Task Description

- For your own variant of the task realize in your favorite programming language (e.g. Java)
  - ♣ Euler's method,
  - ♣ improved Euler's method,
  - ♣ Runge-Kutta method

  in your own software application.

- Using this application construct corresponding approximation of the solution of a given initial value problem (provide the possibility of changing of the initial conditions).
- Realize the exact solution of an IVP in your application.
- Provide data visualization capability (charts plotting) in the user interface of your application (e.g. using the JavaFX).
- Investigate the convergence of these numerical methods on different grid sizes (provide the possibility of changing of the number of grid steps).
- Compare approximation errors of these methods plotting the corresponding chart for different grid sizes (provide the possibility of changing of the range of grid steps).

Specific Initial Value Problem based on my variant

Differential Equation: $y' = f(x, y) = xy^2 + 3xy$

Initial values: $x_0 = 0$, $y_0 = 3$, i.e. $y(0) = 3$

And, $x \in [x_0; X]$ where $X = 5.5$, so $x \in [0; 5.5]$

Computer Program Implementation Details

I have implemented a JavaScript program to calculate the approximations of solutions of our IVP above using different numerical methods. The program allows any user to change the initial conditions (x_0, y_0), the range/interval in which the approximations are to be calculated $[x_0; X]$, and either the height of each grid step h or the number of grid steps N in the interval.

The program is deployed as a web app and is publicly accessible through the following url:

https://kckusal.github.io/IU-F18_Differential-Equation_Computational-Practicum_Approximating-solutions-numerically/

The original source code is maintained in the following Github repository:

https://github.com/kckusal/IU-F18_Differential-Equation_Computational-Practicum_Approximating-solutions-numerically

Solving the given differential equation analytically

Before proceeding with numerical methods of approximation, let us take an analytic look at the differential equation and try to solve it.

The Initial Value Problem we have is:

$$y' = xy^2 + 3xy; \qquad y(0) = 3; \qquad x \in [0; 5.5]$$

We have a first-order, non-linear (non-constant coefficient) ordinary differential equation. In fact, it is a *Bernoulli differential equation*. This equation is also separable as it can be seen from the following rearrangement:

$$\frac{1}{y^2+3y}\,\frac{dy}{dx} = x$$

We can exploit this feature and apply the method of separation of variables to solve this equation. However, we will proceed to solve this differential equation like any other Bernoulli equation.

Let us consider $v = y^{-1}$ such that like y, v is also a function of x. Then, we have $v' = -y^{-2}y'$.

We now divide the differential equation by y^2 on both sides and solve:

$$y^{-2}y' - 3xy^{-1} = x$$

$$-v' - 3xv = x \qquad [\because \text{ Substitute } v = y^{-1} \ \& -v' = y^{-2}y']$$

$$\frac{v'}{3v+1} = -x$$

Integrating on both sides gives:

$$\int \left(\frac{1}{3v+1}\right) dv = \int (-x)dx$$

$$\ln|3v + 1| = -\frac{3}{2}x^2 + C$$

where C is a ***simplified*** constant defined by the initial conditions of the IVP.

$$v = \frac{e^{-1.5x^2+C}-1}{3}$$

Substituting back $y = v^{-1}$, we get

$$y = \frac{3}{e^{-1.5x^2+C}-1} \qquad \text{.....Equation (1)}$$

This is the general solution to our differential equation. The constant C is found by plugging in the initial values $(x, y) = (x_0, y_0)$ and is given by expression below:

$$C = \ln\left|\frac{3}{y_0} + 1\right| + 1.5x_0^2$$

For this Initial Value Problem with $y(0) = 3$, we have

$$C = \ln\left|\frac{3}{3} + 1\right| + 1.5\,(0)^2 = \ln(2),$$

$$y = \frac{3}{e^{-1.5x^2+\ln(2)}-1} = \frac{3}{2e^{-1.5x^2}-1}$$

Going back to equation (1), we note that the solution is undefined whenever the following occurs:

$$e^{-1.5x^2+C} - 1 = 0$$

$$-1.5x^2 + C = \ln(1) = 0$$

$$x^2 = \frac{C}{1.5}$$

$$x = \pm\sqrt{\frac{C}{1.5}}$$

This means the solutions will have vertical asymptotes at $x = \pm\sqrt{\frac{C}{1.5}}$ which depends on the initial conditions of the IVP as the constant C is defined by them.

- Accordingly, our IVP here will have asymptotes at $x = \pm\sqrt{\frac{\ln(2)}{1.5}} = \pm\,0.679778$.

We notice that the function $y(x)$ is not continuous in the interval we are considering $x \in [0; 5.5]$ as one of the vertical asymptotes $x = 0.679778$ lies in this interval. We can see the graph of our exact solution $y = \frac{3}{2e^{-1.5x^2}-1}$ below:

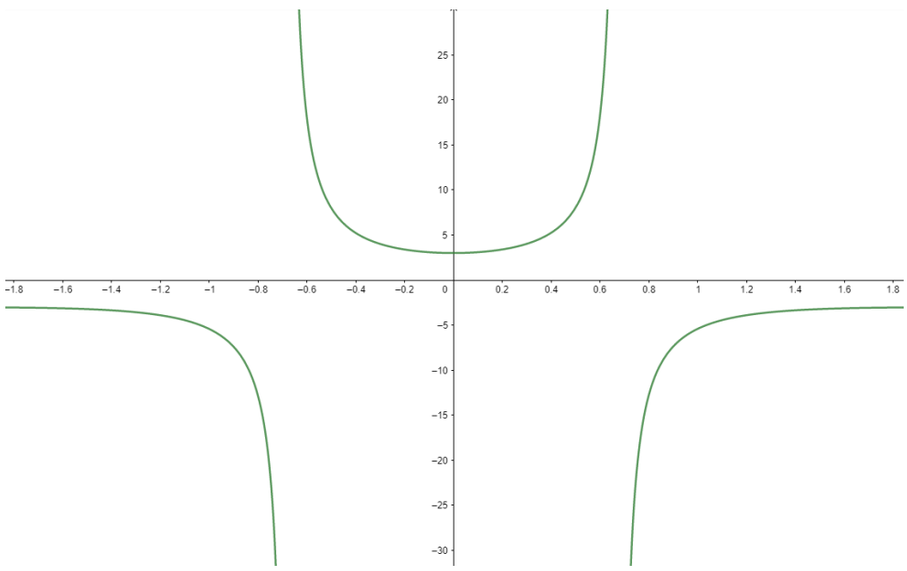

Fig. Graph generated online using https://www.geogebra.org/graphing tool

Since $\lim_{x\to 0.679778} \frac{3}{2e^{-1.5x^2}-1} = \infty$, it is clear that $x = 0.679778$ is a vertical asymptote to our solution.

We will see in a later section how the presence of a vertical asymptote affects our approximations of solutions.

Numerical Approximations using Euler method

The Euler method algorithm for calculating numerical approximations is as follows:

- From (x_n, y_n), the next approximation at a step height h is given by:

$$x_{n+1} = x_n + h,$$
$$y_{n+1} = y_n + h\, f(x_n, y_n),$$

  where f is the derivative function.

The source code implementation as a function in our JavaScript program:

```
// apply Euler method with given f=dy/dx function, initial values (xi, yi),
// each step height h, and number of steps n and return the solution set.
function applyEulerMethod(f, xi, yi, h, n) {
  // initialize solutions with initial values
  let solutions = {
    x: [xi],
    y: [yi]
  };

  for (let j=1; j<=n; j++) {
    yi += h*f(xi, yi);          // calculate next approximation
    xi += h;                    // increment x by step amount

    solutions.x.push(xi);
    solutions.y.push(yi);
  }

  return solutions;
}
```

For our given IVP, the Euler method approximation with grid height h=0.05 produced the following result in interval [0; 5.5]:

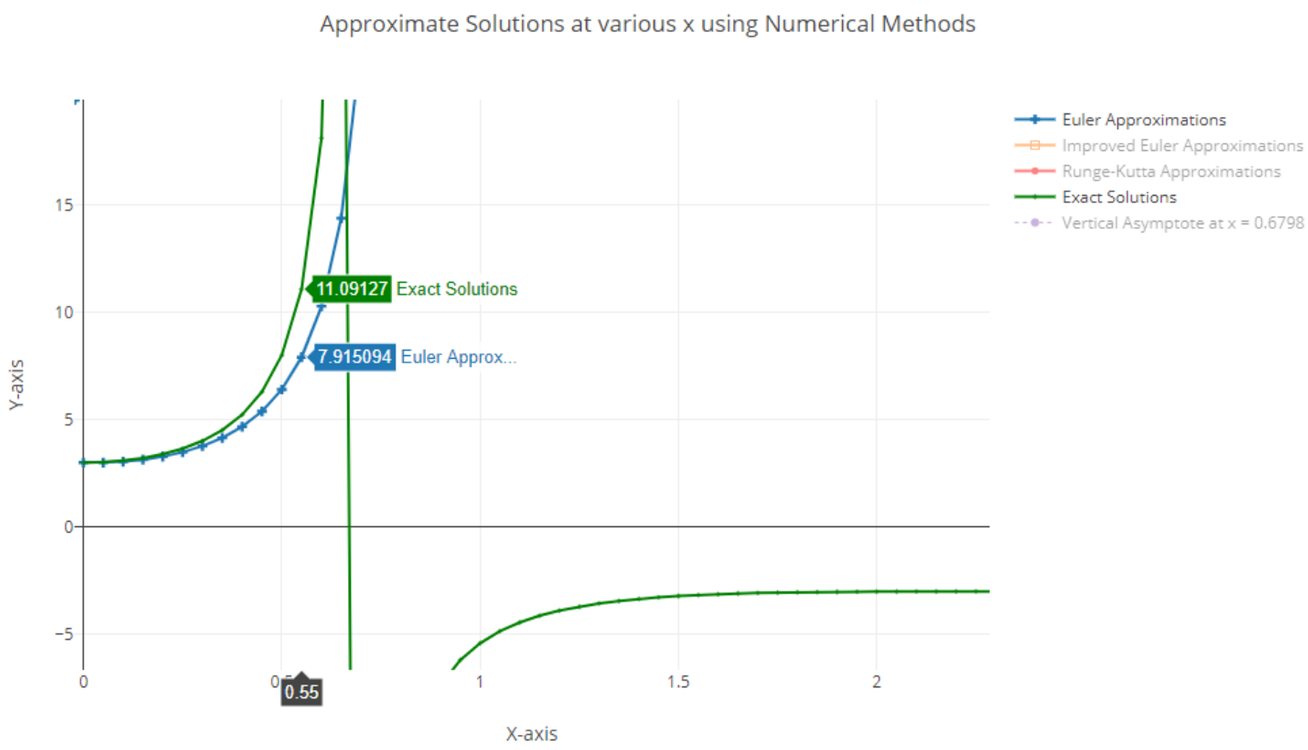

Fig. “Zoomed in” interactive graph to highlight the approximations using Euler Method

Numerical Approximations using Improved Euler method

The Improved Euler method algorithm for calculating numerical approximations is as follows:

- From (x_n, y_n), the next approximation at a step height h is given by:

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

  where f is the derivative function.

The source code implementation as a function in our JavaScript program:

```
// apply Improved Euler method with given f=dy/dx function, initial values
// (xi, yi), step height h, and number of steps n, and return the solution set.
function applyImprovedEulerMethod(f, xi, yi, h, n) {
  // initialize solutions with initial values
  let solutions = {
    x: [xi],
    y: [yi]
  };

  let temp=-1;
  for (let j=1; j<=n; j++) {
    temp = f(xi, yi);
    xi += h;
    yi += 0.5*h*(temp + f(xi, yi+h*temp));

    solutions.x.push(xi);
    solutions.y.push(yi);
  }

  return solutions;
}
```

For our given IVP, the Improved Euler method approximation with grid height h=0.05 produced the following result in interval [0; 5.5]:

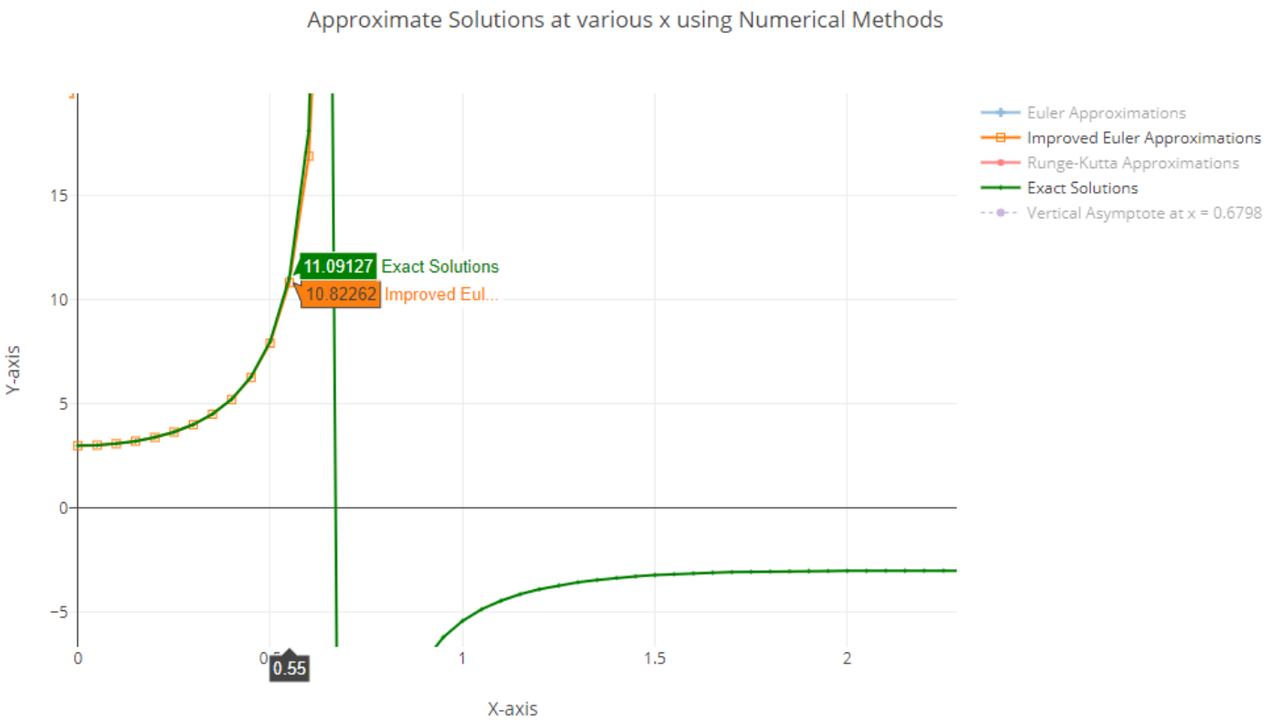

Fig. “Zoomed in” interactive graph to highlight the approximations using Improved Euler Method

Numerical Approximations using 4th Order Runge-Kutta method

The fourth-order Runge Kutta method algorithm for calculating numerical approximations is as follows:

- From (x_n, y_n), the next approximation at a step height h is given by:
  - $k_1 = f(x_n, y_n)$
  - $k_2 = f(x_n + \frac{h}{2},\ y_n + \frac{hk_1}{2})$
  - $k_3 = f(x_n + \frac{h}{2},\ y_n + \frac{hk_2}{2})$
  - $k_4 = f(x_n + h,\ y_n + hk_3)$

$$x_{n+1} = x_n + h,$$
$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$

where f is the derivative function.

The source code implementation in our program is as follows:

```
// Apply fourth order Runge-Kutta method in function dy/dx=f(x,y), given the
// initial values (xi, yi), each step height h, and number of grid steps n.
// Return the approximated solution set.
function applyRungeKuttaMethod(f, xi, yi, h, n) {
  let solutions = {
    x: [xi],
    y: [yi]
  };

  let k1, k2, k3, k4;
  for (let j=1; j<=n; j++) {
    k1 = f(xi, yi);
    k2 = f(xi + 0.5*h, yi + 0.5*h*k1);
    k3 = f(xi + 0.5*h, yi + 0.5*h*k2);
    k4 = f(xi + h, yi + h*k3);

    xi += h;
    yi += (k1 + 2*k2 + 2*k3 + k4)*h/6;

    solutions.x.push(xi);
    solutions.y.push(yi);
  }

  return solutions;
}
```

For our given IVP, the fourth order Runge-Kutta method of approximation with grid height h=0.05 produced the following result in interval [0; 5.5]:

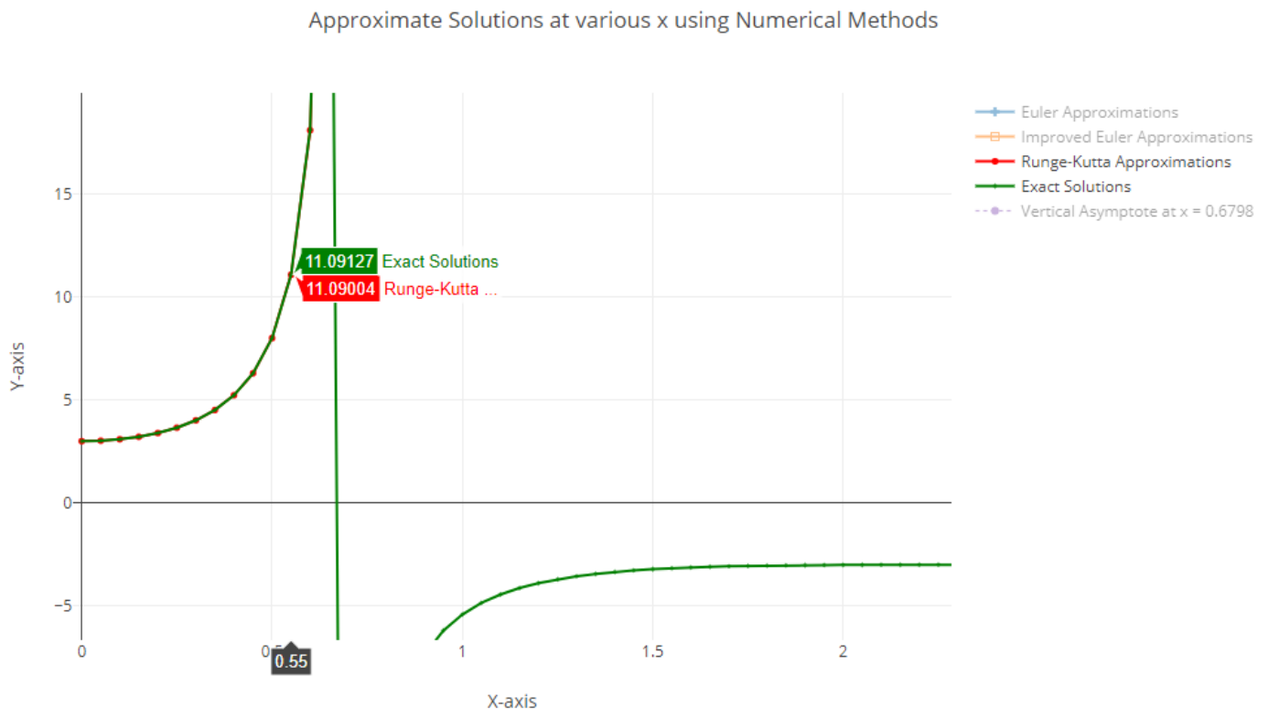

Fig. “Zoomed in” interactive graph to highlight the approximations using Runge-Kutta Method

Convergence of Numerical Methods of Approximation

The following is the overall comparison graph of approximations plotted in contrast with exact solutions for a chosen step height h=0.05 (N=110) in the interval [0; 5.5]:

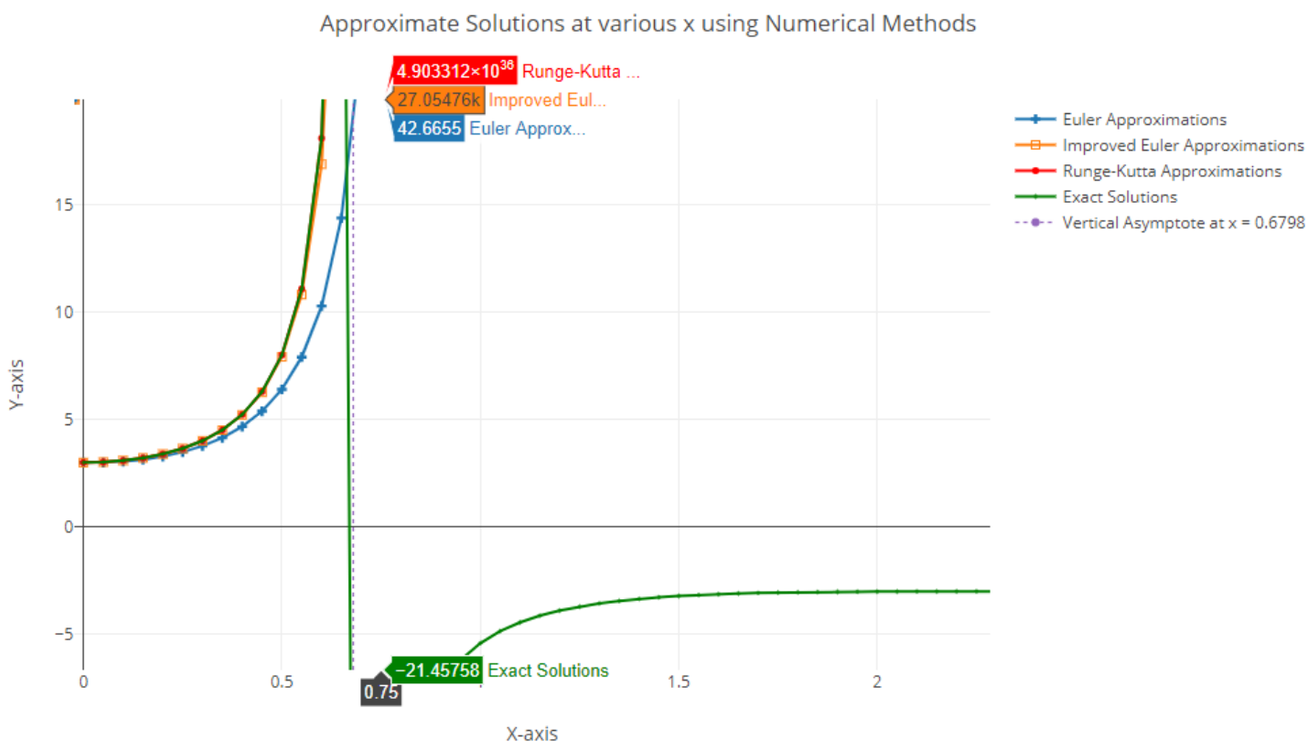

Fig. “Zoomed in” interactive graph comparing approximations produced by different methods.

General Observation

Note that the solution is not convergent over the range [0, 5.5], it instead **converges over the range** [0, 0.679778).

(Wolfram reference for failure to converge over the range [0; 5.5], https://www.wolframalpha.com/input/?i=Runge-Kutta+method,+dy%2Fdx+%3D+x*y*y%2B3*x*y,+y(0)+%3D+3,+from+0+to+5.5,+h+%3D+.05).

Because our solution has a vertical asymptote at x=0.679778, we'd have to say the solutions "blows up" at this point. We say that the solution exists locally in any interval to the left of x = 0.679778, but we don't have global existence

(Reference to solutions blowing up at a point, Example 13, https://ocw.mit.edu/courses/mathematics/18-330-introduction-to-numerical-analysis-spring-2012/lecture-notes/MIT18_330S12_Chapter5.pdf).

We notice that the approximate solutions are closer to exact solutions when the step height h is chosen smaller (i.e. the number of steps, N, is larger) as seen from following series of graph and one we already have above (use program for more):

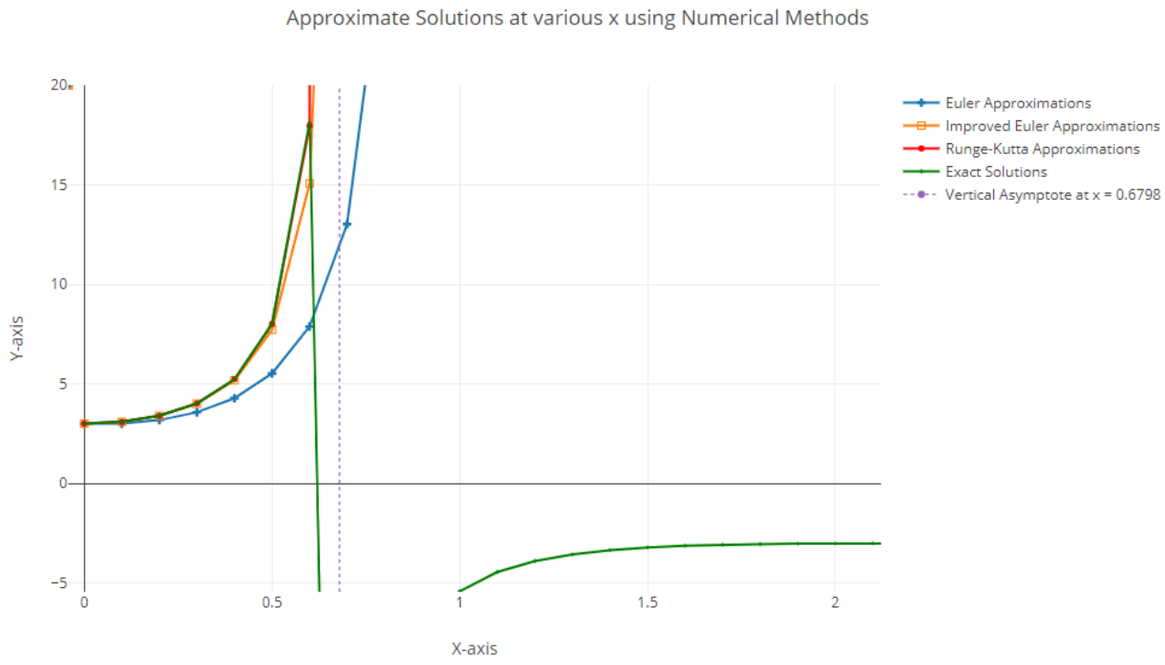

Fig. Graph comparing approximations produced by different methods with h=0.1 (N=55).

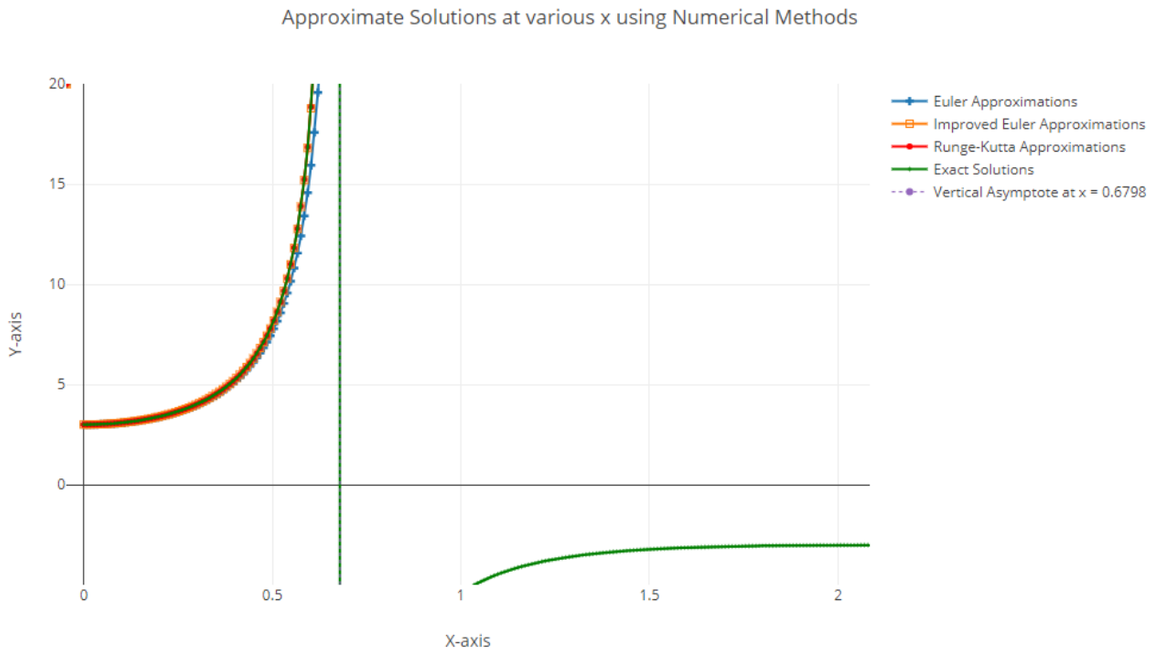

Fig. Graph comparing approximations produced by different methods with h=0.009 (N=611).

Hence, as we can see, the **solutions converge better** when the step height h is chosen smaller (i.e. number of steps N is larger).

We have to carefully chose h: if h is too small, it will take too long to compute solutions; similarly, if h is large enough depending on the interval under consideration, the *solutions overshoot the asymptote* losing accuracy as is the case in above graph when h=0.1 (see in the graph how the Euler method produces solution beyond the asymptote). In other words, if we take h to be too large, the solution "blows up" quickly, even though the Euler method, for instance, is consistent.

Approximation Errors at various x using Numerical Methods

The following is the overall comparison graph of approximation errors at various values of x using these numerical methods plotted in contrast with exact solutions for a chosen step height h=0.05 (N=110) in the interval [0; 5.5]:

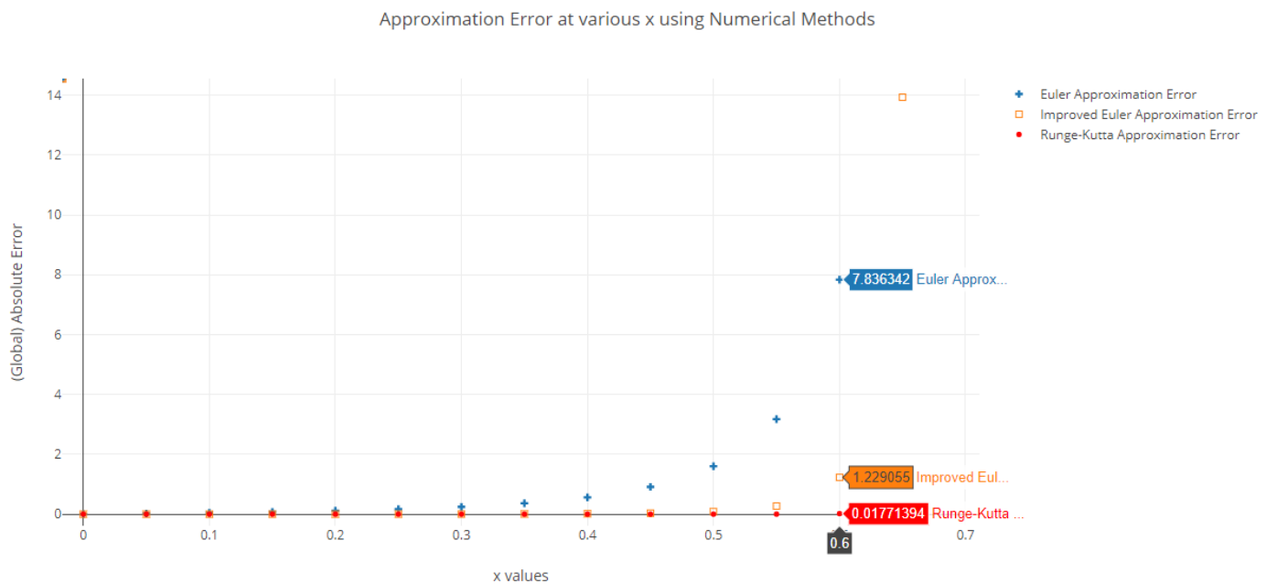

Fig. "Zoomed in" interactive graph comparing different methods' approximation errors when h=0.05.

Let us for the sake of comparison also see how these approximation errors vary for two further values of h, h=0.1 and h=0.009 whose approximation graphs are in the previous sections:

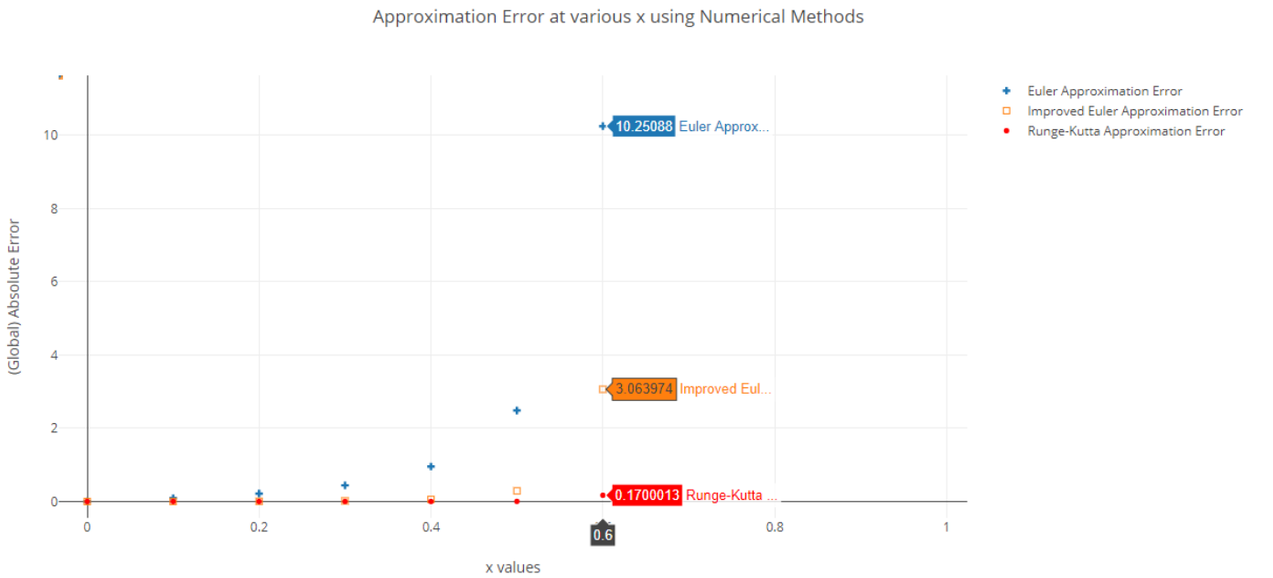

Fig. "Zoomed in" interactive graph comparing different methods' approximation errors when h=0.1.

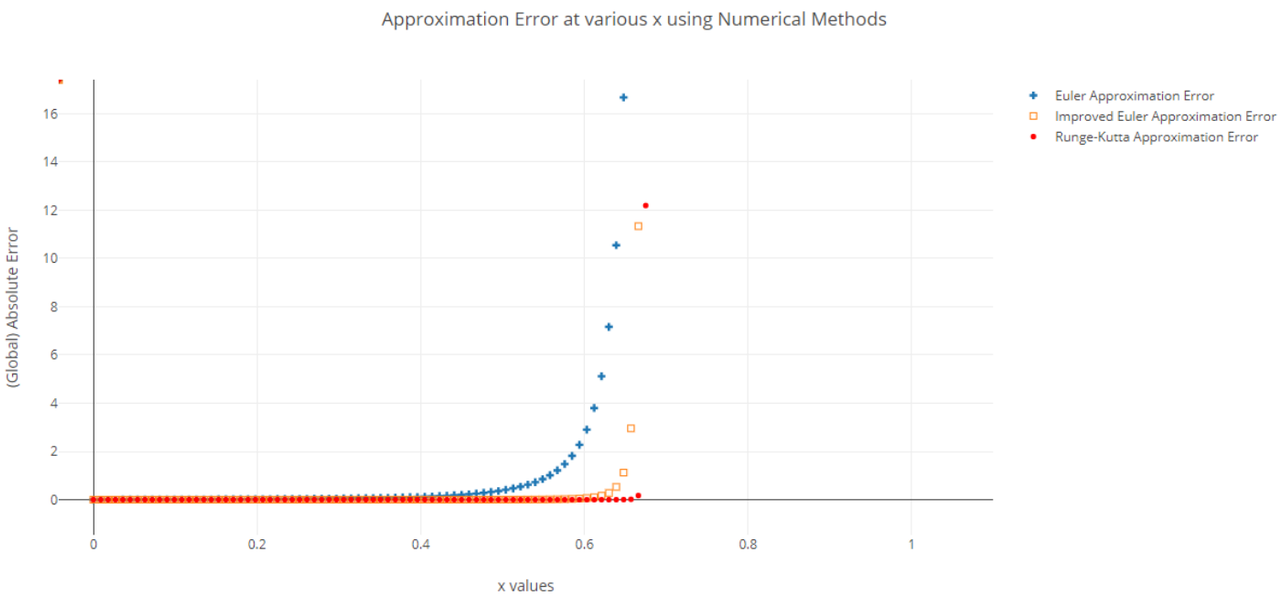

Fig. “Zoomed in” interactive graph comparing different methods’ approximation errors when h=0.009.

General Observation

We will be dealing with two types of errors in our approximations: local errors and global errors. But since convergence is the study of global error, we limit ourselves with the observations originating from global error in approximations. Also, since our program is capable of representing the numeric values during calculations with large number of digits, we will ignore any “truncation” or “round off” errors as well.

(Reference to convergence as study of global error: Section 5.2.1 – Convergence, https://ocw.mit.edu/courses/mathematics/18-330-introduction-to-numerical-analysis-spring-2012/lecture-notes/MIT18_330S12_Chapter5.pdf)

Let us start by noting that the less the absolute error, the better our approximation is (it is closer to the exact value). From the above graphs with h=0.05, h=0.1, and h=0.009, we can see that the errors are high in general for larger step height h (or smaller number of steps, n).

Also, note that approximations can be seen to get worse (error increasing) as x increases. This behavior is fairly common in these approximations and is as expected. Each successive approximation is found using a previous approximation. Therefore, at each step we introduce error and so approximations should, in general, get worse as x increases.

Since our exact solution has an asymptote in the interval under consideration, there is one more observation to make. Notice that the approximation is worst where the function is changing rapidly (error is relatively very high near asymptote). This explains why the errors shoot up as the approximations get close to x=0.679778. And also, since the solutions blow up at this point, we are unable to approximate solutions past this and the errors cannot be determined.

Overall, in comparison to the various methods we used for approximations, we can see that the absolute error is higher with Euler Method (worst approximation) and is very low with Runge-Kutta Method (best approximation).