GEORGIA STATE UNIVERSITY
Department of Mathematics and Statistics

METHODS FOR SOLVING INITIAL VALUE PROBLEMS - A Report

BY

JESSE ANNAN

April 15, 2023

Contents

List of Algorithms

List of Tables

List of Figures

List of Abbreviations

IVP Initial Value Problem

DE Differential Equation

ODE Ordinary Differential Equation

RKF Runge-Kutta-Fehlberg Method

RK2 2nd-Order Runge-Kutta Method

RK4 4th-Order Runge-Kutta Method

LTE Local Truncation Error

GTE Global Truncation Error

Chapter 1

INTRODUCTION

An Initial Value Problem (IVP) is a Differential Equation (DE) together with one or more initial values.[11][14] It takes what would otherwise be an entire rainbow of possible solutions and whittles them down to one specific solution. The basic idea behind this problem is that, once you differentiate a function, you lose some information about that function, more specifically, you lose the constant. By integrating $y'(x)$, you get a family of solutions that only differ by a constant.[9]

1.1 Definition

An IVP is a differential equation: [14]

$$\begin{aligned} y'(t) = f(t, y(t)) \text{ with } f : \Omega \subset \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n \\ (t_0, y_0) \in \Omega, \text{ called the initial condition.} \end{aligned} \tag{1.1}$$

Observations:

1. The given f in (1.1) is the defining function of IVP.

2. A unique solution, $y(t)$, of the (1.1) exists and it satisfies $y(t_0) = y_0$.

Example

Given $y'(t) = 5$ and $y(0) = -3$, find $y(t)$.

Solution:

We first integrate our $y'(t)$, then we substitute our initial condition to determine the constant (from our integration).

$$\begin{aligned}
\int y'(t)dt &= \int 5dt \\
y(t) &= 5x + c \quad \text{where } c \text{ is the constant of integration} \\
\text{using } y(t = 0) &= -3 \\
-3 &= 5(0) + c \\
-3 &= c \\
y(t) &= 2x - 3
\end{aligned}$$

Remark: Note that with a different $y(0)$, the solution would be different.

1.2 Objective

In real-life situations, the differential equation that models a problem is too complicated to solve exactly, therefore one of the ways which is used to solved such problems is using methods which approximates the solution of the original problem.[2] In this report, I will discuss methods that approximates solutions at certain specified timestamps.
They are: [§ 2.1] The Euler's Method, [§ 2.2] The Modified Euler's Method, [§ 2.3] The 2nd-Order Runge-Kutta Method, [§ 2.4] The 4th-Order Runge-Kutta Method, [§ 2.5] The Adams-Bashforth 4th-Order Explicit, [§ 2.6] The Adams 4th-Order Predictor Corrector, [§ 2.7] The Runge-Kutta-Fehlberg, and [§ 2.8] The Predictor-Corrector methods

Chapter 2

METHODS

2.1 The Euler's Method

The Euler method, named after Leonhard Euler, was published in his three-volume work *Institutiones Calculi Integralis* in the years 1768 to 1770, and republished in his collected works.(Euler, 1913) [3] The Euler method is a first-order numerical procedure for solving Ordinary Differential Equation (ODE) with a given initial value. It is the most basic explicit method for numerical integration of ODE and is the simplest Runge–Kutta method.[12]

The fundamental idea of the method is based on the principle that, we can compute (or approximate) the shape of an unknown curve - in the form of a differential equation $f(t, y)$, which starts at a given point y_0 and time t_0. With this information known, we can proceed to calculate the slope (tangent line) of the curve at y_0.

The tangent line is [5]

$$y = y_0 + f(t_0, y_0) \cdot (t - t_0)$$

Now we assume that $f(t0, y0)$ is sufficiently accurate, and thus, taking a small step along that tangent line, we can approximate the actual value of the solution, y_1, at timestamp t_1, using the formula:

$$y_1 = y_0 + f(t_0, y_0) \cdot (t_1 - t_0) \tag{2.1}$$

In general, we continue to find the next approximated solution y_{n+1} at t_{n+1}, if we have the nth timestamp t_n and the approximation to the solution at this point, y_n.

We only need to modify (2.1) in this manner:

$$y_{n+1} = y_n + f(t_n, y_n) \cdot (t_{n+1} - t_n) \tag{2.2a}$$

If we assume uniform step sizes between times, t, we can define, $h = t_{n+1} - t_n$. Therefore, the formula is simplified as [5]

$$y_{n+1} = y_n + h \cdot f(t_n, y_n) \tag{2.2b}$$

The Truncation Errors

1. The **Local Truncation Error (LTE)** of the Euler method is the error made in a single step. It is the difference between the numerical solution after one step, y_1, and the exact solution (obtained using Taylor's expansion) at time $t_1 = t_0 + h$.[12]

$$\begin{aligned} \text{The numerical solution: } y_1 &= y_0 + hf(t_0, y_0) \\ \text{The exact solution: } y(t_0 + h) &= y(t_0) + hy'(t_0) + \frac{1}{2}h^2 y''(t_0) + O(h^3) \\ \text{LTE} = y(t_0 + h) - y_1 &= \frac{1}{2}h^2 y''(t_0) + O(h^3) \end{aligned}$$

2. The **Global Truncation Error (GTE)** is the error at a fixed time t_i, after however many steps the method needs to take to reach that time from the initial time. The global truncation error is the cumulative effect of the local truncation errors committed in each step.[1]

$$|y(t_i) - y_i| \leq \frac{hM}{2L}(e^{L(t_i - t_0)} - 1)$$

where M is an upper bound on the second derivative of y on the given interval and L is the Lipschitz constant of f.[1]

The Pseudocode

To approximate the solution of the IVP

$$y' = f(t, y), \;\; a \leq t \leq b, \;\; y(a) = \alpha$$

at N equally spaced numbers in the interval $[a, b]$: [2]

Algorithm 1 :: Euler's Method

Require: endpoints a, b; integer N; initial condition α

Ensure: approximation w to y at the $(N + 1)$ values of t

```
h = (b − a)/N;  t_0 = a;  w_0 = α
for i = 0, 1, 2, ··· , N − 1 do
    w_{i+1} = w_i + h f(t_i, w_i);      {Compute next w_i}
    t = a + ih      {Compute next t_i}
end for
return (t, w)
```

2.2 The Modified Euler's Method

Euler's method is used as the foundation for Modified Euler's method. Euler's method uses the line tangent to the function at the beginning of the interval as an estimate of the slope of the function over the interval, assuming that if the step size is small, the error will be small. However, even when extremely small step sizes are used, over a large number of steps the error starts to accumulate and the estimate diverges from the actual functional value.[13]
The Modified Euler (which may sparingly be referred to as the Heun's method [13]) was developed to improve the approximated solution at t_{i+1} by taking the arithmetic average of the approximated solution at the slopes t_i and t_{i+1}.
The procedure for calculating the numerical solution to the (1.1) by first computing the Euler method to roughly estimate the coordinates of the next point in the solution, and then, the original estimate is recalculated using the rough estimate [4]:

$$\begin{aligned} \text{rough estimate: } & \tilde{y}_{i+1} = y_i + hf(t_i, y_i) \\ \text{original estimate: } & y_{i+1} = y_i + \frac{h}{2}\left[f(t_i, y_i) + f(t_{i+1}, \tilde{y}_{i+1})\right] \end{aligned} \tag{2.2}$$

where h is the step size an $t_{i+1} = t_i + h$.

The Truncation Errors

the local truncation error is $O(h^3)$. The modified Euler Method is second order accurate.

The Pseudocode

To approximate the solution of the IVP

$$y' = f(t, y), \;\; a \leq t \leq b, \;\; y(a) = \alpha$$

at N equally spaced numbers in the interval $[a, b]$:

Algorithm 2 :: Modified Euler's Method

Require: endpoints a, b; integer N; initial condition α

Ensure: approximation w to y at the N values of t

```
h = (b − a)/N;  t_0 = a;  w_0 = α
for i = 0, 1, 2, ··· , N − 1 do
    w̃_{i+1} = w_i + h f(t_i, w_i);      {Compute rough (next) w_i}
    w_{i+1} = w_i + (h/2) [f(t_i, w_i) + f(t_i + h, w̃_{i+1})];      {Compute corrected (next) w_i}
    t = a + ih      {Compute next t_i}
end for
return (t, w)
```

2.3 The 2nd-Order Runge-Kutta Method

The Runge-Kutta methods have the higher-order local truncation error of the Taylor methods while eliminating the need to compute and evaluate the derivative of $f(t, y)$. The RK2 is one of the methods that computes the approximation of y for a given timestamp. The RK2 can only be used to used to solve first-order ordinary differential equations. This is done by computing, k_1, the increment based on the slopes at the beginning of the interval using y and k_2, the increment based on the slope at the midpoint of the interval, using $(y + \frac{h}{2}k_1)$ [7]

$$\begin{aligned} w_0 &= \alpha \\ k_1 &= f(t_i, w_i) \\ k_2 &= f\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}k_1\right); \\ w_{i+1} &= w_i + hk_2 \quad \forall i = 0, 1, \cdots, N-1 \end{aligned} \tag{2.3}$$

The Truncation Errors

The method is a second-order method, meaning that the local truncation error is in the order of $O(h^3)$, while the total accumulated error is order of $O(h^4)$.[7]

The Pseudocode

To approximate the solution of the IVP

$$y' = f(t, y), \;\; a \leq t \leq b, \;\; y(a) = \alpha$$

at N equally spaced numbers in the interval $[a, b]$:

Algorithm 3 :: RK2

Require: endpoints a, b; integer N; initial condition α

Ensure: approximation w to y at the N values of t

$h = (b - a)/N;\;\; t_0 = a;\;\; w_0 = \alpha$

for $i = 0, 1, 2, \cdots, N - 1$ **do**

  $k_1 = f(t_i, w_i);$ {Compute k_1}

  $k_2 = f\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}k_1\right);$ {Compute k_2}

  $w_{i+1} = w_i + hk_2;$ {Compute next w_i}

  $t = a + ih$ {Compute next t_i}

end for

return (t, w)

2.4 The 4th-Order Runge-Kutta Method

(2.3) can be extended to higher order methods. The RK4 is another scheme of the Runge-Kutta methods. This methods involves computing four increments $k_{1,2,3,4}$ (instead of two done above by RK2). Each increment is the product of the size of the interval, h. k_1 is the slope at the beginning of the interval, using y (similar to Euler's method), k_2 is the slope at the midpoint of the interval, using the y and k_1, k_3 is the slope at the midpoint, using y and k_2 and finally the forth k, k_4 is the slope at the end of the interval, using y and k_3.[15]

$$
\begin{aligned}
k_1 &= f(t_i, w_i) \\
k_2 &= f\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}k_1\right); \\
k_3 &= f\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}k_2\right); \\
k_4 &= f\left(t_i + h, w_i + hk_3\right); \\
w_{i+1} &= w_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4);
\end{aligned}
\tag{2.4}
$$

The Truncation Errors

Since RK4 needs to evaluate the function f for four times in each step meaning the local truncation error is on the order of $O(h^5)$, while the total accumulated error is on the order of $O(h^4)$.[15]

The Pseudocode

To approximate the solution of the IVP

$$y' = f(t, y), \;\; a \le t \le b, \;\; y(a) = \alpha$$

at N equally spaced numbers in the interval $[a, b]$: [2]

Algorithm 4 :: RK4

Require: endpoints a, b; integer N; initial condition α

Ensure: approximation w to y at the N values of t

$h = (b - a)/N$; $t_0 = a$; $w_0 = \alpha$

for $i = 0, 1, 2, \cdots, N - 1$ **do**

  $k_1 = f(t_i, w_i)$; $\{$Compute k_1 *to* $k_4\}$

  $k_2 = f\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}k_1\right)$;

  $k_3 = f\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}k_2\right)$;

  $k_4 = f\left(t_i + h, w_i + hk_3\right)$;

  $K = k_1 + 2k_2 + 2k_3 + k_4$

  $w_{i+1} = w_i + \frac{h}{6}K$; $\{$Compute next $w_i\}$

  $t = a + ih$ $\{$Compute next $t_i\}$

end for

return (t, w)

2.5 The Adams-Bashforth 4th-Order Explicit Method

$$\begin{aligned} w_0 &= \alpha \\ w_1 &= \alpha_1 \\ w_2 &= \alpha_2 \\ w_3 &= \alpha_3 \end{aligned} \tag{2.5}$$

The Truncation Errors

The Adams-Bashforth 4th-Order Explicit Method local truncation error is [2]

$$\tau_{i+1}(h) = \frac{251}{720} y^5(\mu_i) h^4$$

for some $\mu_i \in (t_{i-3}, t_{i+1})$

The Pseudocode

To approximate the solution of the IVP

$$y' = f(t, y), \ \ a \leq t \leq b, \ \ y(a) = \alpha$$

at N equally spaced numbers in the interval $[a, b]$: [2]

Algorithm 5 :: Adams-Bashforth 4th-Order Explicit Method

Require: endpoints a, b; integer N; initial condition α

Ensure: approximation w to y at the N values of t

Compute the first 3 values for t_i and w_i using the $RK4$ method
Call $RK4$ { RK4 method above }
$h = (b - a)/N;\ t_0 = a;\ w_0 = \alpha$
for $i = 3, 4, 5, \cdots, N - 1$ **do**
    $k_1 = f(t_i, w_i);$ {Compute k_1 *to* k_4}
    $k_2 = f(t_i - h, w_{i-1});$
    $k_3 = f(t_i - 2h, w_{i-2});$
    $k_4 = f(t_i - 3h, w_{i-3});$
    $K = 55k_1 - 59k_2 + 37k_3 - 9k_4$
    $w_{i+1} = w_i + \frac{h}{24}K;$ {Compute next w_i}
    $t = a + ih$ {Compute next t_i}
end for
return (t, w)

2.6 The Adams 4th-Order Predictor-Corrector Method

The methods discussed above only involves predicting the approximate solution just ones. The Adams-Bashforth 4th-Order Explicit Method is one of the many multi-stage methods. The method involves the combination of an explicit and implicit technique and is called a predictor-corrector method. [2] The explicit method predicts an approximation, and the implicit method corrects this prediction.
Consider the (IVP) stated above. The first step is to calculate the starting values w_0, w_1, w_2 and w_3 for the four-step explicit Adams-Bashforth method. This is done by using the Runge-Kutta 4th-order method, then approximate the immediate w_i using the explicit Adams-Bashforth. The approximation is then improved in the next step using the three-step impplicit Adams-Moulton method: [2]

$$\begin{aligned}
\text{Predict } w_i\text{: } w_p &= w_3 + h[55f(t_3.w_3) - 59f(t_2, w_2) \\
&\quad + 37f(t_1, w_1) - 9f(t_0, w_0)]/24; \\
\text{Correct } w_i\text{: } w_c &= w_3 + h[9f(t_{i+1}.w_{i+1}) - 19f(t_3, w_3) \\
&\quad - 5f(t_2, w_2) + f(t_1, w_1)]/24;
\end{aligned} \tag{2.6}$$

The Truncation Errors

The Adams 4th-Order Predictor-Corrector Method has an error order of $O(h^4)$. [2]

The Pseudocode

To approximate the solution of the IVP

$$y' = f(t, y), \;\; a \le t \le b, \;\; y(a) = \alpha$$

at N equally spaced numbers in the interval $[a, b]$: [2]

Algorithm 6 :: Adams Forth-Order Predictor-Corrector

Require: endpoints a, b; integer N; initial condition α

Ensure: approximation w to y at the N values of t

$h = (b - a)/N$

$t_0 = a$

$w_0 = \alpha$

for $i = 0, 1, 2$ **do**

  $k_1 = hf(t_i, w_i)$; {Compute starting values using Runge-Kutta Method}

  $k_2 = hf(t_i + \frac{h}{2}, w_i + \frac{k_1}{2})$;

  $k_3 = hf(t_i + \frac{h}{2}, w_i + \frac{k_2}{2})$;

  $k_4 = hf(t_i + h, w_i + k_3)$;

  $K = k_1 + 2k_2 + 2k_3 + k_4$

  $w_{i+1} = w_i + \frac{K}{6}$;

  $t_{i+1} = a + ih$

  return (t_i, w_i)

end for

for $i = 3, \cdots, N - 1$ **do**

  $t = a + ih$;

  $w_{i+1} = w_3 + h\frac{[55f(t_3.w_3) - 59f(t_2,w_2) + 37f(t_1,w_1) - 9f(t_0,w_0)]}{24}$; {Predict w_{i+1}}

  $w_{i+1} = w_3 + h\frac{[9f(t_{i+1}.w_{i+1}) - 19f(t_3,w_3) - 5f(t_2,w_2) + f(t_1,w_1)]}{24}$; {Correct w_{i+1}}

  return (t, w)

  for $j = 0, 1, 2$ **do**

    $t_j = t_{j+1}$;

    $w_j = w_{j+1}$

  end for

  $t_3 = t_{i+1}$;

  $w_3 = w_{i+1}$

end for

2.7 The Runge-Kutta-Fehlberg Method

The Runge–Kutta–Fehlberg method (or Fehlberg method) is an algorithm in numerical analysis for the numerical solution of ordinary differential equations. It was developed by the German mathematician Erwin Fehlberg and is based on the large class of Runge–Kutta methods. The novelty of Fehlberg's method is that it is an embedded method from the Runge–Kutta family, meaning that identical function evaluations are used in conjunction with each other to create methods of varying order and similar error constants. The method presented in Fehlberg's 1969 paper has been dubbed the RKF45 method, and is a method of order 4 with an error estimator of order 5. By performing one extra calculation, the error in the solution can be estimated and controlled by using the higher-order embedded method that allows for an adaptive stepsize to be determined automatically.[10]

$$
\begin{aligned}
k_1 &= hf(t, w);\\
k_2 &= hf(t + \frac{1}{4}h, w + \frac{1}{4}k_1);\\
k_3 &= hf(t + \frac{3}{8}h, w + \frac{3}{32}k_1 + \frac{9}{32}k_2);\\
k_4 &= hf(t + \frac{12}{13}h, w + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3);\\
k_5 &= hf(t + h, w + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4);\\
k_6 &= hf(t + \frac{1}{2}h, w - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5);
\end{aligned}
\tag{2.7}
$$

The Truncation Errors

It can be shown that the first formula is an O(h5) while the second is $O(h^6)$ (though only $O(h^4)$ and $O(h^5)$. [8]

The Pseudocode

To approximate the solution of the IVP $y' = f(t, y),\ \ a \leq t \leq b,\ \ y(a) = \alpha$ with local truncation error within a given tolerance: [2]

Algorithm 7 :: RKF

Require: endpoints a, b; a tolerance TOL; initial condition α maximum step size $hmax$; minimum step size $hmin$

Ensure: t, w, h where w approximates $y(t)$ and the step size h was used, or a message that the minimum step size was exceeded.

$t = a$; $w = \alpha$; $h = hmax; Flag = 1$;
while $Flag == 1$ **do**
    $k_1 = hf(t, w)$;
    $k_2 = hf(t + \frac{1}{4}h, w + \frac{1}{4}k_1)$;
    $k_3 = hf(t + \frac{3}{8}h, w + \frac{3}{32}k_1 + \frac{9}{32}k_2)$;
    $k_4 = hf(t + \frac{12}{13}h, w + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3)$;
    $k_5 = hf(t + h, w + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4)$;
    $k_6 = hf(t + \frac{1}{2}h, w - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5)$;
    $R = \frac{1}{h}|\frac{1}{360}k_1 - \frac{128}{4275}k_3 - \frac{2197}{75240}k_4 + \frac{1}{50}k_5 + \frac{2}{55}k_6|$ {Note: $R = \frac{1}{h}|\tilde{w}_{i+1} - w_{i+1}|$}
    if $R \leq TOL$ **then**
        $t = t + h$;
        $w = w + \frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5$
        return (t, w, h)
    end if
    $\delta = 0.84(TOL/R)^{\frac{1}{4}}$
    if $\delta \leq 0.1$ **then**
        $h = 0.1h$ {Calculate new h}
    else if $\delta \geq 4$ **then**
        $h = 4h$
    else
        $h = \delta h$
    end if
    if $h > hmax$ **then**
        $h = hmax$
    end if
    if $t \geq b$ **then**
        $Flag = 0$
    else if $t + h > b$ **then**
        $h = b - t$
    else if $h < hmin$ **then**
        $Flag = 0$
        print "minimum h exceeded" {Procedure completed unsuccessfully}
    end if
end while
return (t, w)

2.8 The Adams Variable Step-Size Predictor-Corrector

The Truncation Errors

The Pseudocode

To approximate the solution of the IVP

$$y' = f(t, y), \ \ a \leq t \leq b, \ \ y(a) = \alpha$$

with local truncation error within a given tolerance: [2]

Algorithm 8 :: Adams Variable Step-Size Predictor-Corrector

Require: endpoints a, b; a tolerance TOL; initial condition α maximum step size $hmax$; minimum step size $hmin$

Ensure: t, w, h where w approximates $y(t)$ and the step size h was used, or a message that the minimum step size was exceeded.

{Set up a subalgorithm for the Runge-Kutta 4th-Order method to be called $RK4(h, v_0, x_0, v_1, x_1, v_2, x_2, v_3, x_3)$ that accepts as input a step size h and starting values $v_0 \approx y(t_0)$ and returns $\{(x_j, v_j) | j = 1, 2, 3\}$ defined by the following:}

for $j = 1, 2, 3$ **do**

  $k_1 = hf(x_{j-1}, v_{j-1})$;

  $k_2 = hf(x_{j-1} + \frac{h}{2}, v_{j-1} + \frac{k_1}{2})$;

  $k_3 = hf(x_{j-1} + \frac{h}{2}, v_{j-1} + \frac{k_2}{2})$;

  $k_4 = hf(x_{j-1} + h, v_{j-1} + k_3)$;

  $K = k_1 + 2k_2 + 2k_3 + k_4$

  $v_j = v_{j-1} + \frac{K}{6}$;

  $x_j = x_0 + jh$

end for

$t_0 = a$;

$w_0 = \alpha$;

$h = hmax$;

$Flag = 1$ {$Flag$ will be used to exit a loop.}

$Last = 0$ {$Last$ will indicate when the last value is calculated.}

Call $RK4(h, v_0, x_0, v_1, x_1, v_2, x_2, v_3, x_3)$

$Nflag = 1$;   {Indicates computation from $RK4$}

$i = 4$;

$t = t_3 + h$

while $Flag == 1$ **do**

  $w_p = w_{i-1} + \frac{h}{24}[55f(t_{i-1}, w_{i-1}) - 59f(t_{i-2}, w_{i-2}) + 37f(t_{i-3}, w_{i-3}) - 9f(t_{i-4}, w_{i-4});]$ {Predict w_i}

  $w_c = w_{i-1} + \frac{h}{24}[9f(t, w_p) + 19f(t_{i-1}, w_{i-1}) - 5f(t_{i-2}, w_{i-2}) + f(t_{i-3}, w_{i-3});]$ {Correct w_i}

  $\sigma = 19|w_c - w_p|/270h$

  $\vdots$

  $\vdots$ {next page}

  $\vdots$

end while

Algorithm 9 :: Adams Variable Step-Size Predictor-Corrector Cont'd

if $\sigma \leq TOL$ **then**
  $w_i = w_c$ {Result accepted}
  $t_i = t$
  if $Nflag == 1$ **then**
    for $j = i-3, i-2, i-1, i$ **do**
      return (j, t_j, w_j, h); {Previous results also accepted}
    end for
  else
    return (i, t_i, w_i, h) {Previous results already accepted}
  end if
  if $Last == 1$ **then**
    $Flag = 0$
  else
    $i = i + 1$
    $Nflag = 0$
    if $\sigma \leq 0.1TOL$ **or** $t_{i-1} + h > b$ **then**
      $q = (TOL/2\sigma)^{\frac{1}{4}}$
      if $q > 4$ **then**
        $h = 4h$
      else
        $h = qh$
      end if
      if $h > hmax$ **then**
        $h = hmax$
      end if
      if $t_{i-1} + 4h > b$ **then**
        $h = (b - t_{i-1})/4$
        $Last = 1$
      end if
      Call $RK4(h, w_{i-1}, t_{i-1}, w_i, t_i, w_{i+1}, t_{i+1}, w_{i+2}, t_{i+2})$
      $Nflag = 1$
      $i = i + 3$ {True branch completed}
    end if
  end if
else
  $q = (TOL/2\sigma)^{\frac{1}{4}}$ {False branch result rejected}
  $\vdots$ {next page}
end if
$t = t_{i-1} + h$

Algorithm 20 :: Adams Variable Step-Size Predictor-Corrector Cont'd Part 2

```
if q < 0.1 then
    h = 0.1h
else
    h = qh
end if
if h < hmin then
    Flag = 0
    print "hmin exceeded"
else
    if Nflag == 1 then
        i = i − 3    {Previous results also rejected}
        Call RK4(h, w_{i−1}, t_{i−1}, w_i, t_i, w_{i+1}, t_{i+1}, w_{i+2}, t_{i+2})
        i = i + 3
        Nflag = 1
    end if
end if
```

Chapter 3

NUMERICAL EXPERIMENTS

The above methods were implemented on a ubuntu 22.10 system with python 3.10.7. To demonstrate the performance of the methods mentioned above, we'll consider the IVP

$$f(t, y(t)) = \begin{bmatrix} f_1(t, y_1, y_2) \\ f_2(t, y_1, y_2) \end{bmatrix} = \begin{bmatrix} -4y_1 + 3y_2 - 6 \\ -2.4y_1 + 1.6y_2 + 3.6 \end{bmatrix} \tag{3.1}$$

and initial value $y(0) = 0$.
The true solution to (3.1) is

$$\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} -3.375e^{-2t} + 1.875e^{-0.4t} + 1.5 \\ -2.25e^{-2t} + 2.25e^{-0.4t} \end{bmatrix} \tag{3.2}$$

Figure 3.1: Python implementation of (3.1) and (3.2)

```
def f(t,y):
    y1 = y[0]
    y2 = y[1]

    dy1 = -4 * y1 + 3 * y2 + 6
    dy2 = -2.4 * y1 + 1.6 * y2 + 3.6
    dy = np.array([dy1, dy2])

    return dy

def true_f(t):
    from numpy import exp

    yt1 = -3.375*exp(-2*t) + 1.875*exp(-0.4*t) + 1.5
    yt2 = -2.25*exp(-2*t) + 2.25*exp(-0.4*t)

    return (yt1, yt2)
```

Table 3.1: Comparing Actual values to predicted Euler values

	Actual y_i		Pred. Euler y_i		Compt. Errors	
t	y1_actual	y2_actual	y1_euler	y2_euler	y1_a - y1_e	y2_a - y2_e
0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.1	0.5383	0.3196	0.6	0.36	0.0617	0.0404
0.2	0.9685	0.5688	1.068	0.6336	0.0995	0.0648
0.3	1.3107	0.7607	1.4309	0.8387	0.1201	0.0779
0.4	1.5813	0.9063	1.7101	0.9894	0.1288	0.0831
0.5	1.7935	1.0144	1.9229	1.0973	0.1294	0.0829
0.6	1.9584	1.0922	2.0829	1.1714	0.1245	0.0792
0.7	2.0848	1.1457	2.2012	1.2189	0.1163	0.0732
0.8	2.1801	1.1796	2.2864	1.2456	0.1062	0.0661
0.9	2.2503	1.1978	2.3455	1.2562	0.0953	0.0584
1.0	2.3001	1.2037	2.3842	1.2543	0.0841	0.0506

Table 3.2: Comparing Actual values to predicted Modified Euler values

	Actual y_i		Pred. Mod. Euler y_i		Compt. Errors	
t	y1_actual	y2_actual	y1_me	y2_me	y1_a - y1_me	y2_a - y2_me
0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.1	0.5383	0.3196	0.534	0.3168	0.0043	0.0028
0.2	0.9685	0.5688	0.9615	0.5642	0.007	0.0046
0.3	1.3107	0.7607	1.3022	0.7551	0.0086	0.0057
0.4	1.5813	0.9063	1.5719	0.9001	0.0094	0.0062
0.5	1.7935	1.0144	1.784	1.0081	0.0096	0.0063
0.6	1.9584	1.0922	1.949	1.086	0.0094	0.0062
0.7	2.0848	1.1457	2.0759	1.1398	0.009	0.0059
0.8	2.1801	1.1796	2.1718	1.174	0.0084	0.0055
0.9	2.2503	1.1978	2.2426	1.1928	0.0077	0.0051
1.0	2.3001	1.2037	2.2931	1.1991	0.007	0.0046

Table 3.3: Comparing Actual values to predicted RK2 values

	Actual y_i		Pred. RK2 y_i		Compt. Errors	
t	y1_actual	y2_actual	y1_rk2	y2_rk2	y1_a - y1_rk2	y2_a - y2_rk2
0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.1	0.5383	0.3196	0.534	0.3168	0.0043	0.0028
0.2	0.9685	0.5688	0.9615	0.5642	0.007	0.0046
0.3	1.3107	0.7607	1.3022	0.7551	0.0086	0.0057
0.4	1.5813	0.9063	1.5719	0.9001	0.0094	0.0062
0.5	1.7935	1.0144	1.784	1.0081	0.0096	0.0063
0.6	1.9584	1.0922	1.949	1.086	0.0094	0.0062
0.7	2.0848	1.1457	2.0759	1.1398	0.009	0.0059
0.8	2.1801	1.1796	2.1718	1.174	0.0084	0.0055
0.9	2.2503	1.1978	2.2426	1.1928	0.0077	0.0051
1.0	2.3001	1.2037	2.2931	1.1991	0.007	0.0046

Table 3.4: Comparing Actual values to predicted RK4 values

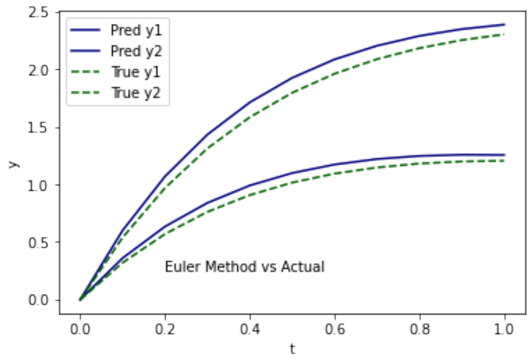
	Actual y_i		Pred. RK4 y_i		Compt. Errors	
t	y1_actual	y2_actual	y1_rk4	y2_rk4	y1_a - y1_rk4	y2_a - y2_rk4
0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.1	0.5383	0.3196	0.5383	0.3196	1e-05	1e-05
0.2	0.9685	0.5688	0.9685	0.5688	1e-05	1e-05
0.3	1.3107	0.7607	1.3107	0.7607	2e-05	1e-05
0.4	1.5813	0.9063	1.5813	0.9063	2e-05	1e-05
0.5	1.7935	1.0144	1.7935	1.0144	2e-05	1e-05
0.6	1.9584	1.0922	1.9584	1.0922	2e-05	1e-05
0.7	2.0848	1.1457	2.0848	1.1457	2e-05	1e-05
0.8	2.1801	1.1796	2.1801	1.1796	2e-05	1e-05
0.9	2.2503	1.1978	2.2502	1.1978	2e-05	1e-05
1.0	2.3001	1.2037	2.3001	1.2037	1e-05	1e-05

Table 3.5: Comparing Actual values to predicted Adams Explicit values

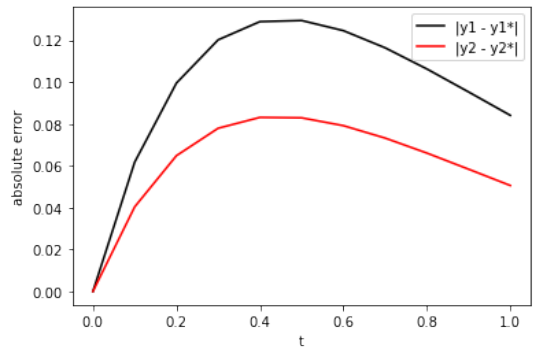
	Actual y_i		Pred. Adams exp. y_i		Compt. Errors	
t	y1_actual	y2_actual	y1_ae4	y2_ae4	y1_a - y1_ae4	y2_a - y2_ae4
0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.1	0.5383	0.3196	0.5383	0.3196	1e-05	1e-05
0.2	0.9685	0.5688	0.9685	0.5688	1e-05	1e-05
0.3	1.3107	0.7607	1.3107	0.7607	2e-05	1e-05
0.4	1.5813	0.9063	1.581	0.9062	0.00027	0.00018
0.5	1.7935	1.0144	1.7932	1.0142	0.00036	0.00024
0.6	1.9584	1.0922	1.9579	1.0919	0.0005	0.00033
0.7	2.0848	1.1457	2.0843	1.1453	0.00051	0.00034
0.8	2.1801	1.1796	2.1796	1.1792	0.00054	0.00036
0.9	2.2503	1.1978	2.2497	1.1975	0.00051	0.00034
1.0	2.3001	1.2037	2.2996	1.2034	0.0005	0.00033

Table 3.6: Comparing Actual values to predicted Adams Predictor-Corrector values

	Actual y_i		Pred. Pred.-Corr. y_i		Compt. Errors	
t	y1_actual	y2_actual	y1_pc4	y2_pc4	y1_a - y1_pc4	y2_a - y2_pc4
0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.1	0.5383	0.3196	0.5383	0.3196	1e-05	1e-05
0.2	0.9685	0.5688	0.9685	0.5688	1e-05	1e-05
0.3	1.3107	0.7607	1.3107	0.7607	2e-05	1e-05
0.4	1.5813	0.9063	1.5813	0.9063	2e-05	1e-05
0.5	1.7935	1.0144	1.7936	1.0144	5e-05	3e-05
0.6	1.9584	1.0922	1.9585	1.0923	6e-05	4e-05
0.7	2.0848	1.1457	2.0849	1.1457	7e-05	5e-05
0.8	2.1801	1.1796	2.1802	1.1796	7e-05	5e-05
0.9	2.2503	1.1978	2.2503	1.1979	7e-05	5e-05
1.0	2.3001	1.2037	2.3002	1.2038	7e-05	5e-05

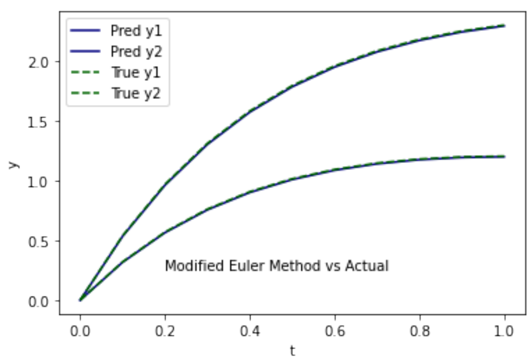


(a) Approximated y_i

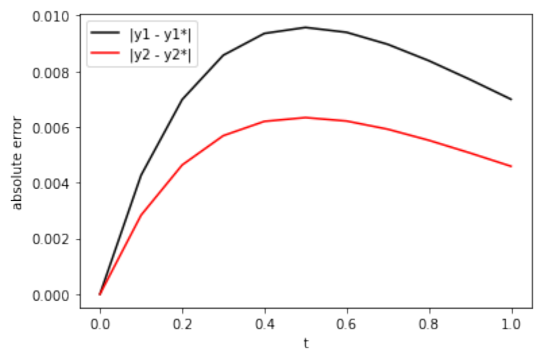


(b) Errors

Figure 3.2: Approximation $y(t)$ obtained by the Euler's Method

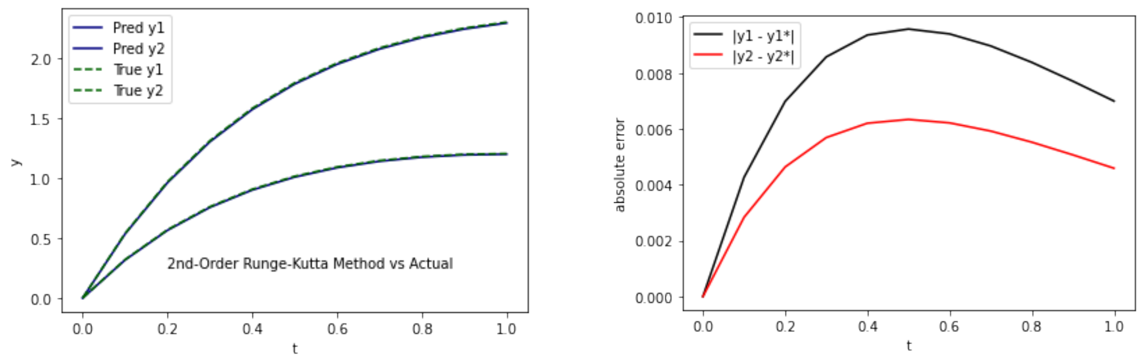


(a) Approximated y_i



(b) Errors

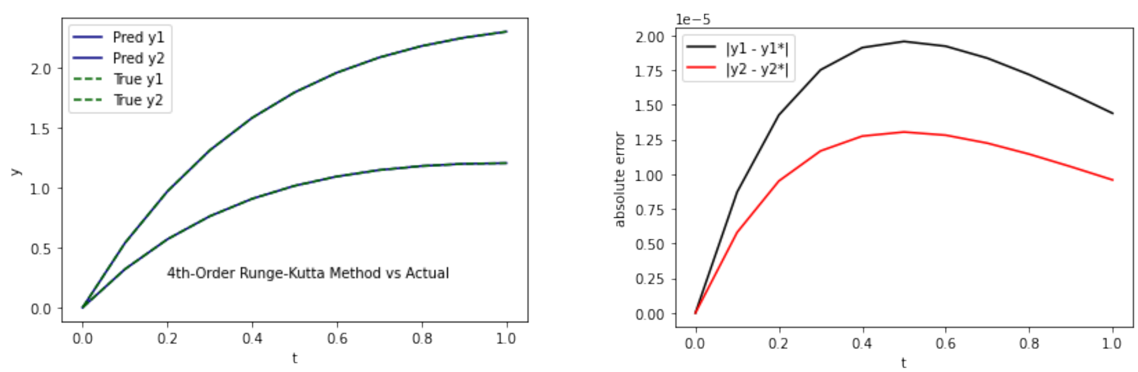
Figure 3.3: Approximation $y(t)$ obtained by the Modified Euler's Method



(a) Approximated y_i

(b) Errors

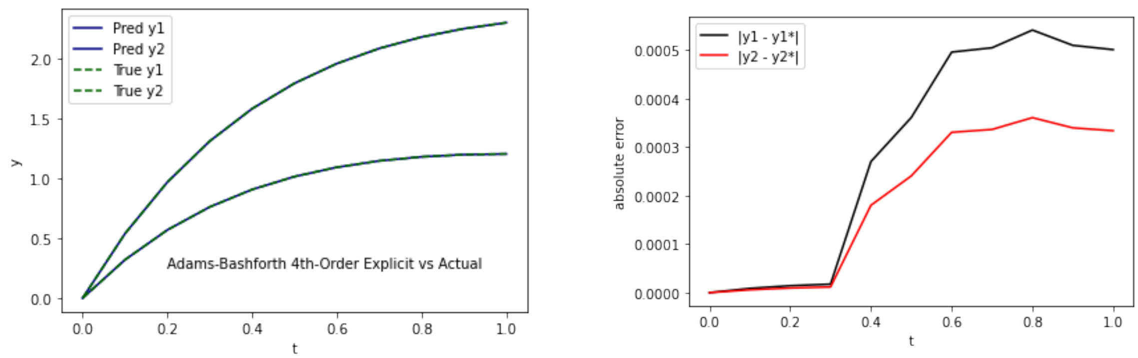
Figure 3.4: Approximation $y(t)$ obtained by the RK2 Method



(a) Approximated y_i

(b) Errors

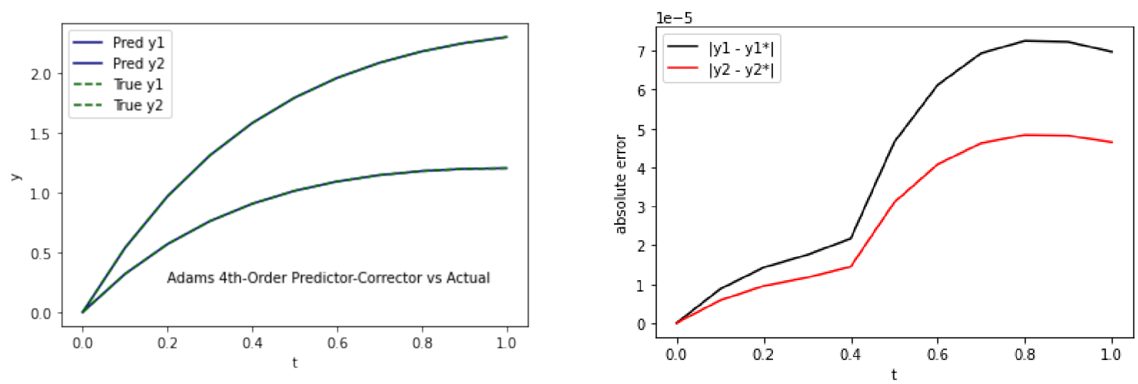
Figure 3.5: Approximation $y(t)$ obtained by the RK4 Method



(a) Approximated y_i

(b) Errors

Figure 3.6: Approximation $y(t)$ obtained by the Adam Explicit Method



(a) Approximated y_i

(b) Errors

Figure 3.7: Approximation $y(t)$ obtained by the Predictor-Corrector Method

From my python implementation of all methods discussed in **Chapter 2**. The tables show the timestamps where each y_i approximation is computed and also shows the actual y_i values obtained from (3.2). The errors at each timestamp is computed and plotted (right) as well. From the tables and error plots we can clearly see that the Euler is the least accurate method for approximating our IVP (3.1) this is due to the order of error as discussed above. The error of Euler Method accumulates over time thus methods like the Modified Euler is preferred. However, from our numerical experiment we can observe that although the Modified Euler may be a better option compared to the Euler's it also has large errors compared to Runge-Kutta Methods. RK4 has the best approximation compared to the earlier methods thus it is the most used method in approximation IVP.
As discussed most of our earlier methods only makes one prediction and that is it; the Adams Predictor-Corrector method we discussed, like the name says, first approximates the function with an explicit method and corrects the approximation with an implicit method. Observing the error plot generated by the Adams Predictor Corrector method, the error at each timestamp is lesser and hence more accurate approximations are produced by this method.

REFERENCES

K. Atkinson. *An introduction to numerical analysis*. John wiley & sons, 1991.

R. L. Burden, J. D. Faires, and A. M. Burden. *Numerical analysis*. Cengage learning, 2015.

J. C. Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.

W.-f. Ch'en, D. De Kee, and P. N. Kaloni. Advanced mathematics for engineering and science. *(No Title)*.

P. Dawkins. Differential equations. URL `https://tutorial.math.lamar.edu/classes/de/eulersmethod.aspx`. Accessed: 04-15-2023.

L. Euler. *Institutiones Calculi Integralis 2nd Part*, volume 1. Springer Science & Business Media, 1913.

Geek4Geeks. Runge-kutta 2nd order method to solve differential equations. URL `https://www.geeksforgeeks.org/runge-kutta-2nd-order-method-to-solve-differential-equations`. Accessed: 04-15-2023.

D. W. Harder. Runge kutta fehlberg. URL `https://ece.uwaterloo.ca/~dwharder/NumericalAnalysis/14IVPs/rkf45/complete.html`. Accessed: 04-15-2023.

K. King. How to solve initial value problems. URL `https://www.kristakingmath.com/blog/solving-initial-value-problems`. Accessed: 04-15-2023.

M. Mahooti. Runge-kutta-fehlberg (rkf45), 2023. URL `https://www.mathworks.com/matlabcentral/fileexchange/73881-runge-kutta-fehlberg-rkf45`. Retrieved April 15, 2023.

R. Melton. Initial-value problems. URL `https://courses.lumenlearning.com/calculus2/chapter/initial-value-problems`. Accessed: 04-15-2023.

wikipedia.org. Initial value problem, a. URL `https://en.wikipedia.org/wiki/Euler_method`. Accessed: 04-15-2023.

wikipedia.org. Heun's method, b. URL https://en.wikipedia.org/wiki/Heun%27s_method. Accessed: 04-15-2023.

wikipedia.org. Initial value problem, c. URL https://en.wikipedia.org/wiki/Initial_value_problem. Accessed: 04-15-2023.

wikipedia.org. Runge–kutta methods, d. URL https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods. Accessed: 04-15-2023.