# Numerical-solver
# Assignment Report

Hui Jia Farm

March 15, 2021

# Contents

1. Explain finite difference methods. 2. Derive the numerical methods and their truncation error (show error behaviour).

Methods included: 1. One-step method a. Euler's explicit b. Euler's implicit c. 4-stage Runge-Kutta method d. trapezium rule remarks: fixed point iteration method included to compute implicit methods 2. Predictor-corrector method a. Euler-trapezium method 3. Adaptive method a. ode23 method (Matlab) b. ode45 method (Matlab)

# Chapter 1

# Numerical methods

In real world problems, ODE systems are frequently used. However, these models cannot be solved analytically. Therefore, the solution has to be estimated by numerical methods. The most popular and simple method is the Euler's method.

Intuitively, the Euler's explicit method tries to estimate the value at the next step following the gradient of the solution at current point. If the step size is sufficiently small, the estimation will be accurate.

An initial value problem, has the general form of

$$y' = f(x, y) \tag{1.1}$$

$$y(x_0) = y_0 \tag{1.2}$$

for $x \in [x_0, X_M]$.

Notation: Throughout the report, we will use the following notation. $y_n$ - numerical approximation of $y(x_n)$ $y(x_n)$ - analytical solution at mesh point $x_n$ $x_n$ - mesh points of defined range, where

$$x_n = x_0 + nh \tag{1.3}$$

$$h = \frac{(X_M - x_0)}{N} \tag{1.4}$$

for $n = 0, \ldots, N$

## 1.1 One-step methods

For the simple Euler's explicit method,

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{1.5}$$

The implementation is as follows:

```
y_n = [self.initial_value]
x_n = [self.x_min]

# Calculate approximated solution for each mesh point.
for n in range(1, self.mesh_points + 1):
    step = [self.mesh_size * f for f in self.func(x_n[-1], y_n
    [-1])]
    y_n.append([a + b for a, b in zip(y_n[-1], step)])
    x_n.append(self.x_min + n * self.mesh_size)

return x_n, y_n
```

The truncation error is defined to be the difference of exact solution with the numerical solution given the exact solution of previous mesh point is known. Therefore, we have that the truncation error for Euler's explicit method to be

$$T_n := \frac{y(x_{n+1}) - y(x_n)}{h} - f(x_n, y(x_n)) \tag{1.6}$$

According to Taylor's series expansion, we have

$$y(x_n + h) = y(x_n) + hy'(x_n) + \frac{1}{2}h^2 y''(\xi_n) \tag{1.7}$$

for $\xi_n \in (x_n, x_{n+1})$. Substitute this to the truncation error, noting that $f(x_n, y(x_n)) = y'(x_n)$ we get

$$T_n = \frac{1}{2}h^2 y''(\xi_n) \tag{1.8}$$

Therefore, the truncation error for Euler's explicit method varies linearly with the step size.

In this report, we will use an example model

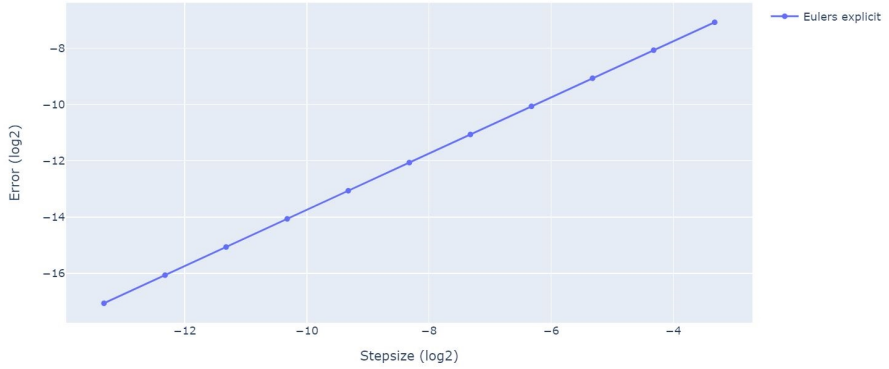
$$f(x, y) = -y \tag{1.9}$$

$$y(x_0) = 1 \tag{1.10}$$

for $x \in [0, 5]$ to show that the implementation follows the theory.

Some examples of the solution to the given model can be found here: Example model notebook

Result image extracted from the notebook above shows that the truncation error follows




Other than the Euler's explicit method, the other one-step methods implemented are the Euler's implicit method, trapezium rule method and four-stage explicit Runge-Kutta method.

The Euler's implicit method is defined to be

$$y_{n+1} = y_n + hf(x_{n+1}, y_{n+1}) \tag{1.11}$$

, while the trapezium rule method is

$$y_{n+1} = y_n + \frac{1}{2}h[f(x_n, y_n) + f(x_{n+1}, y_{n+1})] \tag{1.12}$$

. These methods, including Euler's explicit method, Euler's implicit method and trapezium rule method, can be generalised under the $\theta$-method,

$$y_{n+1} = y_n + h[(1-\theta)f(x_n, y_n) + \theta f(x_{n+1}, y_{n+1})] \tag{1.13}$$

. For $\theta = 0$ and $\theta = 1$, the methods are Euler's explicit and Euler's implicit method respectively. Using the same definition for truncation error in 1.6, we can see that the truncation error are $T_n = -\frac{1}{2}hy''(\xi_n)$ for $\xi_n \in (x_n, x_{n+1})$ and $T_n = -\frac{1}{12}h^2y^{(3)}(\xi_n)$ for $\xi_n \in (x_n, x_{n+1})$ respectively.

The Euler's implicit method implementation,

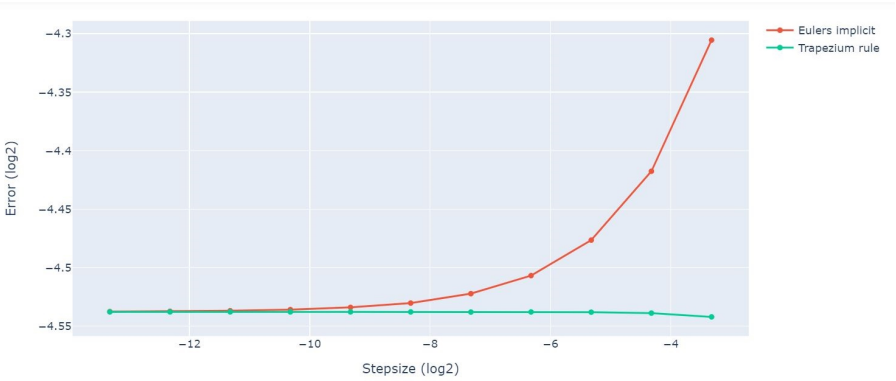
```
y_n = [self.initial_value]
x_n = [self.x_min]

# Use value at previous mesh point as prediction for
# fixed point iteration if no prediction is given.
if prediction is None:
    prediction = y_n[-1]

# Calculate approximated solution for each mesh point.
# Use fixed point iteration to solve numerical equation.
for n in range(1, self.mesh_points + 1):

    def num_method(prediction):
        step = [self.mesh_size * f for f in self.func(
            x_n[-1] + self.mesh_size, prediction)]
        return [a + b for a, b in zip(y_n[-1], step)]

    est_y = self.fixed_pt_iteration(prediction, num_method)
    y_n.append(est_y)
    x_n.append(self.x_min + n * self.mesh_size)

return x_n, y_n
```

The trapezium rule method implementation,

```
y_n = [self.initial_value]
x_n = [self.x_min]

# Use value at previous mesh point as prediction for
# fixed point iteration if no prediction is given.
if prediction is None:
    prediction = y_n[-1]

# Calculate approximated solution for each mesh point.
# Use fixed point iteration to solve numerical equation.
for n in range(1, self.mesh_points + 1):

    def num_method(prediction):
        previous_func = self.func(x_n[-1], y_n[-1])
        new_func = self.func(x_n[-1] + self.mesh_size,
    prediction)
        return [a + self.mesh_size / 2 * (b + c) for a, b, c in
     zip(
                y_n[-1], previous_func, new_func)]

    est_y = self.fixed_pt_iteration(prediction, num_method)
    y_n.append(est_y)
    x_n.append(self.x_min + n * self.mesh_size)

return x_n, y_n
```

When these methods are tested on the example model 1.9, the truncation error does not behave as expected. This is probably due to the use of fixed

point iteration algorithm to estimate the solution of implicit functions. The error graph is shown below. Moreover, the error of the fixed point iteration algorithm is added to the error of the numerical method. Therefore, the difference of the numerical solution with the exact solution is larger for implicit methods as compared to Euler's explicit methods.




## 1.2 Predictor-corrector methods

For the implicit methods, numerical value at previous mesh point is chosen as the initial value for the fixed point iteration algorithm. The predictor-corrector method suggests a more carefully chosen initial guess for the implicit methods. An explicit numerical method is used as a predictor of the initial value of an implicit method. The initial value is then used as an initial guess for iterations to solve an implicit function. The implicit method that refines the solution is known as the corrector method.

The Euler-Trapezoidal method is a predictor-corrector that uses an Euler's explicit method as the predictor and trapezium rule method as the corrector. In this implementation, the trapezium rule method corrector is iterated until a set of conditions are satisfied. The conditions are defined to be the difference between current iteration and previous iteration is lesser than a given threshold value or the number of iterations exceeds a certain amount.

## 1.3 Adaptive method

In some types of problem or model, the solution to the problem exhibits a behaviour where step sizes have to be sufficiently small for a stable solution. In other words, as the mesh point varies, the solution have small changes. Such problems are called stiff problems. In order to obtain a stable solution, the computational cost is high. Moreover, such stable solution would have resolutions higher than required for practical purposes.

The adaptive method focus on the achieving desired accuracy with low computational cost. The main idea of an adaptive method is to control the precision at each mesh point. The truncation error at each mesh point is estimated. If the truncation error is larger than a threshold value, a smaller step size is chosen. These steps are repeated until the truncation error is smaller than the given threshold value.

The adaptive method implemented in this software is based on the BS23 (Bogacki and Shampine) and RK45 (Runge-Kutta-Fehlberg) method.

Absolute tolerance and relative tolerance were used in the implementation of the adaptive methods. Therefore two comparisons were made, one based on absolute tolerance and the other on relative tolerance. Similarly, the implemented adaptive method is used to solve the example model 1.9. The adaptive methods were tested for convergence. As shown in the graph below, the error of the method is smaller for smaller tolerance, regardless of absolute tolerance or relative tolerance.