Tutorial-RK_Butcher_Table_Dictionary

May 12, 2023

-1 Explicit Runge Kutta methods and their Butcher tables

-1.1 Authors: Brandon Clark, Zach Etienne, & Gabriel M Steward

-1.2 This tutorial notebook stores known explicit Runge Kutta-like methods as Butcher tables in a Python dictionary format.

Notebook Status: Validated

Validation Notes: This tutorial notebook has been confirmed to be **self-consistent with its corresponding NRPy+ module**, as documented below. In addition, each of these Butcher tables has been verified to yield an RK method to the expected local truncation error in a challenging battery of ODE tests in the RK Butcher Table Validation tutorial notebook.

-1.2.1 NRPy+ Source Code for this module: MoLtimestepping/RK_Butcher_Table_Dictionary.py

-1.3 Introduction:

The family of explicit Runge Kutta-like methods are commonly used when numerically solving ordinary differential equation (ODE) initial value problems of the form

$$y'(t) = f(y,t), \quad y(t_0) = y_0.$$

These methods can be extended to solve time-dependent partial differential equations (PDEs) via the Method of Lines. In the Method of Lines, the above ODE can be generalized to N coupled ODEs, all written as first-order-in-time PDEs of the form

$$\partial_t \mathbf{u}(t,x,y,u_1,u_2,u_3,...) = \mathbf{f}(t,x,y,...,u_1,u_{1,x},...),$$

where $\mathbf{u}$ and $\mathbf{f}$ are vectors. The spatial partial derivatives of components of $\mathbf{u}$, e.g., $u_{1,x}$, may be computed using approximate numerical differentiation, like finite differences.

As any explicit Runge-Kutta method has its own unique local truncation error, can in principle be used to solve time-dependent PDEs using the Method of Lines, and may be stable under different Courant-Friedrichs-Lewy (CFL) conditions, it is useful to have multiple methods at one's disposal. **This module provides a number of such methods.**

More details about the Method of Lines is discussed further in the Tutorial-RK_Butcher_Table_Generating_C_Code module where we generate the C code to implement the Method of Lines, and additional description can be found in the Numerically Solving the Scalar Wave Equation: A Complete C Code NRPy+ tutorial notebook.

0 Table of Contents

This notebook is organized as follows

1 Step 1: Initialize needed Python modules [Back to top]

Let's start by importing all the needed modules from Python:

[1]:
```
# Step 1: Initialize needed Python modules
import sympy as sp  # SymPy: The Python computer algebra package upon which NRPy+ depends
```

2 Step 2: The Family of Explicit Runge-Kutta-Like Schemes (Butcher Tables) [Back to top]

In general, a predictor-corrector method performs an estimate timestep from n to $n+1$, using e.g., a Runge Kutta method, to get a prediction of the solution at timestep $n+1$. This is the "predictor" step. Then it uses this prediction to perform another, "corrector" step, designed to increase the accuracy of the solution.

Let us focus on the ordinary differential equation (ODE)

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

which acts as an analogue for a generic PDE $\partial_t u(t,x,y,...) = f(t,x,y,...,u,u_x,...)$.

The general family of Runge Kutta "explicit" timestepping methods are implemented using the following scheme:

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

where

$$k_1 = \Delta t f(y_n, t_n) \tag{1}$$
$$k_2 = \Delta t f(y_n + [a_{21}k_1], t_n + c_2\Delta t) \tag{2}$$
$$k_3 = \Delta t f(y_n + [a_{31}k_1 + a_{32}k_2], t_n + c_3\Delta t) \tag{3}$$
$$\vdots \tag{4}$$
$$k_s = \Delta t f(y_n + [a_{s1}k_1 + a_{s2}k_2 + \cdots + a_{s,s-1}k_{s-1}], t_n + c_s\Delta t). \tag{5}$$

Note s is the number of right-hand side evaluations necessary for any given method, i.e., for RK2 $s=2$ and for RK4 $s=4$, and for RK6 $s=7$. These schemes are often written in the form of a so-called "Butcher tableau" or "Butcher table."

$$\begin{array}{c|ccccc} 0 & & & & & \\ c_2 & a_{21} & & & & \\ c_3 & a_{31} & a_{32} & & & \\ \vdots & \vdots & & \ddots & & \\ c_s & a_{s_1} & a_{s2} & \cdots & a_{s,s-1} & \\ \hline & b_1 & b_2 & \cdots & b_{s-1} & b_s \end{array}$$

As an example, the "classic" fourth-order Runge Kutta (RK4) method obtains the solution $y(t)$ to the single-variable ODE $y'(t) = f(y(t),t)$ at time t_{n+1} from t_n via:

$$k_1 = \Delta t f(y_n, t_n), \tag{6}$$
$$k_2 = \Delta t f(y_n + \frac{1}{2}k_1, t_n + \frac{\Delta t}{2}), \tag{7}$$
$$k_3 = \Delta t f(y_n + \frac{1}{2}k_2, t_n + \frac{\Delta t}{2}), \tag{8}$$
$$k_4 = \Delta t f(y_n + k_3, t_n + \Delta t), \tag{9}$$
$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + \mathcal{O}\big((\Delta t)^5\big). \tag{10}$$

Its corresponding Butcher table is constructed as follows:

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	1/3	1/3	1/6

This is one example of many explicit Runge Kutta methods. Throughout the following sections we will highlight different Runge Kutta schemes and their Butcher tables from the first-order Euler's method up to and including an eighth-order method.

2.1 Step 2.a: Generating a Dictionary of Butcher Tables for Explicit Runge Kutta Techniques [Back to top]

We can store all of the Butcher tables in Python's **Dictionary** format using the curly brackets {} and 'key':value pairs. The 'key' will be the *name* of the Runge Kutta method and the value will be the Butcher table itself stored as a list of lists. The convergence order for each Runge Kutta method is also stored. We will construct the dictionary `Butcher_dict` one Butcher table at a time in the following sections.

[2]:
```
# Step 2a: Generating a Dictionary of Butcher Tables for Explicit Runge Kutta Techniques

# Initialize the dictionary Butcher_dict
Butcher_dict = {}
```

2.1.1 Step 2.a.i: Euler's Method [Back to top]

Forward Euler's method is a first order Runge Kutta method. Euler's method obtains the solution $y(t)$ at time t_{n+1} from t_n via:

$$y_{n+1} = y_n + \Delta t f(y_n, t_n)$$

with the trivial corresponding Butcher table

0	
	1

.

[3]:
```
# Step 2.a.i: Euler's Method

Butcher_dict['Euler'] = (
[[sp.sympify(0)],
["", sp.sympify(1)]]
, 1)
```

2.1.2 Step 2.a.ii: RK2 Heun's Method [Back to top]

Heun's method is a second-order RK method that obtains the solution $y(t)$ at time t_{n+1} from t_n via:

$$k_1 = \Delta t f(y_n, t_n), \tag{11}$$
$$k_2 = \Delta t f(y_n + k_1, t_n + \Delta t), \tag{12}$$
$$y_{n+1} = y_n + \frac{1}{2}(k_1 + k_2) + \mathcal{O}\big((\Delta t)^3\big) \tag{13}$$

with corresponding Butcher table

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

[4]:
```
# Step 2.a.ii: RK2 Heun's Method

Butcher_dict['RK2 Heun'] = (
[[sp.sympify(0)],
[sp.sympify(1), sp.sympify(1)],
["", sp.Rational(1,2), sp.Rational(1,2)]]
, 2)
```

2.1.3 Step 2.a.iii: RK2 Midpoint Method [Back to top]

Midpoint method is a second-order RK method that obtains the solution $y(t)$ at time t_{n+1} from t_n via:

$$k_1 = \Delta t f(y_n, t_n), \tag{14}$$
$$k_2 = \Delta t f(y_n + \frac{2}{3}k_1, t_n + \frac{2}{3}\Delta t), \tag{15}$$
$$y_{n+1} = y_n + \frac{1}{2}k_2 + \mathcal{O}\big((\Delta t)^3\big) \tag{16}$$

with corresponding Butcher table

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

[5]:
```
# Step 2.a.iii: RK2 Midpoint (MP) Method

Butcher_dict['RK2 MP'] = (
[[sp.sympify(0)],
[sp.Rational(1,2), sp.Rational(1,2)],
["", sp.sympify(0), sp.sympify(1)]]
, 2)
```

2.1.4 Step 2.a.iv: RK2 Ralston's Method [Back to top]

Ralston's method (see Ralston (1962), is a second-order RK method that obtains the solution $y(t)$ at time t_{n+1} from t_n via:

$$k_1 = \Delta t f(y_n, t_n), \tag{17}$$

$$k_2 = \Delta t f(y_n + \frac{1}{2}k_1, t_n + \frac{1}{2}\Delta t), \tag{18}$$

$$y_{n+1} = y_n + \frac{1}{4}k_1 + \frac{3}{4}k_2 + \mathcal{O}\big((\Delta t)^3\big) \tag{19}$$

with corresponding Butcher table

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

[6]:
```
# Step 2.a.iv: RK2 Ralston's Method

Butcher_dict['RK2 Ralston'] = (
[[sp.sympify(0)],
[sp.Rational(2,3), sp.Rational(2,3)],
["", sp.Rational(1,4), sp.Rational(3,4)]]
, 2)
```

2.1.5 Step 2.a.v: Kutta's Third-order Method [Back to top]

Kutta's third-order method obtains the solution $y(t)$ at time t_{n+1} from t_n via:

$$k_1 = \Delta t f(y_n, t_n), \tag{20}$$

$$k_2 = \Delta t f(y_n + \frac{1}{2}k_1, t_n + \frac{1}{2}\Delta t), \tag{21}$$

$$k_3 = \Delta t f(y_n - k_1 + 2k_2, t_n + \Delta t) \tag{22}$$

$$y_{n+1} = y_n + \frac{1}{6}k_1 + \frac{2}{3}k_2 + \frac{1}{6}k_3 + \mathcal{O}\big((\Delta t)^4\big) \tag{23}$$

with corresponding Butcher table

$$\begin{array}{c|ccc} 0 & & & \\ 1/2 & 1/2 & & \\ 1 & -1 & 2 & \\ \hline & 1/6 & 2/3 & 1/6 \end{array}.$$

[7]:
```
# Step 2.a.v: Kutta's  Third-order Method

Butcher_dict['RK3'] = (
[[sp.sympify(0)],
[sp.Rational(1,2), sp.Rational(1,2)],
[sp.sympify(1), sp.sympify(-1), sp.sympify(2)],
["", sp.Rational(1,6), sp.Rational(2,3), sp.Rational(1,6)]]
, 3)
```

2.1.6 Step 2.a.vi: RK3 Heun's Method [Back to top]

Heun's third-order method obtains the solution $y(t)$ at time t_{n+1} from t_n via:

$$k_1 = \Delta t f(y_n, t_n), \tag{24}$$

$$k_2 = \Delta t f(y_n + \frac{1}{3}k_1, t_n + \frac{1}{3}\Delta t), \tag{25}$$

$$k_3 = \Delta t f(y_n + \frac{2}{3}k_2, t_n + \frac{2}{3}\Delta t) \tag{26}$$

$$y_{n+1} = y_n + \frac{1}{4}k_1 + \frac{3}{4}k_3 + \mathcal{O}\big((\Delta t)^4\big) \tag{27}$$

with corresponding Butcher table

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

[8]:
```
# Step 2.a.vi: RK3 Heun's Method

Butcher_dict['RK3 Heun'] = (
[[sp.sympify(0)],
[sp.Rational(1,3), sp.Rational(1,3)],
[sp.Rational(2,3), sp.sympify(0), sp.Rational(2,3)],
["", sp.Rational(1,4), sp.sympify(0), sp.Rational(3,4)]]
, 3)
```

2.1.7 Step 2.a.vii: RK3 Ralton's Method [Back to top]

Ralston's third-order method (see Ralston (1962), obtains the solution $y(t)$ at time t_{n+1} from t_n via:

$$k_1 = \Delta t f(y_n, t_n), \tag{28}$$
$$k_2 = \Delta t f(y_n + \frac{1}{2}k_1, t_n + \frac{1}{2}\Delta t), \tag{29}$$
$$k_3 = \Delta t f(y_n + \frac{3}{4}k_2, t_n + \frac{3}{4}\Delta t) \tag{30}$$
$$y_{n+1} = y_n + \frac{2}{9}k_1 + \frac{1}{3}k_2 + \frac{4}{9}k_3 + \mathcal{O}\big((\Delta t)^4\big) \tag{31}$$

with corresponding Butcher table

0			
1/2	1/2		
3/4	0	3/4	
	2/9	1/3	4/9

.

[9]:
```
# Step 2.a.vii: RK3 Ralton's Method

Butcher_dict['RK3 Ralston'] = (
[[0],
[sp.Rational(1,2), sp.Rational(1,2)],
[sp.Rational(3,4), sp.sympify(0), sp.Rational(3,4)],
["", sp.Rational(2,9), sp.Rational(1,3), sp.Rational(4,9)]]
, 3)
```

2.1.8 Step 2.a.viii: Strong Stability Preserving Runge-Kutta (SSPRK3) Method [Back to top]

The Strong Stability Preserving Runge-Kutta (SSPRK3) method obtains the solution $y(t)$ at time t_{n+1} from t_n via:

$$k_1 = \Delta t f(y_n, t_n), \tag{32}$$
$$k_2 = \Delta t f(y_n + k_1, t_n + \Delta t), \tag{33}$$
$$k_3 = \Delta t f(y_n + \frac{1}{4}k_1 + \frac{1}{4}k_2, t_n + \frac{1}{2}\Delta t) \tag{34}$$
$$y_{n+1} = y_n + \frac{1}{6}k_1 + \frac{1}{6}k_2 + \frac{2}{3}k_3 + \mathcal{O}\big((\Delta t)^4\big) \tag{35}$$

with corresponding Butcher table

$$\begin{array}{c|ccc} 0 & & & \\ 1 & 1 & & \\ 1/2 & 1/4 & 1/4 & \\ \hline & 1/6 & 1/6 & 2/3 \end{array}.$$

[10]:
```
# Step 2.a.viii: Strong Stability Preserving Runge-Kutta (SSPRK3) Method

Butcher_dict['SSPRK3'] = (
[[0],
[sp.sympify(1), sp.sympify(1)],
[sp.Rational(1,2), sp.Rational(1,4), sp.Rational(1,4)],
["", sp.Rational(1,6), sp.Rational(1,6), sp.Rational(2,3)]]
, 3)
```

2.1.9 Step 2.a.ix: Classic RK4 Method [Back to top]

The classic RK4 method obtains the solution $y(t)$ at time t_{n+1} from t_n via:

$$k_1 = \Delta t f(y_n, t_n), \tag{36}$$
$$k_2 = \Delta t f(y_n + \frac{1}{2}k_1, t_n + \frac{\Delta t}{2}), \tag{37}$$
$$k_3 = \Delta t f(y_n + \frac{1}{2}k_2, t_n + \frac{\Delta t}{2}), \tag{38}$$
$$k_4 = \Delta t f(y_n + k_3, t_n + \Delta t), \tag{39}$$
$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + \mathcal{O}\big((\Delta t)^5\big) \tag{40}$$

with corresponding Butcher table

$$
\begin{array}{c|cccc}
0 & & & & \\
1/2 & 1/2 & & & \\
1/2 & 0 & 1/2 & & \\
1 & 0 & 0 & 1 & \\
\hline
 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}
\,.
$$

[11]:
```
# Step 2.a.vix: Classic RK4 Method

Butcher_dict['RK4'] = (
[[sp.sympify(0)],
[sp.Rational(1,2), sp.Rational(1,2)],
[sp.Rational(1,2), sp.sympify(0), sp.Rational(1,2)],
[sp.sympify(1), sp.sympify(0), sp.sympify(0), sp.sympify(1)],
["", sp.Rational(1,6), sp.Rational(1,3), sp.Rational(1,3), sp.Rational(1,6)]]
, 4)
```

2.1.10 Step 2.a.x: RK5 Dormand-Prince Method [Back to top]

The fifth-order Dormand-Prince (DP) method from the RK5(4) family (see Dormand, J. R.; Prince, P. J. (1980)) Butcher table is:

$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & & \\
\frac{4}{5} & \frac{44}{45} & \frac{-56}{15} & \frac{32}{9} & & & & \\
\frac{8}{9} & \frac{19372}{6561} & \frac{-25360}{2187} & \frac{64448}{6561} & \frac{-212}{729} & & & \\
1 & \frac{9017}{3168} & \frac{-355}{33} & \frac{46732}{5247} & \frac{49}{176} & \frac{-5103}{18656} & & \\
1 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & \frac{-2187}{6784} & \frac{11}{84} & \\
\hline
 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & \frac{-2187}{6784} & \frac{11}{84} & 0
\end{array}
\,.
$$

[12]:
```
# Step 2.a.x:  RK5 Dormand-Prince Method

Butcher_dict['DP5'] = (
[[0],
[sp.Rational(1,5), sp.Rational(1,5)],
[sp.Rational(3,10),sp.Rational(3,40), sp.Rational(9,40)],
[sp.Rational(4,5), sp.Rational(44,45), sp.Rational(-56,15), sp.Rational(32,9)],
[sp.Rational(8,9), sp.Rational(19372,6561), sp.Rational(-25360,2187), sp.Rational(64448,6561), sp.Rational(-212,729)],
[sp.sympify(1), sp.Rational(9017,3168), sp.Rational(-355,33), sp.Rational(46732,5247), sp.Rational(49,176), sp.Rational(-5103,18656)],
[sp.sympify(1), sp.Rational(35,384), sp.sympify(0), sp.Rational(500,1113), sp.Rational(125,192), sp.Rational(-2187,6784), sp.
↪Rational(11,84)],
```

```
["", sp.Rational(35,384), sp.sympify(0), sp.Rational(500,1113), sp.Rational(125,192), sp.Rational(-2187,6784), sp.Rational(11,84), sp.
↪sympify(0)]]
, 5)
```

2.1.11 Step 2.a.xi: RK5 Dormand-Prince Method Alternative [Back to top]

The fifth-order Dormand-Prince (DP) method from the RK6(5) family (see Dormand, J. R.; Prince, P. J. (1981)) Butcher table is:

$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{10} & \frac{1}{10} & & & & & & \\
\frac{2}{9} & \frac{-2}{81} & \frac{20}{81} & & & & & \\
\frac{3}{7} & \frac{615}{1372} & \frac{-270}{343} & \frac{1053}{1372} & & & & \\
\frac{3}{5} & \frac{3243}{5500} & \frac{-54}{55} & \frac{50949}{71500} & \frac{4998}{17875} & & & \\
\frac{4}{5} & \frac{-26492}{37125} & \frac{72}{55} & \frac{2808}{23375} & \frac{-24206}{37125} & \frac{338}{459} & & \\
1 & \frac{5561}{2376} & \frac{-35}{11} & \frac{-24117}{31603} & \frac{899983}{200772} & \frac{-5225}{1836} & \frac{3925}{4056} & \\
 & \frac{821}{10800} & 0 & \frac{19683}{71825} & \frac{175273}{912600} & \frac{395}{3672} & \frac{785}{2704} & \frac{3}{50}
\end{array}
.
$$

[13]:
```
# Step 2.a.xi:  RK5 Dormand-Prince Method Alternative

Butcher_dict['DP5alt'] = (
[[0],
[sp.Rational(1,10), sp.Rational(1,10)],
[sp.Rational(2,9), sp.Rational(-2, 81), sp.Rational(20, 81)],
[sp.Rational(3,7), sp.Rational(615, 1372), sp.Rational(-270, 343), sp.Rational(1053, 1372)],
[sp.Rational(3,5), sp.Rational(3243, 5500), sp.Rational(-54, 55), sp.Rational(50949, 71500), sp.Rational(4998, 17875)],
[sp.Rational(4, 5), sp.Rational(-26492, 37125), sp.Rational(72, 55), sp.Rational(2808, 23375), sp.Rational(-24206, 37125), sp.
↪Rational(338, 459)],
[sp.sympify(1), sp.Rational(5561, 2376), sp.Rational(-35, 11), sp.Rational(-24117, 31603), sp.Rational(899983, 200772), sp.
↪Rational(-5225, 1836), sp.Rational(3925, 4056)],
["", sp.Rational(821, 10800), sp.sympify(0), sp.Rational(19683, 71825), sp.Rational(175273, 912600), sp.Rational(395, 3672), sp.
↪Rational(785, 2704), sp.Rational(3, 50)]]
, 5)
```

2.1.12 Step 2.a.xii: RK5 Cash-Karp Method [Back to top]

The fifth-order Cash-Karp Method (see J. R. Cash, A. H. Karp. (1980)) Butcher table is:

$$
\begin{array}{c|cccccc}
0 & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & \\
\frac{3}{5} & \frac{3}{10} & \frac{-9}{10} & \frac{6}{5} & & & \\
1 & \frac{-11}{54} & \frac{5}{2} & \frac{-70}{27} & \frac{35}{27} & & \\
\frac{7}{8} & \frac{1631}{55296} & \frac{175}{512} & \frac{575}{13824} & \frac{44275}{110592} & \frac{253}{4096} & \\
\hline
 & \frac{37}{378} & 0 & \frac{250}{621} & \frac{125}{594} & 0 & \frac{512}{1771}
\end{array}
.
$$

[14]:
```
# Step 2.a.xii:  RK5 Cash-Karp Method

Butcher_dict['CK5'] = (
[[0],
[sp.Rational(1,5), sp.Rational(1,5)],
[sp.Rational(3,10),sp.Rational(3,40), sp.Rational(9,40)],
[sp.Rational(3,5), sp.Rational(3,10), sp.Rational(-9,10), sp.Rational(6,5)],
[sp.sympify(1), sp.Rational(-11,54), sp.Rational(5,2), sp.Rational(-70,27), sp.Rational(35,27)],
[sp.Rational(7,8), sp.Rational(1631,55296), sp.Rational(175,512), sp.Rational(575,13824), sp.Rational(44275,110592), sp.
 ↪Rational(253,4096)],
["",sp.Rational(37,378), sp.sympify(0), sp.Rational(250,621), sp.Rational(125,594), sp.sympify(0), sp.Rational(512,1771)]]
, 5)
```

2.1.13 Step 2.a.xiii: RK6 Dormand-Prince Method [Back to top]

The sixth-order Dormand-Prince method (see Dormand, J. R.; Prince, P. J. (1981)) Butcher Table is:

$$
\begin{array}{c|cccccccc}
0 & & & & & & & & \\
\frac{1}{10} & \frac{1}{10} & & & & & & & \\
\frac{2}{9} & \frac{-2}{81} & \frac{20}{81} & & & & & & \\
\frac{3}{7} & \frac{615}{1372} & \frac{-270}{343} & \frac{1053}{1372} & & & & & \\
\frac{3}{5} & \frac{3243}{5500} & \frac{-54}{55} & \frac{50949}{71500} & \frac{4998}{17875} & & & & \\
\frac{4}{5} & \frac{-26492}{37125} & \frac{72}{55} & \frac{2808}{23375} & \frac{-24206}{37125} & \frac{338}{459} & & & \\
1 & \frac{5561}{2376} & \frac{-35}{11} & \frac{-24117}{31603} & \frac{899983}{200772} & \frac{-5225}{1836} & \frac{3925}{4056} & & \\
1 & \frac{465467}{266112} & \frac{-2945}{1232} & \frac{-5610201}{14158144} & \frac{10513573}{3212352} & \frac{-424325}{205632} & \frac{376225}{454272} & 0 & \\
\hline
 & \frac{61}{864} & 0 & \frac{98415}{321776} & \frac{16807}{146016} & \frac{1375}{7344} & \frac{1375}{5408} & \frac{-37}{1120} & \frac{1}{10}
\end{array}
.
$$

[15]:
```
# Step 2.a.xiii:  RK6 Dormand-Prince Method

Butcher_dict['DP6'] = (
[[0],
[sp.Rational(1,10), sp.Rational(1,10)],
[sp.Rational(2,9), sp.Rational(-2, 81), sp.Rational(20, 81)],
```

```
[sp.Rational(3,7), sp.Rational(615, 1372), sp.Rational(-270, 343), sp.Rational(1053, 1372)],
[sp.Rational(3,5), sp.Rational(3243, 5500), sp.Rational(-54, 55), sp.Rational(50949, 71500), sp.Rational(4998, 17875)],
[sp.Rational(4, 5), sp.Rational(-26492, 37125), sp.Rational(72, 55), sp.Rational(2808, 23375), sp.Rational(-24206, 37125), sp.
 ↪Rational(338, 459)],
[sp.sympify(1), sp.Rational(5561, 2376), sp.Rational(-35, 11), sp.Rational(-24117, 31603), sp.Rational(899983, 200772), sp.
 ↪Rational(-5225, 1836), sp.Rational(3925, 4056)],
[sp.sympify(1), sp.Rational(465467, 266112), sp.Rational(-2945, 1232), sp.Rational(-5610201, 14158144), sp.Rational(10513573, 3212352),␣
 ↪sp.Rational(-424325, 205632), sp.Rational(376225, 454272), sp.sympify(0)],
["", sp.Rational(61, 864), sp.sympify(0), sp.Rational(98415, 321776), sp.Rational(16807, 146016), sp.Rational(1375, 7344), sp.
 ↪Rational(1375, 5408), sp.Rational(-37, 1120), sp.Rational(1,10)]]
, 6)
```

2.1.14 Step 2.a.xiv: RK6 Luther's Method [Back to top]

Luther's sixth-order method (see H. A. Luther (1968)) Butcher table is:

$$
\begin{array}{c|cccccccc}
0 & & & & & & & & \\
1 & 1 & & & & & & & \\
\frac{1}{2} & \frac{3}{8} & & & & & & & \\
\frac{2}{3} & \frac{8}{27} & \frac{1}{8} & & & & & & \\
\frac{(7-q)}{14} & \frac{(-21+9q)}{392} & \frac{2}{27} & \frac{8}{27} & & & & & \\
\frac{(7+q)}{14} & \frac{(-1155-255q)}{1960} & \frac{(-56+8q)}{392} & \frac{(336-48q)}{392} & \frac{(-63+3q)}{392} & & & & \\
1 & \frac{(330+105q)}{180} & \frac{(-280-40q)}{1960} & \frac{320q}{1960} & \frac{(63+363q)}{1960} & \frac{(2352+392q)}{1960} & & & \\
 & \frac{1}{20} & \frac{2}{3} & \frac{(-200+280q)}{180} & \frac{(126-189q)}{180} & \frac{(-686-126q)}{180} & \frac{(490-70q)}{180} & & \\
 & & 0 & \frac{16}{45} & 0 & \frac{49}{180} & \frac{49}{180} & \frac{1}{20} &
\end{array}
$$

where $q = \sqrt{21}$.

[16]:
```
#  Step 2.a.xiv:  RK6 Luther's Method

q = sp.sqrt(21)
Butcher_dict['L6'] = (
[[0],
[sp.sympify(1), sp.sympify(1)],
[sp.Rational(1,2), sp.Rational(3,8), sp.Rational(1,8)],
[sp.Rational(2,3), sp.Rational(8,27), sp.Rational(2,27), sp.Rational(8,27)],
[(7 - q)/14, (-21 + 9*q)/392, (-56 + 8*q)/392, (336 -48*q)/392, (-63 + 3*q)/392],
[(7 + q)/14, (-1155 - 255*q)/1960, (-280 -  40*q)/1960, (-320*q)/1960, (63 + 363*q)/1960, (2352 + 392*q)/1960],
[sp.sympify(1), ( 330 + 105*q)/180, sp.Rational(2,3), (-200 + 280*q)/180, (126 - 189*q)/180, (-686 - 126*q)/180, (490 -  70*q)/180],
["", sp.Rational(1, 20), sp.sympify(0), sp.Rational(16, 45), sp.sympify(0), sp.Rational(49, 180), sp.Rational(49, 180), sp.Rational(1,␣
 ↪20)]]
, 6)
```

2.1.15 Step 2.a.xv: RK8 Dormand-Prince Method [Back to top]

The eighth-order Dormand-Prince Method (see Dormand, J. R.; Prince, P. J. (1981)) Butcher table is:

0									
$\frac{1}{18}$	$\frac{1}{18}$								
$\frac{1}{12}$	$\frac{1}{48}$	$\frac{1}{16}$							
$\frac{1}{8}$	$\frac{1}{32}$	0	$\frac{3}{32}$						
$\frac{5}{16}$	$\frac{5}{16}$	0	$\frac{-75}{64}$	$\frac{75}{64}$					
$\frac{3}{8}$	$\frac{3}{80}$	0	0	$\frac{3}{16}$	$\frac{3}{20}$				
$\frac{59}{400}$	$\frac{29443841}{614563906}$	0	0	$\frac{77736538}{692538347}$	$\frac{-28693883}{1125000000}$	$\frac{23124283}{1800000000}$			
$\frac{93}{200}$	$\frac{16016141}{946692911}$	0	0	$\frac{61564180}{158732637}$	$\frac{22789713}{633445777}$	$\frac{545815736}{2771057229}$	$\frac{-180193667}{1043307555}$		
$\frac{5490023248}{9719169821}$	$\frac{39632708}{573591083}$	0	0	$\frac{-433636366}{683701615}$	$\frac{-421739975}{2616292301}$	$\frac{100302831}{723423059}$	$\frac{790204164}{839813087}$	$\frac{800635310}{3783071287}$	
$\frac{13}{20}$	$\frac{246121993}{1340847787}$	0	0	$\frac{-37695042795}{15268766246}$	$\frac{-309121744}{1061227803}$	$\frac{-12992083}{490766935}$	$\frac{6005943493}{2108947869}$	$\frac{393006217}{1396673457}$	$\frac{123872331}{1001029789}$
$\frac{1201146811}{1299019798}$	$\frac{-1028468189}{846180014}$	0	0	$\frac{8478235783}{508512852}$	$\frac{1311729495}{1432422823}$	$\frac{-10304129995}{1701304382}$	$\frac{-48777925059}{3047939560}$	$\frac{15336726248}{1032824649}$	$\frac{-45442868181}{3398467696}$
$\frac{3065993473}{597172653}$	$\frac{185892177}{718116043}$	0	0	$\frac{-3185094517}{667107341}$	$\frac{-477755414}{1098053517}$	$\frac{-703635378}{230739211}$	$\frac{5731566787}{1027545527}$	$\frac{5232866602}{850066563}$	$\frac{-4093664535}{808688257}$
1	$\frac{65686358}{487910083}$								
$\frac{3962137247}{1805957418}$	$\frac{403863854}{491063109}$	0	0	$\frac{-5068492393}{434740067}$	$\frac{-411421997}{543043805}$	$\frac{652783627}{914296604}$	$\frac{11173962825}{925320556}$	$\frac{-13158990841}{6184727034}$	$\frac{3936647629}{1978049680}$
1	$\frac{248638103}{1413531060}$	0							
$\frac{-160528059}{685178525}$	$\frac{14005451}{335480064}$	0	0	0	0	$\frac{-59238493}{1068277825}$	$\frac{181606767}{758867731}$	$\frac{561292985}{797845732}$	$\frac{-1041891430}{1371343529}$
$\frac{760417239}{1151165299}$	$\frac{118820643}{751138087}$	$\frac{-528747749}{2220607170}$	$\frac{1}{4}$						

[17]:
```
# Step 2.a.xv:  RK8 Dormand-Prince Method

Butcher_dict['DP8']=(
[[0],
[sp.Rational(1, 18), sp.Rational(1, 18)],
[sp.Rational(1, 12), sp.Rational(1, 48), sp.Rational(1, 16)],
[sp.Rational(1, 8), sp.Rational(1, 32), sp.sympify(0), sp.Rational(3, 32)],
[sp.Rational(5, 16), sp.Rational(5, 16), sp.sympify(0), sp.Rational(-75, 64), sp.Rational(75, 64)],
[sp.Rational(3, 8), sp.Rational(3, 80), sp.sympify(0), sp.sympify(0), sp.Rational(3, 16), sp.Rational(3, 20)],
[sp.Rational(59, 400), sp.Rational(29443841, 614563906), sp.sympify(0), sp.sympify(0), sp.Rational(77736538, 692538347), sp.
 ↪Rational(-28693883, 1125000000), sp.Rational(23124283, 1800000000)],
[sp.Rational(93, 200), sp.Rational(16016141, 946692911), sp.sympify(0), sp.sympify(0), sp.Rational(61564180, 158732637), sp.
 ↪Rational(22789713, 633445777), sp.Rational(545815736, 2771057229), sp.Rational(-180193667, 1043307555)],
[sp.Rational(5490023248, 9719169821), sp.Rational(39632708, 573591083), sp.sympify(0), sp.sympify(0), sp.Rational(-433636366,␣
 ↪683701615), sp.Rational(-421739975, 2616292301), sp.Rational(100302831, 723423059), sp.Rational(790204164, 839813087), sp.
 ↪Rational(800635310, 3783071287)],
```

```
[sp.Rational(13, 20), sp.Rational(246121993, 1340847787), sp.sympify(0), sp.sympify(0), sp.Rational(-37695042795, 15268766246), sp.
↪Rational(-309121744, 1061227803), sp.Rational(-12992083, 490766935), sp.Rational(6005943493, 2108947869), sp.Rational(393006217,␣
↪1396673457), sp.Rational(123872331, 1001029789)],
[sp.Rational(1201146811, 1299019798), sp.Rational(-1028468189, 846180014), sp.sympify(0), sp.sympify(0), sp.Rational(8478235783,␣
↪508512852), sp.Rational(1311729495, 1432422823), sp.Rational(-10304129995, 1701304382), sp.Rational(-48777925059, 3047939560), sp.
↪Rational(15336726248, 1032824649), sp.Rational(-45442868181, 3398467696), sp.Rational(3065993473, 597172653)],
[sp.sympify(1), sp.Rational(185892177, 718116043), sp.sympify(0), sp.sympify(0), sp.Rational(-3185094517, 667107341), sp.
↪Rational(-477755414, 1098053517), sp.Rational(-703635378, 230739211), sp.Rational(5731566787, 1027545527), sp.Rational(5232866602,␣
↪850066563), sp.Rational(-4093664535, 808688257), sp.Rational(3962137247, 1805957418), sp.Rational(65686358, 487910083)],
[sp.sympify(1), sp.Rational(403863854, 491063109), sp.sympify(0), sp.sympify(0), sp.Rational(-5068492393, 434740067), sp.
↪Rational(-411421997, 543043805), sp.Rational(652783627, 914296604), sp.Rational(11173962825, 925320556), sp.Rational(-13158990841,␣
↪6184727034), sp.Rational(3936647629, 1978049680), sp.Rational(-160528059, 685178525), sp.Rational(248638103, 1413531060), sp.
↪sympify(0)],
["", sp.Rational(14005451, 335480064), sp.sympify(0), sp.sympify(0), sp.sympify(0), sp.sympify(0), sp.Rational(-59238493, 1068277825),␣
↪sp.Rational(181606767, 758867731), sp.Rational(561292985, 797845732), sp.Rational(-1041891430, 1371343529), sp.Rational(760417239,␣
↪1151165299), sp.Rational(118820643, 751138087), sp.Rational(-528747749, 2220607170), sp.Rational(1, 4)]]
, 8)
```

3 Step 3: The Family of Adaptive Runge-Kutta-Like Schemes (Butcher Tables) [Back to top]

In the previous parts of this notebook, we worked exclusively with Explicit Runge-Kutta-Like Schemes. There are a second category of schemes that are useful, however, and those are the Adaptive Runge-Kutta-Like Schemes. These methods, while technically still explicit in function, nonetheless need to be implemented differently. They have an extra row at the bottom of their butcher tables which define a second method of a different order. Adaptive Runge-Kutta-Like methods will calculate the predicted values of both results and use the difference between them to estimate the error of the lower-order method in a highly efficient manner. Details can be found on the Runge-Kutta wikipedia page.

As a reminder, the general result of any Runge-Kutta-Like scheme is

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

An Adaptive method will also calculate the following

$$y^*_{n+1} = y_n + \sum_{i=1}^{s} b^*_i k_i$$

Where the b^* coefficients are stored in the butcher table on an extra row, like so:

$$
\begin{array}{c|ccccc}
0 & & & & & \\
c_2 & a_{21} & & & & \\
c_3 & a_{31} & a_{32} & & & \\
\vdots & \vdots & \vdots & \ddots & & \\
c_s & a_{s_1} & a_{s2} & \cdots & a_{s,s-1} & \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s \\
 & b_1^* & b_2^* & \cdots & b_{s-1}^* & b_s^*
\end{array}
$$

From this method the error estimate is calcualted as $ e_{n+1} = y_{n+1} - y^*_{n+1} $. The exact use of this error is implementation-dependent, but in general if the error exceeds a provided threshhold the scheme will discard the step and try again at a smaller timestep, and if the error is below a provided threshhold the scheme will increase the step size (but in general it will not discard the data, it already calculated it, why waste it?).

The higher order method is the higher row of b-coefficients.

3.1 Step 3.a: Generating a Dictionary of Butcher Tables for Explicit Runge Kutta Techniques [Back to top]

As before, we can store all of the Butcher tables in Python's **Dictionary** format usingf the curly brackets {} and 'key':value pairs. The 'key' will be the *name* of the Runge Kutta method and the value will be the Butcher table itself stored as a list of lists. The convergence order for each Runge Kutta method is also stored. We will add to the dictionary `Butcher_dict` one Butcher table at a time in the following sections.

Adaptive methods are all prefaced with an A in their name, for clarity's sake.

3.1.1 Step 3.a.i: Adaptive Heun-Euler Method [Back to top]

The Heun-Euler method is a combination of Heun's second-order method and the first-order Euler method, creating the simplest Adaptive method.

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

[18]:
```
Butcher_dict['AHE'] = (
[[sp.sympify(0)],
[sp.sympify(1), sp.sympify(1)],
["", sp.Rational(1,2), sp.Rational(1,2)],
["", sp.sympify(1), sp.sympify(0)]]
, 2)
```

3.1.2 Step 3.a.ii: Adaptive Bogacki-Shampine Method [Back to top]

The Bogacki-Shampine Method is a third and second order method.

0			
1/2	1/2		
3/4	0	3/4	
1	2/9	1/3	
4/9			
	2/9	1/3	
4/9	0		
	7/24	1/4	
1/3	1/8		

[19]:
```
Butcher_dict['ABS'] = (
[[sp.sympify(0)],
[sp.Rational(1,2), sp.Rational(1,2)],
[sp.Rational(3,4), sp.sympify(0), sp.Rational(3,4)],
[sp.sympify(1), sp.Rational(2,9), sp.Rational(1,3), sp.Rational(4,9)],
["", sp.Rational(2,9), sp.Rational(1,3), sp.Rational(4,9), sp.sympify(0)],
["", sp.Rational(7,24), sp.Rational(1,4), sp.Rational(1,3), sp.Rational(1,8)]]
, 3)
```

3.1.3 Step 3.a.iii: Adaptive Runge-Kutta-Fehlberg [Back to top]

The Runge-Kutta-Fehlberg Method is a fifth and fourth order method.

$$
\begin{array}{c|cc}
0 & & \\
\frac{1}{4} & \frac{1}{4} & \\
\frac{3}{8} & \frac{3}{32} & \frac{9}{32} \\
\frac{12}{13} & \frac{1932}{2197} & \frac{-7200}{2197} \\
\frac{7296}{2197} & & \\
1 & \frac{439}{216} & -8 \\
\frac{3680}{513} & \frac{-845}{4104} & \\
\frac{1}{2} & \frac{-8}{27} & 2 \\
\frac{-3544}{2565} & \frac{1859}{4104} & \frac{11}{40} \\
\hline
 & \frac{16}{135} & 0 \\
\frac{6656}{12825} & \frac{28561}{56430} & \frac{-9}{50} \\
\frac{2}{55} & & \\
 & \frac{25}{216} & 0 \\
\frac{1408}{2565} & \frac{2197}{4104} & \frac{-1}{5} \\
0 & &
\end{array}
$$

[20]:
```
Butcher_dict['ARKF'] = (
[[sp.sympify(0)],
[sp.Rational(1,4), sp.Rational(1,4)],
[sp.Rational(3,8), sp.Rational(3,32), sp.Rational(9,32)],
[sp.Rational(12,13), sp.Rational(1932,2197), sp.Rational(-7200,2197), sp.Rational(7296,2197)],
[sp.sympify(1), sp.Rational(439,216), sp.sympify(-8), sp.Rational(3680,513), sp.Rational(-845,4104)],
[sp.Rational(1,2), sp.Rational(-8,27), sp.sympify(2), sp.Rational(-3544,2565), sp.Rational(1859,4104), sp.Rational(-11,40)],
["", sp.Rational(16,135), sp.sympify(0), sp.Rational(6656,12825), sp.Rational(28561,56430), sp.Rational(-9,50), sp.Rational(2,55)],
["", sp.Rational(25,216), sp.sympify(0), sp.Rational(1408,2565), sp.Rational(2197,4104), sp.Rational(-1,5), sp.sympify(0)]]
, 5)
```

3.1.4 Step 3.a.iv: Adaptive Cash-Karp [Back to top]

The Cash-Karp Method is a fifth and fourth order method.

$$
\begin{array}{c|cc}
0 & & \\
\frac{1}{5} & \frac{1}{5} & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} \\
\frac{3}{5} & \frac{3}{10} & \frac{-9}{10} \\
\frac{6}{5} & & \\
1 & \frac{-11}{54} & \frac{5}{2} \\
\frac{-70}{27} & \frac{35}{27} & \\
\frac{7}{8} & \frac{1631}{55296} & \frac{175}{512} \\
\frac{575}{1384} & \frac{44275}{110592} & \frac{-253}{4096} \\
\hline
 & \frac{37}{378} & 0 \\
\frac{250}{621} & \frac{125}{594} & 0 \\
\frac{512}{1771} & & \\
 & \frac{2825}{27648} & 0 \\
\frac{18575}{48384} & \frac{13525}{55296} & \frac{277}{14336} \\
\frac{1}{4} & &
\end{array}
$$

[21]:
```
Butcher_dict['ACK'] = (
[[0],
[sp.Rational(1,5), sp.Rational(1,5)],
[sp.Rational(3,10),sp.Rational(3,40), sp.Rational(9,40)],
[sp.Rational(3,5), sp.Rational(3,10), sp.Rational(-9,10), sp.Rational(6,5)],
[sp.sympify(1), sp.Rational(-11,54), sp.Rational(5,2), sp.Rational(-70,27), sp.Rational(35,27)],
[sp.Rational(7,8), sp.Rational(1631,55296), sp.Rational(175,512), sp.Rational(575,13824), sp.Rational(44275,110592), sp.
 ↪Rational(253,4096)],
["",sp.Rational(37,378), sp.sympify(0), sp.Rational(250,621), sp.Rational(125,594), sp.sympify(0), sp.Rational(512,1771)],
["",sp.Rational(2825,27648), sp.sympify(0), sp.Rational(18575,48384), sp.Rational(13525,55296), sp.Rational(277,14336), sp.
 ↪Rational(1,4)]]
, 5)
```

3.1.5 Step 3.a.v: Adaptive Dormand-Prince 5(4) [Back to top]

The Dormand-Prince5(4) Method is a fifth and fourth order method.

$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & & \\
\frac{4}{5} & \frac{44}{45} & \frac{-56}{15} & \frac{32}{9} & & & & \\
\frac{8}{9} & \frac{19372}{6561} & \frac{-25360}{2187} & \frac{64448}{6561} & \frac{-212}{729} & & & \\
1 & \frac{9017}{3168} & \frac{-355}{33} & \frac{46732}{5247} & \frac{49}{176} & \frac{-5103}{18656} & & \\
1 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & \frac{-2187}{6784} & \frac{11}{84} & \\
\hline
 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & \frac{-2187}{6784} & \frac{11}{84} & 0 \\
 & \frac{5179}{57600} & 0 & \frac{7571}{16695} & \frac{393}{640} & \frac{-92097}{339200} & \frac{187}{2100} & \frac{1}{40}
\end{array}
.
$$

[22]:
```
Butcher_dict['ADP5'] = (
[[0],
[sp.Rational(1,5), sp.Rational(1,5)],
[sp.Rational(3,10),sp.Rational(3,40), sp.Rational(9,40)],
[sp.Rational(4,5), sp.Rational(44,45), sp.Rational(-56,15), sp.Rational(32,9)],
[sp.Rational(8,9), sp.Rational(19372,6561), sp.Rational(-25360,2187), sp.Rational(64448,6561), sp.Rational(-212,729)],
[sp.sympify(1), sp.Rational(9017,3168), sp.Rational(-355,33), sp.Rational(46732,5247), sp.Rational(49,176), sp.Rational(-5103,18656)],
[sp.sympify(1), sp.Rational(35,384), sp.sympify(0), sp.Rational(500,1113), sp.Rational(125,192), sp.Rational(-2187,6784), sp.
 ↪Rational(11,84)],
["", sp.Rational(35,384), sp.sympify(0), sp.Rational(500,1113), sp.Rational(125,192), sp.Rational(-2187,6784), sp.Rational(11,84), sp.
 ↪sympify(0)],
["", sp.Rational(5179,57600), sp.sympify(0), sp.Rational(7571,16695), sp.Rational(393,640), sp.Rational(-92097,339200), sp.
 ↪Rational(187,2100), sp.Rational(1,40)]]
, 5)
```

3.1.6 Step 3.vi: Adaptive Dormand-Prince 8(7) [Back to top]

The Domand-Prince 8(7) Method is an eighth and seventh order method.

$$
\begin{array}{c|cccccccc}
0 & & & & & & & & \\
\frac{1}{18} & \frac{1}{18} & & & & & & & \\
\frac{1}{12} & \frac{1}{48} & \frac{1}{16} & & & & & & \\
\frac{1}{8} & \frac{1}{32} & 0 & \frac{3}{32} & & & & & \\
\frac{5}{16} & \frac{5}{16} & 0 & \frac{-75}{64} & \frac{75}{64} & & & & \\
\frac{3}{8} & \frac{3}{80} & 0 & 0 & \frac{3}{16} & \frac{3}{20} & & & \\
\frac{59}{400} & \frac{29443841}{614563906} & 0 & 0 & \frac{77736538}{692538347} & \frac{-28693883}{1125000000} & \frac{23124283}{1800000000} & & \\
\frac{93}{200} & \frac{16016141}{946692911} & 0 & 0 & \frac{61564180}{158732637} & \frac{22789713}{633445777} & \frac{545815736}{2771057229} & \frac{-180193667}{1043307555} & \\
\frac{5490023248}{9719169821} & \frac{39632708}{573591083} & 0 & 0 & \frac{-433636366}{683701615} & \frac{-421739975}{2616292301} & \frac{100302831}{723423059} & \frac{790204164}{839813087} & \frac{800635310}{3783071287} \\
\frac{13}{20} & \frac{246121993}{1340847787} & 0 & 0 & \frac{-37695042795}{15268766246} & \frac{-309121744}{1061227803} & \frac{-12992083}{490766935} & \frac{6005943493}{2108947869} & \frac{393006217}{1396673457} & \frac{123872331}{1001029789} \\
\frac{1201146811}{1299019798} & \frac{-1028468189}{846180014} & 0 & 0 & \frac{8478235783}{508512852} & \frac{1311729495}{1432422823} & \frac{-10304129995}{1701304382} & \frac{-48777925059}{3047939560} & \frac{15336726248}{1032824649} & \frac{-45442868181}{3398467696} \\
\frac{3065993473}{597172653} & \frac{185892177}{718116043} & 0 & 0 & \frac{-3185094517}{667107341} & \frac{-477755414}{1098053517} & \frac{-703635378}{230739211} & \frac{5731566787}{1027545527} & \frac{5232866602}{850066563} & \frac{-4093664535}{808688257} \\
1 & \frac{65686358}{487910083} & & & & & & & & \\
\frac{3962137247}{1805957418} & \frac{403863854}{491063109} & 0 & 0 & \frac{-5068492393}{434740067} & \frac{-411421997}{543043805} & \frac{652783627}{914296604} & \frac{11173962825}{925320556} & \frac{-13158990841}{6184727034} & \frac{3936647629}{1978049680} \\
1 & \frac{248638103}{1413531060} & 0 & & & & & & & \\
\frac{-160528059}{685178525} & \frac{14005451}{335480064} & 0 & 0 & 0 & 0 & \frac{-59238493}{1068277825} & \frac{181606767}{758867731} & \frac{561292985}{797845732} & \frac{-1041891430}{1371343529} \\
\frac{760417239}{1151165299} & \frac{118820643}{751138087} & \frac{-528747749}{2220607170} & \frac{1}{4} & & & & & & \\
 & \frac{13451932}{455176623} & 0 & 0 & 0 & 0 & \frac{-808719846}{976000145} & \frac{1757004468}{5645159321} & \frac{656045339}{265891186} & \frac{-3867574721}{1518517206} \\
\frac{465885868}{322736535} & \frac{53011238}{667516719} & \frac{2}{45} & 0 & & & & & &
\end{array}
.
$$

[23]:
```
Butcher_dict['ADP8']=(
[[0],
[sp.Rational(1, 18), sp.Rational(1, 18)],
[sp.Rational(1, 12), sp.Rational(1, 48), sp.Rational(1, 16)],
[sp.Rational(1, 8), sp.Rational(1, 32), sp.sympify(0), sp.Rational(3, 32)],
[sp.Rational(5, 16), sp.Rational(5, 16), sp.sympify(0), sp.Rational(-75, 64), sp.Rational(75, 64)],
[sp.Rational(3, 8), sp.Rational(3, 80), sp.sympify(0), sp.sympify(0), sp.Rational(3, 16), sp.Rational(3, 20)],
[sp.Rational(59, 400), sp.Rational(29443841, 614563906), sp.sympify(0), sp.sympify(0), sp.Rational(77736538, 692538347), sp.
 ↪Rational(-28693883, 1125000000), sp.Rational(23124283, 1800000000)],
[sp.Rational(93, 200), sp.Rational(16016141, 946692911), sp.sympify(0), sp.sympify(0), sp.Rational(61564180, 158732637), sp.
 ↪Rational(22789713, 633445777), sp.Rational(545815736, 2771057229), sp.Rational(-180193667, 1043307555)],
[sp.Rational(5490023248, 9719169821), sp.Rational(39632708, 573591083), sp.sympify(0), sp.sympify(0), sp.Rational(-433636366,␣
 ↪683701615), sp.Rational(-421739975, 2616292301), sp.Rational(100302831, 723423059), sp.Rational(790204164, 839813087), sp.
 ↪Rational(800635310, 3783071287)],
[sp.Rational(13, 20), sp.Rational(246121993, 1340847787), sp.sympify(0), sp.sympify(0), sp.Rational(-37695042795, 15268766246), sp.
 ↪Rational(-309121744, 1061227803), sp.Rational(-12992083, 490766935), sp.Rational(6005943493, 2108947869), sp.Rational(393006217,␣
 ↪1396673457), sp.Rational(123872331, 1001029789)],
[sp.Rational(1201146811, 1299019798), sp.Rational(-1028468189, 846180014), sp.sympify(0), sp.sympify(0), sp.Rational(8478235783,␣
 ↪508512852), sp.Rational(1311729495, 1432422823), sp.Rational(-10304129995, 1701304382), sp.Rational(-48777925059, 3047939560), sp.
 ↪Rational(15336726248, 1032824649), sp.Rational(-45442868181, 3398467696), sp.Rational(3065993473, 597172653)],
```

```
[sp.sympify(1), sp.Rational(185892177, 718116043), sp.sympify(0), sp.sympify(0), sp.Rational(-3185094517, 667107341), sp.
↪Rational(-477755414, 1098053517), sp.Rational(-703635378, 230739211), sp.Rational(5731566787, 1027545527), sp.Rational(5232866602,␣
↪850066563), sp.Rational(-4093664535, 808688257), sp.Rational(3962137247, 1805957418), sp.Rational(65686358, 487910083)],
[sp.sympify(1), sp.Rational(403863854, 491063109), sp.sympify(0), sp.sympify(0), sp.Rational(-5068492393, 434740067), sp.
↪Rational(-411421997, 543043805), sp.Rational(652783627, 914296604), sp.Rational(11173962825, 925320556), sp.Rational(-13158990841,␣
↪6184727034), sp.Rational(3936647629, 1978049680), sp.Rational(-160528059, 685178525), sp.Rational(248638103, 1413531060), sp.
↪sympify(0)],
["", sp.Rational(14005451, 335480064), sp.sympify(0), sp.sympify(0), sp.sympify(0), sp.sympify(0), sp.Rational(-59238493, 1068277825),␣
↪sp.Rational(181606767, 758867731), sp.Rational(561292985, 797845732), sp.Rational(-1041891430, 1371343529), sp.Rational(760417239,␣
↪1151165299), sp.Rational(118820643, 751138087), sp.Rational(-528747749, 2220607170), sp.Rational(1, 4)],
["", sp.Rational(13451932, 455176623), sp.sympify(0), sp.sympify(0), sp.sympify(0), sp.sympify(0), sp.Rational(-808719846, 976000145),␣
↪sp.Rational(1757004468, 5645159321), sp.Rational(656045339, 265891186), sp.Rational(-3867574721, 1518517206), sp.Rational(465885868,␣
↪322736535), sp.Rational(53011238, 667516719), sp.Rational(2, 45), sp.sympify(0)]]
, 8)
```

4 Step 4: The Adams-Bashforth Method [Back to top]

There is one method we have on offer that is not an RK-style method. Instead of taking the previous step and evolving to the next one, it takes into account several prior steps and extrapolates from them to the next step. This is the Adams-Bashforth method. Its primary weakness is that it requires several points be evaluated before it can run (specifically, points equal to the desired order of the method). However, its strengths are that it takes into account longer-scale patterns as well as being able to be scaled up to arbitrary order. More detailed information can be found on the Wikipedia page.

The first few Adams-Bashforth methods are as follows.

$$y_{n+1} = y_n + hf(y_n, t_n)$$
$$y_{n+2} = y_{n+1} + h\left(\frac{3}{2}f(y_{n+1}, t_{n+1}) - \frac{1}{2}f(y_n, t_n)\right)$$
$$y_{n+3} = y_{n+2} + h\left(\frac{23}{12}f(y_{n+2}, t_{n+2}) - \frac{16}{12}f(y_{n+1}, t_{n+1}) + \frac{5}{12}f(y_n, t_n)\right)$$

Careful readers may have noticed that the first Adams-Bashforth (AB) method is identical to the Euler method. Note that the second and third-order AB methods require 2 and 3 previous points, respectively, to evaluate the next one. This trend continues to arbitrary order. In general, a method of order s can be stated as

$$y_{n+s} = y_{n+s-1} + h\sum_{i=0}^{s-1} a_i f(y_{n+i}, t_{n+i}).$$

For any singular method, only a vector needs to be specified, the values a_i for the order in question. These coefficients have a closed-form expression for them, although it isn't neat.

$$a_{s-j-1} = \frac{(-1)^j}{j!(s-j-1)!} \int_0^1 \prod_{i=0;i\neq j}^{s-1} (u+i)du$$

From this arbitrary order methods can be chosen. In order to get the order number of points that AB requires to extrapolate, there are a couple of options. If the user does not wish to use RK methods, one can use a "ramping up" effect, starting with the first order AB method, then using those values in the second order, then the second order in the third, and so on until the desired order is reached. More often, though, an RK method is used at the start and then extrapolation is passed to the AB method. Do note: the AB method as dictated here requires that the step size be constant, since it cares a lot about the relation between the previous points. While it is possible to make a generic AB method that can take points separated by any variable step size, that functionality is not included here as it would ruin the closed-form relatively simple analytic expression for the coefficients.

Rather than providing a single table, we actually have a code that can generate a table for *any* AB method, though by default we have it create one of order 19, chosen because it's an arbitrarily large number that we should never need to use. Strictly speaking, only one vector of values is required for an AB method, but when we generate the 19th order vector, we also choose to generate all previous orders so if we wished to use a pure-AB method without relying on RK to seed values, we have access to the lower-order methods.

Of note: experiments with using the AB method indicate that the really high order methods are not necessarily more accurate, believed to be due to roundoff error acumulating over time. It has been observed (though not rigorously proven) that once an AB method is calculating near roundoff error, higher order methods will be less accurate, so the best AB order to use is likely the lowest order one that evaluates near roundoff error.

[24]:
```
import numpy as np
from sympy import symbols
from sympy import integrate

# okay so our goal here is to calculate the coefficients to arbitrary precision for
# Adams-Bashforth methods.
# Careful! AB methods are NOT structured the same as RK methods, the same code
# cannot read them!

# this does not work to generate an order 1 method. But why would you need to generate
# that, it's just a 1x1 matrix with a 1 in it.
order = 19 # change this if you want different orders. Default 19, which takes several seconds to generate.
pythonButcher = [[sp.Rational(0.0/1.0),sp.Rational(0.0/1.0)],
                 [sp.Rational(0.0/1.0),sp.Rational(0.0/1.0)]]
# just to initialize it, we'll set it in a minute

n = 1
while n < order+1:
    j = 0
    while j < n: # the number of coefficients in each method is equal to the order.

        # set up the product
        x = symbols('x')
        expr = x
        i = 0
```

```
        while i < n:
            if i == j:
                i = i+1
            else:
                expr = expr*(x + i)
                i = i+1
        expr = expr/x
        expr2 = integrate(expr,(x,0,1))
        expr2 = expr2 * sp.Rational((-1.0)**j,(sp.factorial(j)* sp.factorial(n - j - 1)))
        if (len(pythonButcher) < n):
            pythonButcher.append([sp.Rational(0.0/1.0),sp.Rational(0.0/1.0)])
            # add another row if we need it.
        if (len(pythonButcher[n-1]) < j+1):
            pythonButcher[n-1].append(expr2)
            i = 0
            while i < n:
                # Uncomment this section to fill the matrix with zeroes
                # if(len(pythonButcher[i]) < len(pythonButcher[n-1])):
                    # pythonButcher[i].append(sp.sympify(0))
                i = i+1
        else:
            pythonButcher[n-1][j] = expr2
        j = j+1
    n = n+1

# Due to the way we set this up to make sure everything was explicitly an array,
# We need to remove a single trailing zero.
# Comment out if you want the zeroes.
pythonButcher[0] = [1]

Butcher_dict['AB']=(
pythonButcher
, order)
```

5 Step 5: Code validation against `MoLtimestepping.RK_Butcher_Table_Dictionary` NRPy+ module [Back to top]

As a code validation check, we verify agreement in the dictionary of Butcher tables between 1. this tutorial and 2. the NRPy+ MoLtimestepping.RK_Butcher_Table_Dictionary module.

We analyze all key/value entries in the dictionary for consistency.

[25]:
```
# Step 3: Code validation against MoLtimestepping.RK_Butcher_Table_Dictionary NRPy+ module
import sys  # Standard Python module for multiplatform OS-level functions
from MoLtimestepping.RK_Butcher_Table_Dictionary import Butcher_dict as B_dict
valid = True
for key, value in Butcher_dict.items():
    if Butcher_dict[key] != B_dict[key]:
        valid = False
        print(key)
if valid == True and len(Butcher_dict.items()) == len(B_dict.items()):
    print("The dictionaries match!")
else:
    print("ERROR: Dictionaries don't match!")
    sys.exit(1)
```

```
The dictionaries match!
```

6 Step 6: Output this notebook to LaTeX-formatted PDF file [Back to top]

The following code cell converts this Jupyter notebook into a proper, clickable LaTeX-formatted PDF file. After the cell is successfully run, the generated PDF may be found in the root NRPy+ tutorial directory, with filename Tutorial-RK_Butcher_Table_Dictionary.pdf. (Note that clicking on this link may not work; you may need to open the PDF file through another means.)

[26]:
```
import cmdline_helper as cmd    # NRPy+: Multi-platform Python command-line interface
cmd.output_Jupyter_notebook_to_LaTeXed_PDF("Tutorial-RK_Butcher_Table_Dictionary")
```

```
Created Tutorial-RK_Butcher_Table_Dictionary.tex, and compiled LaTeX file
    to PDF file Tutorial-RK_Butcher_Table_Dictionary.pdf
```