of Example 3.1.1 in a `Vector`. Between line 7 and line 8, we set the absolute and relative tolerance in the `Options` class. In line 10, we initialize initial values. Next, in line 11, we use `Ode.RK547M` to get an endless sequence of real numbers, $\mathbb{R}^{\infty}$. In line 12, we use `SolveFromToStep` to get a partial solution ($\mathbb{R}^i$) based on the start time, the final time, and the time step. Last, between line 13 and line 15, we run a loop to collect the process variable x_1. With the workflow we described above, the `Ode.RK547M(0.0, initv, f, opts)` returns an object with richer data because $\mathbb{R}^i \subset \mathbb{R}^{\infty}$. Instead of returning $\mathbb{R}^i$, we can have an option to return the $\mathbb{R}^{\infty}$. Here is the new specification. The implementation of this specification is not complete, but

Module Name	**Input**	**Output**
Calculations	$\mathbb{R}^n$	$\mathbb{R}^{\infty}$

Table 3.3: Specification for Calculations Module Return an Infinite Sequence

we provide an analysis of what options the C# OSLO library offers.

Ideally, the ODE is a function that means giving an independent variable will output dependent variables. Here is another proposed specification:

This is the form that I like.

Module Name	**Input**	**Output**
Calculations	$\mathbb{R}^n$	$\mathbb{R} \to \mathbb{R}^k$

Table 3.4: Specification for Calculations Return a Funtion

For $\mathbb{R} \to \mathbb{R}^k$, the $\mathbb{R}$ is the independent variable. The $\mathbb{R}^k$ is a sequence of dependent variables. For a fourth-order ODE, the $\mathbb{R}^k$ would be $\mathbb{R}^4$. Since Drasil Framework can generate a library, the idea of outputting an ODE as a function can be useful. A program can hook up the interface of the generated library, and the library will provide support for calculating the numerical solution of the ODE. The implementation of this specification is not complete, but it gives future students some inspiration on how to generate a library to solve the ODE in Drasil.