Computer Application for Scientific Computing Final Project with $\LaTeX$

Woojeong Park
Department of Mathematical Science
Seoul National University
2015-17231

June 16, 2019

1 Introduction

In this project, we try to solve the non-linear ordinary differential equation called Lotka-Volterra model, which predicts the approximation of preys and predators in reality. And also solve the linear system with huge size of Matrix using several methods and compare what the differences are. There are many numerical methods including direct and iterative things, and we can apply to many mathematical problem in reality. All PseodoCode and graph can be used in FORTRAN90 and MATLABr2018.

2 Project I : Differential Equation

At first, we will solve the ordinary differential equation system, namely called ODE system. It is well-known how to solve it as an analytic solution, but here we consider it as a numerical problem.

2.1 Lotka-Volterra predatoryprey model system

The differential equation system below is called LotkaVolterra predatorprey model system.

$$\frac{dx}{dt} = Ax(t)(1 - By(t)), \quad x(0) = x_0$$
$$\frac{dy}{dt} = Cy(t)(Dx(t) - 1), \quad y(0) = y_0$$

The variable t denotes time, $x(t)$ the number of prey (e.g., rabbits) at time t, and $y(t)$ the number of predators (e.g., foxes). The positive constants A and C mean prey and predator population growth parameter, and B and D mean the species interaction parameters.
Let $A = 4$, $B = 12$, $C = 3$, $D = 13$ and $x(0) = 3$, $y(0) = 5$ be given. We try to solve that equation on $0 \leq t \leq 5$

$$\frac{dx}{dt} = 4x(t)(1 - 12y(t)), \quad x(0) = 3$$

$$\frac{dy}{dt} = 3y(t)(13x(t) - 1), \quad y(0) = 5$$

For solving this differential equation system, here we use 6 methods, plot them between time t and number of preys x and predators y, and also between x and y.

2.2 Explicit Euler Method

Explicit Euler Method, also called as `Foward Euler`, is very simple numerical method for integration of the first-order ODE. IF we should solve the ODE given:

$$\frac{dy}{dt} = f(t, y)$$

Then the basic algorithm using is

$$y_{n+1} = y_n + hf(y_n, t_n),\ n = 0,\ 1,\ 2,\ 3,\ \cdots$$

where y_n denotes $y(t_n)$. So apply this method to our program, we get

$$x_{n+1} = x_n + h(4x_n - 48x_n y_n)$$

$$y_{n+1} = y_n + h(-3y_n + 39x_n y_n)$$

Here is the pseudo code using this algorithm above :

```
PseudoCode :  Explicit Euler Method
1.  do j=1, iter
2.      x(j+1)=x(j)+(4*x(j)-48*x(j)*y(j))*h
3.      y(j+1)=y(j)+(-3*y(j)+39*x(j)*y(j))*h
4.  enddo
```

We want to solve it on $0 \leq t \leq 5$, so if we are going to set $h = 0.001$, $h = 0.005$, $h = 0.00025$, then the value of the `iter` above that algorithm should be 5000, 10000, and 20000.

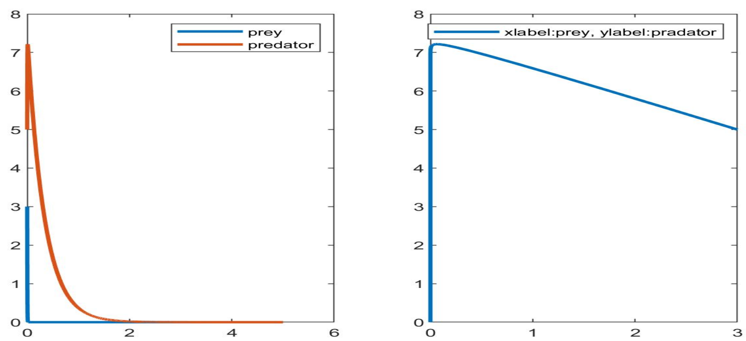

Figure 1: Explicit Euler with $h = 0.001$

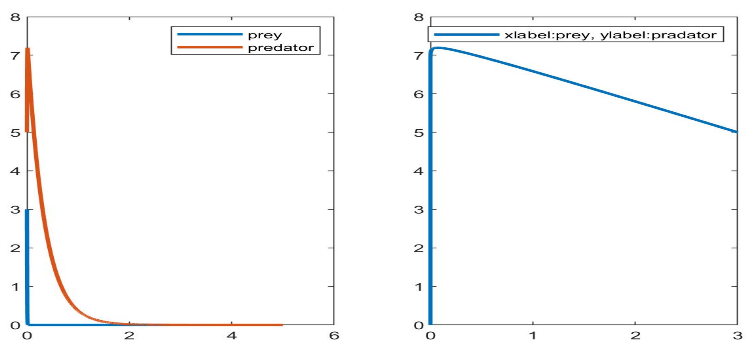

Figure 2: Explicit Euler with $h = 0.0005$

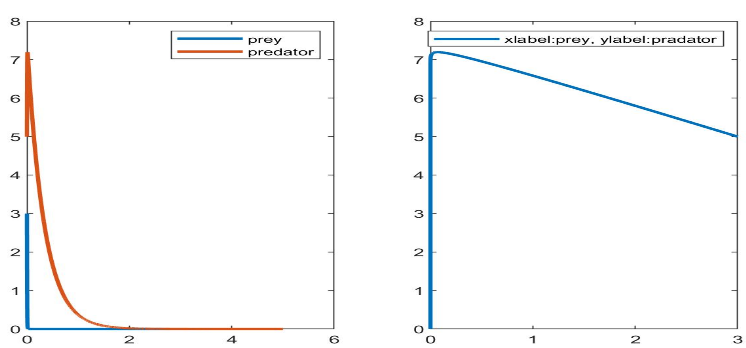

Figure 3: Explicit Euler with $h = 0.00025$

Briefly, we can notice that the number of preys and predators started at 5 and 3 respectively, and they become almost to zero at $t \geq 2$. There are no explicit difference with the different value of h. I think because the number of predators at the begin is larger than preys, so they are predicted all going to be extinct.

2.3 Implicit Euler Method

Implicit Euler Method, also called `Backward Euler`, costs higher than the explicit method before, but it is significantly stable than other methods. The basic algorithm is below :

$$y_{n+1} = y_n + hf(y_{n+1}, t_{n+1})$$

So our problem is :

$$x_{n+1} = x_n + h(4x_{n+1} - 48x_{n+1}y_n)$$
$$y_{n+1} = y_n + h(-3y_{n+1} + 39x_n y_{n+1})$$

Note that for applying this method to solve that ODE system, we should remain y_n and x_n at the first and second equation. Now describe what x_{n+1} and y_{n+1} are. Just describing only the x_{n+1} and y_{n+1} is below :

$$x_{n+1} = \frac{x_n}{1 - h(4 - 48y_n)}$$
$$y_{n+1} = \frac{y_n}{1 - h(-3 + 39x_n)}$$

Here is the pseudo code using this algorithm above :

```
PseudoCode :  Implicit Euler Method
1.  do j=1, iter
2.      x(j+1)=x(j)/(1-h*(4-48*y(j)))
3.      y(j+1)=y(j)/(1-h*(-3+39*x(j)))
4.  enddo
```

Similar to Explicit Euler method, the number of iteration is decided on the same way, and execute the code using FORTRAN90, we get the following result.

We can figure out that the result from `Implicit Euler Method` is the same as the result from `Explicit Euler Method`

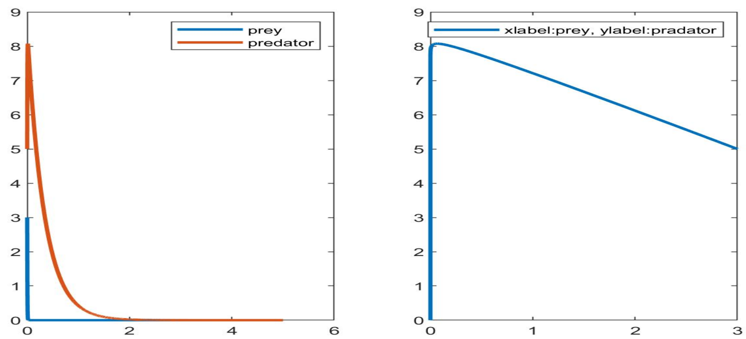

Figure 4: Implicit Euler with $h = 0.001$

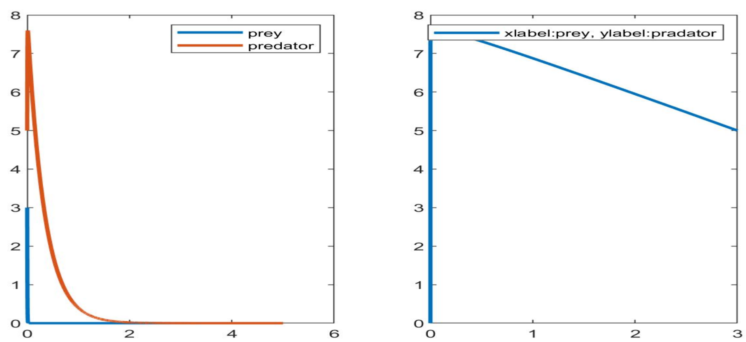

Figure 5: Implicit Euler with $h = 0.0005$

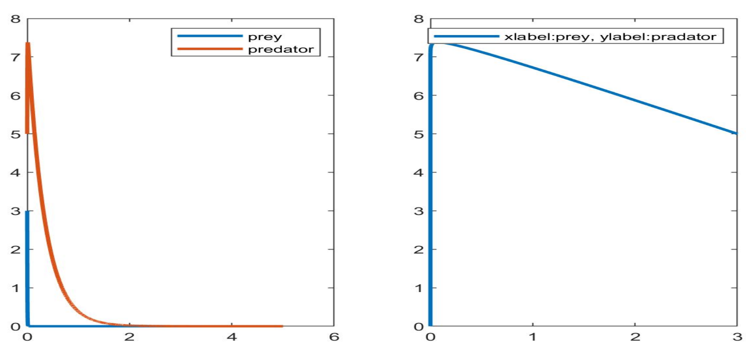

Figure 6: Implicit Euler with $h = 0.00025$

2.4 Improved Euler Method

Improved Euler Method is basically the combination of Explicit and Implicit Methods. There are many combinations for using Improved Euler Methods, we use that relation below : (This method is called Trapezoidal method.)

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

On that problem, similar to implicit method, we should change the form of that relation only for y_{n+1}. Let's see our problem is :

$$x_{n+1} = x_n + \frac{h}{2}(4x_n - 48x_n y_n + 4x_{n+1} - 48x_{n+1} y_n)$$

$$y_{n+1} = y_n + \frac{h}{2}(-3y_n + 39x_n y_n - 3y_{n+1} + 39x_n y_{n+1})$$

So we get

$$x_{n+1} = \frac{x_n(1 + h(2 - 24y_n))}{1 - h(2 - 24y_n)}$$

$$y_{n+1} = \frac{y_n(1 + \frac{h}{2}(-3 + 39x_n))}{1 - \frac{h}{2}(-3 + 39x_n)}$$

It looks kinds of complicated, but it can help to reduce the error of order 3 or higher than explicit or implicit Euler methods. And it also optimizes the computational cost, so if we can get other methods instead of trapezoid, we develop the numerical precision as high as possible.

Here is the pseudo code using this algorithm above :

```
PseudoCode :  Improved Euler Method
do j=1, iter
    x(j+1)=x(j)*(1+h*(2-24*y(j)))/(1-h*(2-24*y(j)))
    y(j+1)=y(j)*(1+h*(-3+39*x(j))/2)/(1-h*(-3+39*x(j))/2)
enddo
```

And, as we expected, excuting the code using `FORTRAN90`, we get the same result.

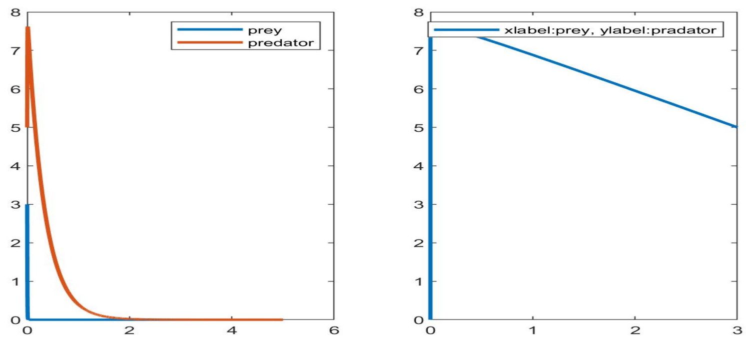

Figure 7: Improved Euler with $h = 0.001$

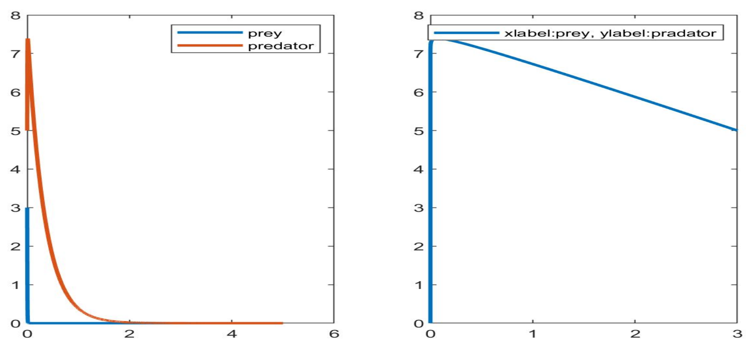

Figure 8: Improved Euler with $h = 0.0005$

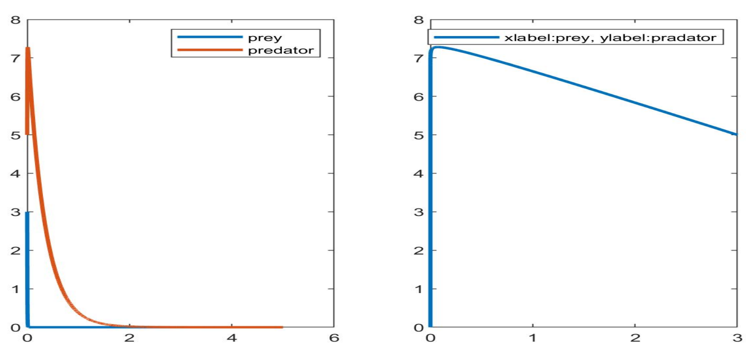

Figure 9: Improved Euler with $h = 0.00025$

2.5 Runge-Kutta Method

Runge-Kutta Method is one of the famous explicit method to solve ODE $\frac{dy}{dt} = f(y,t)$. Runge-Kutta method is to find intermediate point between t_n and t_{n+1} and we can predict the next value y_n more accurately. There 2 kinds of Runge-Kutta method will be introduced. The order means 'how much the error, the difference between approximation and exact value, is reduced'.

2.5.1 RK-2nd Order

The general form of Runge-Kutta Method 2nd Order is :

$$y_{n+1} = y_n + \gamma_1 k_1 + \gamma_1 k_2$$

where γ_1, γ_2 is constant and

$$k_1 = hf(y_n, t_n), \quad k_2 = hf(y_n + \beta k_1, t_n + \alpha h)$$

We're going to use the most popular form. Let $\gamma_1 = 0$, $\gamma_2 = 1$, $\alpha = \beta = \frac{1}{2}$, then we get

$$y_{n+1} = y_n + hf(y_n + \frac{1}{2}hf(y_n, t_n), t_n + \frac{1}{2}h)$$

Apply to our problem, then the final form is :

$$k_{1x} = h(4x_n - 48x_n y_n)$$

$$k_{1y} = h(-3y_n + 39x_n y_n)$$

$$k_{2x} = h(4(x_n + \frac{1}{2}k_{1x}) - 48(x_n + \frac{1}{2}k_{1x})y_n)$$

$$k_{2y} = h(-3(y_n + \frac{1}{2}k_{1y}) + 39(y_n + \frac{1}{2}k_{1y})x_n)$$

$$x_{n+1} = x_n + k_{2x}, \qquad y_{n+1} = y_n + k_{2y}$$

Here is the pseudo code using this algorithm above :

```
PseudoCode :  Runge-Kutta 2nd Order
do j=1, iter
    k1x=h*(4*x(j)-48*x(j)*y(j))
    k1y=h*(-3*y(j)+39*y(j)*x(j))
    k2x=h*(4*(x(j)+k1x/2)-48*(x(j)+k1x/2)*y(j))
    k2y=h*(-3*(y(j)+k1y/2)+39*(y(j)+k1y/2)*x(j))
    x(j+1)=x(j)+k2x
    y(j+1)=y(j)+k2y
enddo
```

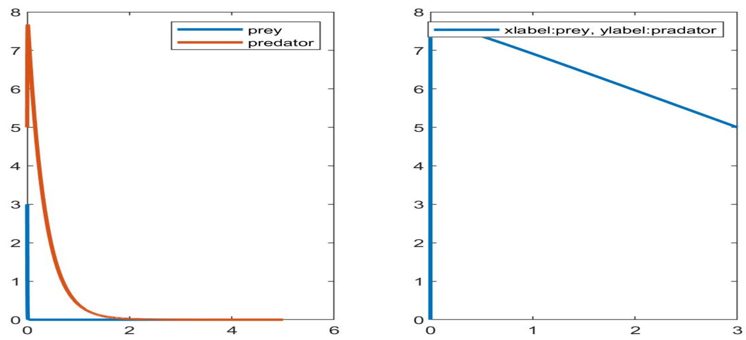

Figure 10: Runge-Kutta 2nd Order with $h = 0.001$

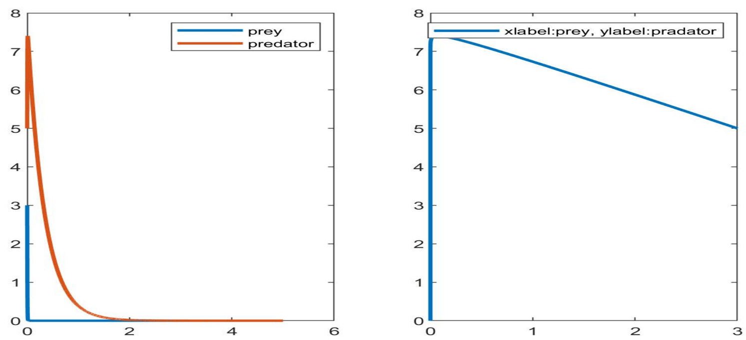

Figure 11: Runge-Kutta 2nd Order with $h = 0.0005$

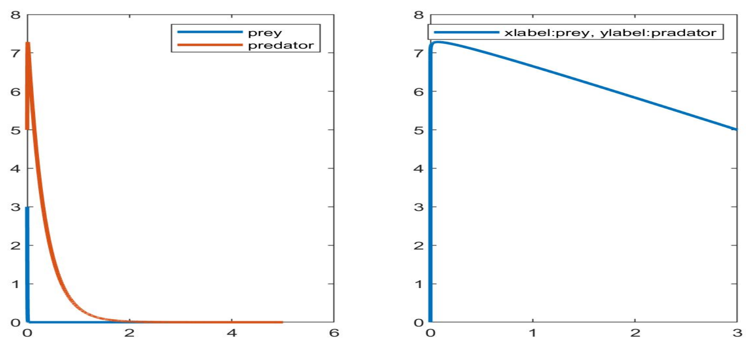

Figure 12: Runge-Kutta 2nd Order with $h = 0.00025$

Table 1: Sample table title

Part		
Name	Description	Size (μm)
Dendrite	Input terminal	$\sim$100
Axon	Output terminal	$\sim$10
Soma	Cell body	up to 10^6

2.5.2 RK-4th Order

3 Examples of citations, figures, tables, references

[1, 2] and see [3].

The documentation for `natbib` may be found at

`http://mirrors.ctan.org/macros/latex/contrib/natbib/natnotes.pdf`

Of note is the command `\citet`, which produces citations appropriate for use in inline text. For example,

```
\citet{hasselmo} investigated\dots
```

produces

> Hasselmo, et al. (1995) investigated...

`https://www.ctan.org/pkg/booktabs`

3.1 Figures

3.2 Tables

See awesome Table 1.

3.3 Lists

- Lorem ipsum dolor sit amet
- consectetur adipiscing elit.
- Aliquam dignissim blandit est, in dictum tortor gravida eget. In ac rutrum magna.

References

[1] George Kour and Raid Saabne. Real-time segmentation of on-line handwritten arabic script. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 417–422. IEEE, 2014.

[2] George Kour and Raid Saabne. Fast classification of handwritten on-line arabic characters. In *Soft Computing and Pattern Recognition (SoCPaR), 2014 6th International Conference of*, pages 312–318. IEEE, 2014.

[3] Guy Hadash, Einat Kermany, Boaz Carmeli, Ofer Lavi, George Kour, and Alon Jacovi. Estimate and replace: A novel approach to integrating deep neural networks with existing applications. *arXiv preprint arXiv:1804.09028*, 2018.