Introduction

There are many ways to solve differential equations of various kinds, but they can be very difficult to solve using analytical methods. Then we can use numerical methods that allow us to calculate the values of the function close to the present.

In the practicum, we will consider the following numerical methods:

- Euler's method
- Improved Euler's method
- Runge-Kutta method

Goals

Compare the result of calculations of numerical methods with the results of an analytical solution. Determine the most accurate method.

Tasks

1. Find the exact solution for the given initial value problem.
2. For the given IVP of the task implement in C# programming language:
   - Euler's method,
   - Improved Euler's method,
   - Runge-Kutta method
3. Implement the exact solution of an IVP in application.
4. Provide data visualization capability (charts plotting) in the user interface.
5. Investigate the convergence of these numerical methods on different grid sizes (provide the possibility of changing the number of grid steps).
6. Compare approximation errors of these methods plotting the corresponding chart for different grid sizes (provide the possibility of changing the range of grid steps).

1. Initial Value Problem

$$\begin{cases} y' = xy - xy^3 \\ y(0) = \sqrt{0.5} \\ x \in (0, 3) \end{cases}$$

Solution:

$$y' = xy - xy^3$$

1. Using variable separation:

$$y' = x(y - y^3)$$

$$\frac{dy}{dx} = xy(1 - y^2)$$

$$\frac{dy}{y(1 - y^2)} = xdx$$

$$\int \frac{dy}{y(1 - y^2)} = \int xdx$$

2. Using partial fraction decomposition:

$$\int \left(\frac{1}{y} - \frac{1}{2(1 + y)} + \frac{1}{2(1 - y)}\right) dy = \int xdx$$

$$\int \frac{dy}{y} - \frac{1}{2}\int \frac{dy}{1 + y} + \frac{1}{2}\int \frac{dy}{1 - y} = \frac{x^2}{2} + C$$

$$\ln\left|\frac{y}{\sqrt{1 - y^2}}\right| = \frac{x^2}{2} + C$$

$$\frac{y}{\sqrt{1 - y^2}} = e^{\frac{x^2}{2} + C}$$

3. Squaring both side:

$$\frac{y^2}{1 - y^2} = e^{x^2 + 2c}$$

$$y^2 = e^{x^2 + 2c}(1 - y^2)$$

$$y^2 = e^{x^2 + 2c} - y^2 e^{x^2 + 2c}$$

$$y^2\left(1 - e^{x^2 + 2c}\right) = e^{x^2 + 2c}$$

$$y^2 = \frac{e^{x^2 + 2C}}{1 + e^{x^2 + 2C}}$$

4. Suppose $C_1 = \frac{1}{e^{2C}}$:

$$y^2 = \frac{e^{x^2}}{C_1 + e^{x^2}}$$

$$y = \sqrt{\frac{e^{x^2}}{C_1 + e^{x^2}}}$$

5. Let's derive a general formula for $C_1 = C$:

$$y^2 = \frac{e^{x^2}}{C + e^{x^2}}$$

$$C + e^{x^2} = \frac{e^{x^2}}{y^2}$$

$$C = \frac{e^{x^2}}{y^2} - e^{x^2}$$

$$C = e^{x^2}(\frac{1}{y^2} - 1)$$

6. We know that $y(0) = \sqrt{0.5}$, let's substitute:

$$C = e^0(\frac{1}{\frac{1}{2}} - 1)$$

$$C = 1$$

Answer:

$$y = \sqrt{\frac{e^{x^2}}{1 + e^{x^2}}}$$

Discontinuity points:

The particular function has no discontinuity points on the given interval. However, C does not exist if $y_0 = 0$

2. Programming Part

You can see the entire program code at the link: https://github.com/RakaVaqaFlow/Computational_practicum

For the implementation of the methods, I chose the C # programming language, since it has the necessary functionality (complex mathematical operations) and is easy to learn. To implement the interface, I used the Windows.Forms tool, since it is very powerful and easy to learn.

Structure of the program

The program has the following structure *(fig. 1)*:

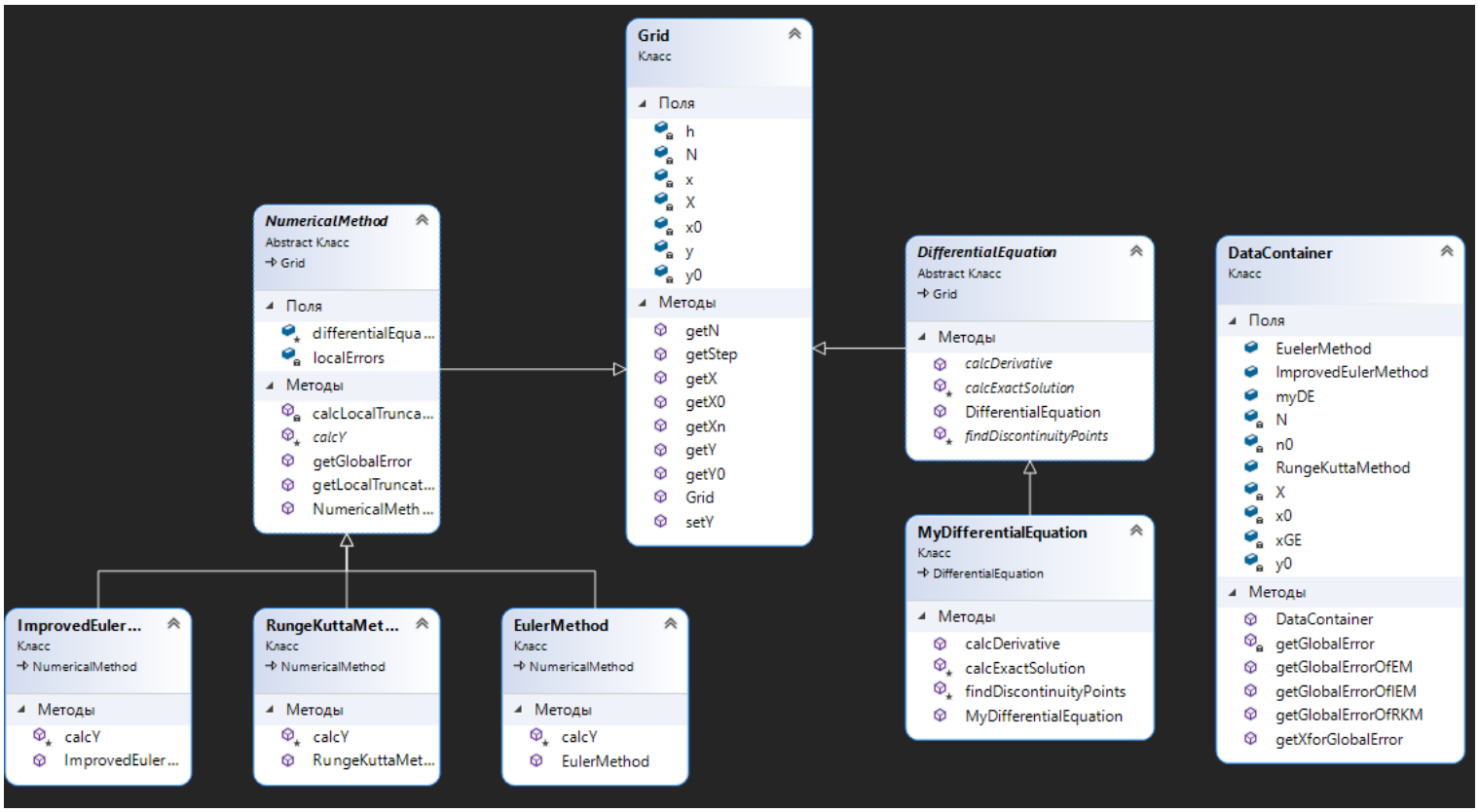

Fig. 1. UML-class diagram

This implementation is independent of a specific example. We can use this scheme for all options, it is enough just to inherit our class from the *DifferentialEquation* class (in our case variant 11 – *MyDifferentialEquation* class).

Graphical User Interface

The graphical interface has the following form

With our initial condition $y(0) = \sqrt{0.5}$ *and* $x \in (0, 3)$ we have:

- Solutions *(fig. 2)*

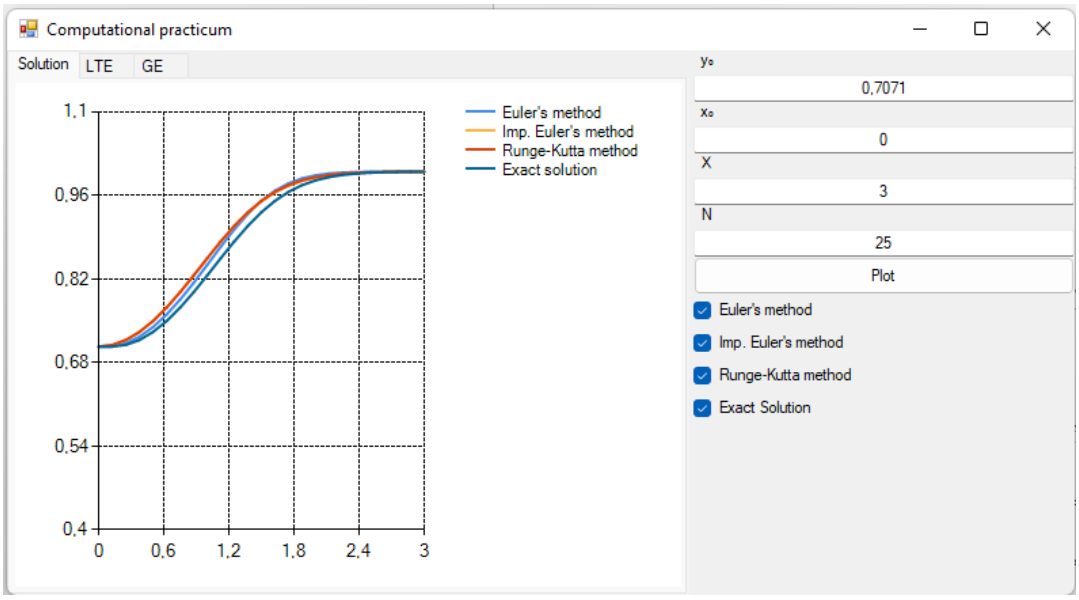

Fig. 2. Solution

- *Local Truncation Error (fig. 3)*

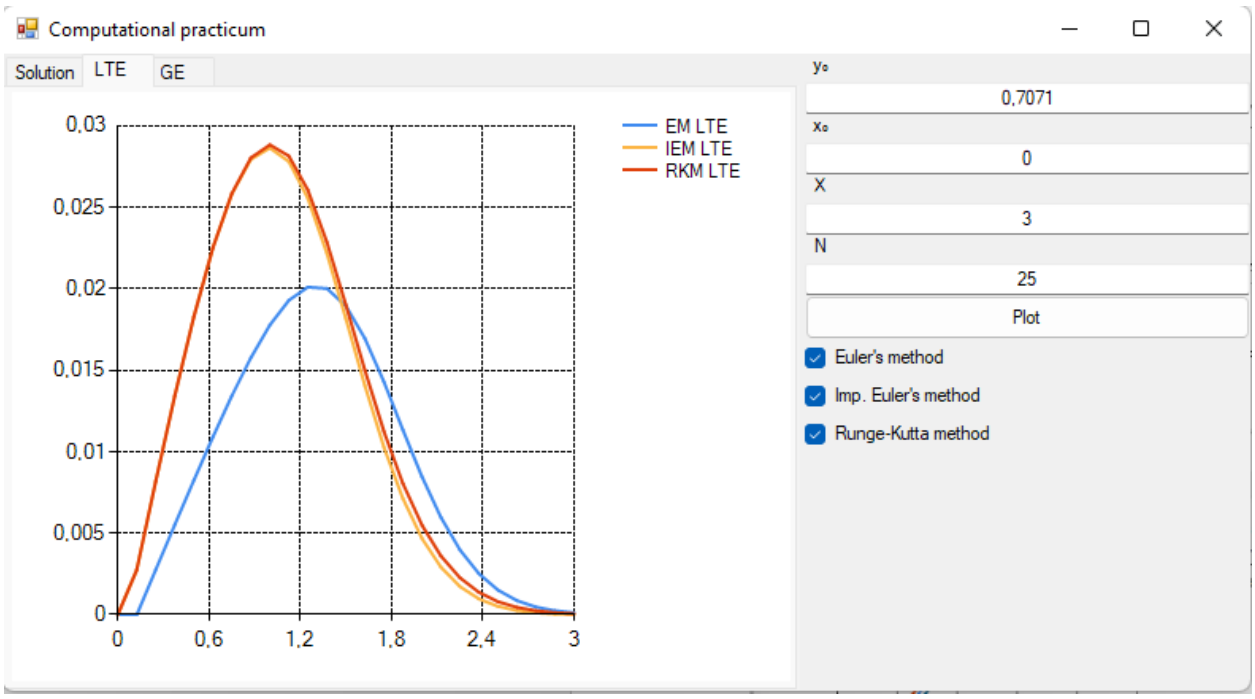

Fig. 3. Local Truncation Error

- *Global Error (fig. 4)*

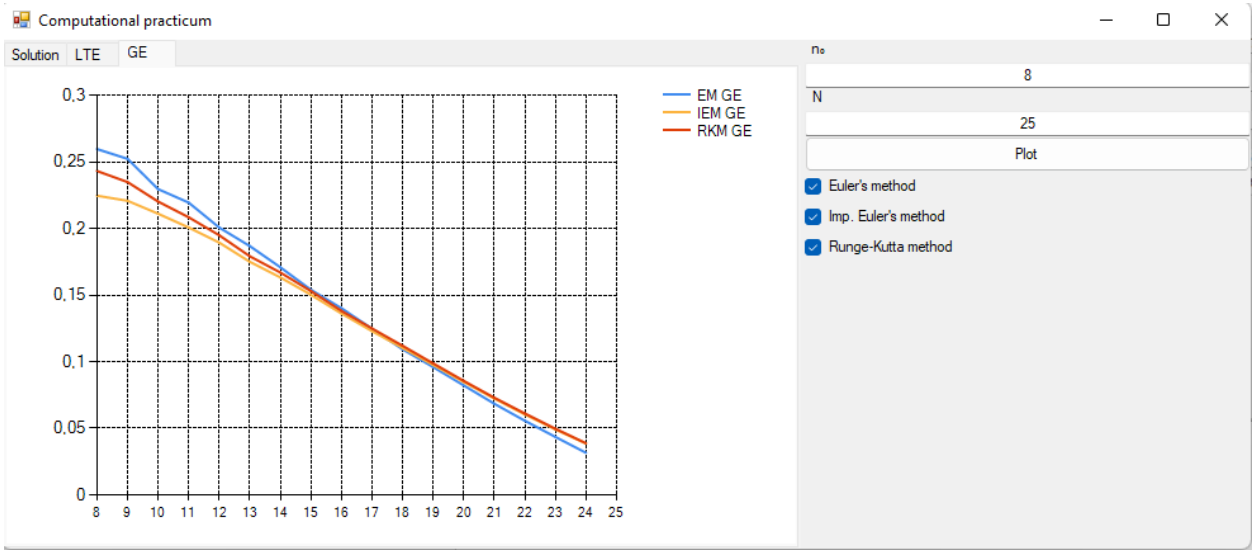

Fig. 4. Global Error

The graphical user interface allows us to change y_0, x_0, X, n_0 *and* N values, as well as hide or show a certain graph *(fig. 5)*:

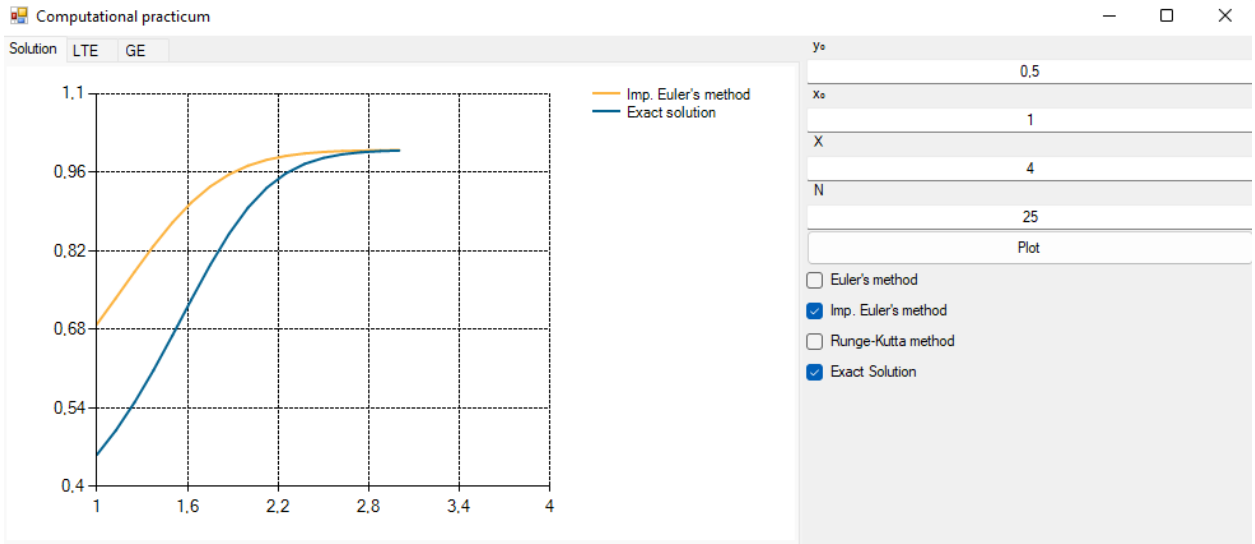

Fig. 5.

If the user enters invalid values, the program will display a message, for example, $y_0 = 0$ or $N < 0$ (fig. 6):

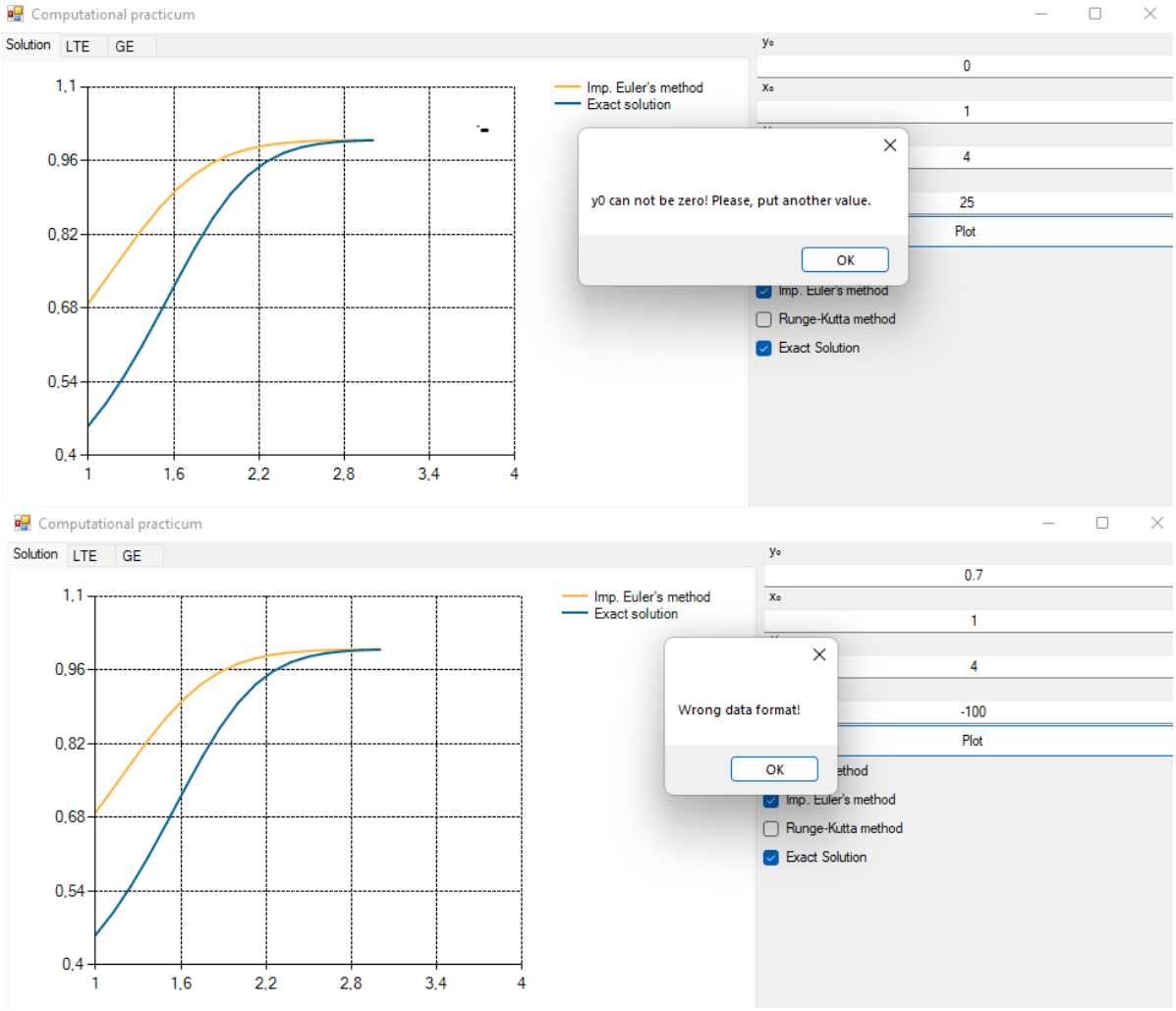

Fig. 6. Messages

Some implementation features

In my implementation, if you want to add a new numerical method, you just need to inherit a new class from the *NumericalMethod* class and redefine the function *calcY*(). For example, implementation of *EulerMethod* class (fig. 7):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Computation_Practicum_app
{
    Ссылок: 3
    public class EulerMethod : NumericalMethod
    {
        Ссылок: 2
        public EulerMethod(int N, double y0, double x0, double X, DifferentialEquation DE)
            : base(N, y0, x0, X, DE) { }

        Ссылок: 2
        protected override double calcY(double prevX, double prevY)
        {
            double h = getStep();
            double Y = prevY + h * differentialEquation.calcDerivative(prevX, prevY);
            return Y;
        }

    }
}
```

Fig. 7. EulerMethod

Also, if you want to see a solution for another varient, inherit the new class from the *DifferentialEquation* class and override the following methods:

- `double calcDerivative(double x, double y)`
- `void calcExactSolution()`
- `double[] findDiscontinuityPoints()`

For example, my variant 11 has structure (fig. 8):

```
public class MyDifferentialEquation : DifferentialEquation
{
    ссылка: 1
    public MyDifferentialEquation(int N, double y0, double x0, double X): base(N, y0, x0, X) {
        if (findDiscontinuityPoints().Length != 0)
        {
            string message = "The function has a discontinuity point in this interval!";
            throw new Exception(message);
        }
        calcExactSolution();
    }

    Ссылок: 8
    public override double calcDerivative(double x, double y) {
        return x * y - x * Math.Pow(y, 3);
    }
    Ссылок: 2
    protected override void calcExactSolution()
    {
        double h = getStep();
        double[] x = getX();
        double[] y = getY();
        double constant = Math.Exp(Math.Pow(getX0(), 2)) * (1 / Math.Pow(getY0(), 2)-1);
        for (int i = 0; i < getN(); i++)
        {
            if (i == 0) y[i] = getY0();
            else y[i] = Math.Sqrt(Math.Exp(Math.Pow(x[i - 1], 2)) / (constant + Math.Exp(Math.Pow(x[i - 1], 2))));
        }
        setY(y);
    }

    //let's return empty list, because my equation has no discontinuity points
    Ссылок: 2
    protected override double[] findDiscontinuityPoints() {
        return new List<double>().ToArray();
    }

}
}
```

Fig. 8. Variant 11 structure.

3. Method analysis

The graph shows *(fig. 3)* that numerical methods give a result close enough to the real one.

The plot of the local error *(fig. 4)* shows that the error of the Euler's method at small values of x is less than that of others (perhaps it depends on the function, since this is not observed for other variants), however, closer to the right border, the Runge-Kutta method and the Improved Euler's method are more accurate. If we look at the graph of the global error *(fig. 5)*, we can see that the Runge-Kutta method is indeed more accurate than the Improved Euler's method and Improved Euler's method more accurate than the Euler's method.

4. Conclusion

The Runge-Kutta method is the most accurate. The improved Euler method is fairly accurate. Euler's method is less accurate