PHYS580 Lab01 Report

Yicheng Feng
PUID: 0030193826

August 28, 2019

Before my solutions to the lab activities, I want to set up my typical **workflow**, which will also be used in this course. I use the Linux system, and code C++ in terminals. For visualization, I use the C++ package – ROOT (made by CERN) to make plots. At the end, the reports are written in LaTeX.

The codes for this lab are written as three files: `nuclei_decay.h`, `nuclei_decay.cxx` for the class `NucleiDecay` to do calculations; `lab1.cxx` for the `main` function to do plots. To each problem of this lab report, I will attach the relevant parts of the code. If you want to check the validation of my code, you need to download the whole code from the link `https://github.com/YichengFeng/phys580/tree/master/lab1`.

---

(1) Not required for the report.

---

(2) Implement the Euler approximation for the nuclear decay problem of Ch 1, in Matlab (or any language of your choice, provided you are familiar with it and do not wish to make use of the provided starter codes). Compute solutions from $t = 0$ to 5τ using time steps $\Delta t/\tau = 0.05, 0.2, 0.8, 1$, and 1.5. Plot the Euler approximation results together with the exact solution.

Plot:

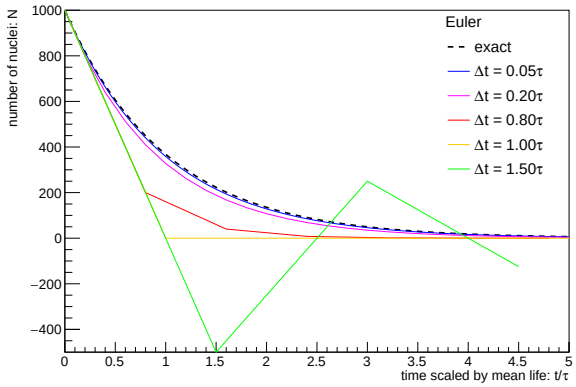

Figure 1: Euler approximation and the exact solution for the nuclear decay problem with various Δt.

Physics explanation:

The nuclear decay problem is given by the differential equation

$$\frac{\mathrm{d}N}{\mathrm{d}t} = -\frac{1}{\tau}N, \tag{1}$$

where N is a function of time t, and τ is a constant called mean life. It is easy to get the exact solution to this equation

$$N(t) = N(0)e^{-t/\tau}. \tag{2}$$

The Euler approximation is a numerical approach to this solution

$$\begin{cases} N_{i+1} = N_i - \dfrac{N_i \Delta t}{\tau} \\ t_{i+1} = t_i + \Delta t \end{cases}, \tag{3}$$

with the initial condition $N(t = 0) = N_0$. From Fig. 1, we can easily see that the accuracy increases with smaller steps.

Relevant code:

```
1  double NucleiDecay::fxt(double x, double t) {
2          return -x;
3  }
4
5  void NucleiDecay::cal_g_exact() {
6
7          int nt = (int)((_t_end - _t_start)/_dt)+1;
8          double  *N_exact = new double[nt];
9          double  *t = new double[nt];
10
11         N_exact[0] = _N0;
12         t[0] = _t_start;
13
```

```
14          for(int i=1; i<nt; i++) {
15                  t[i] = t[i-1] + _dt;
16                  N_exact[i] = _N0*TMath::Exp(-t[i]);
17          }
18
19          _g_exact = new TGraph(nt, t, N_exact);
20
21          return;
22  }
23
24  void NucleiDecay::cal_g_euler() {
25
26          int nt = (int)((_t_end - _t_start)/_dt)+1;
27          double  *N_euler = new double[nt];
28          double  *t = new double[nt];
29
30          N_euler[0] = _N0;
31          t[0] = _t_start;
32
33          for(int i=1; i<nt; i++) {
34                  t[i] = t[i-1] + _dt;
35                  N_euler[i] = N_euler[i-1] + fxt(N_euler[i-1], t[i-1])*_dt;
36          }
37
38          _g_euler = new TGraph(nt, t, N_euler);
39
40          return;
41  }
```

---

(3) Calculate and analyze the deviations between your Euler results in (2) and the exact result, and study their dependence on the step size $\Delta t/\tau$ (at fixed t) and on t (number n of steps used up to t) at fixed $\Delta t/\tau$, in the range $t = 0$ through 5τ. Include a plot of the cumulative deviations (i.e., the global error) as a function of t, both in absolute terms and also as a fraction relative to the exact values.

Plots:

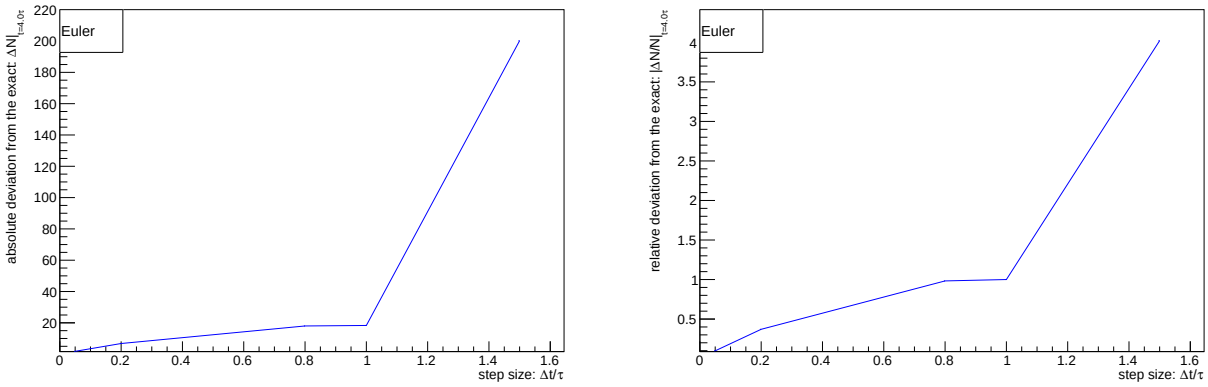

Figure 2: the absolute deviation (left pad) and relative deviation (right pad) between the Euler approximation and the exact solution depending on Δt at $t = 4\tau$. (note: for the last point $\Delta t = 1.5\tau$, the time is $t = 3\tau$.)

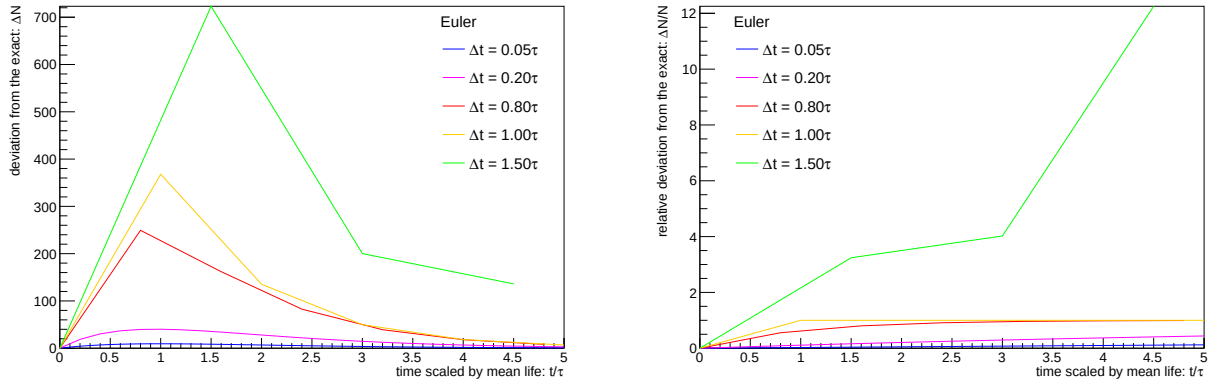

Figure 3: the absolute deviation (left pad) and relative deviation (right pad) between the Euler approximation and the exact solution depending on t.

Physics explanation:

If we denote the exact solution to Eq. 1 as $N_{\text{exact}}(t)$, and the Euler approximation from Eq. 3 as $N_{\text{Euler}}(t)$. The absolute term of the deviation between the Euler result and the exact result is

$$\Delta N_{\text{Euler}}(t) = |N_{\text{Euler}}(t) - N_{\text{exact}}(t)|, \tag{4}$$

and the fraction relative to the exact values:

$$\Delta N_{\text{Euler}}(t)/N_{\text{exact}}(t). \tag{5}$$

To study the dependence of the deviation on the Δt, we need to fix t for various Δt. However, there is no common multiple for $0.05, 0.2, 0.8, 1.0, 1.5$ within $(0, 5]$. For the first four Δt, the number 4 is suitable, so we use the time $t = 4\tau$ for $\Delta t/\tau = 0.05, 0.2, 0.8, 1.0$ and use time $t = 3\tau$ for $\Delta t/\tau = 1.5$. The results are shown in the Fig. 2, left pad for the absolute deviation and right for the relative deviation.

To study the dependence of the deviation on t (and number of step n), we make the plots in Fig. 3. The left pad shows the absolute deviation depending on t (also n). We can see the curves are not always decreasing or increasing, because the exact solution decreases over time. The right pad shows the relative deviation depending on t (also n). We can see the relative deviations always increase with t (or n). Both plots show that the smaller the Δt is, the smaller deviation would be.

Relevant code:

```
TGraph* NucleiDecay::cal_g_error(TGraph *g) {

        if(!g || !_g_exact) {
                cout << "g or _g_exact empty!" << endl;
                return nullptr;
        }

        int nt = g->GetN();
        double *t = g->GetX();
        double *N = g->GetY();

        double *t_exact = _g_exact->GetX();
        double *N_exact = _g_exact->GetY();

        double *N_error = new double[nt];

        for(int i=0; i<nt; i++) {
                if(t[i] != t_exact[i]) {
                        cout << "time not match!" << endl;
                        return nullptr;
                }
        }

        for(int i=0; i<nt; i++) {
                N_error[i] = fabs( N[i] - N_exact[i] );
        }

        TGraph *g_error = new TGraph(nt, t, N_error);

        return g_error;
}

TGraph* NucleiDecay::cal_g_relative_error(TGraph *g) {

        if(!g || !_g_exact) {
                cout << "g or _g_exact empty!" << endl;
                return nullptr;
        }

        int nt = g->GetN();
        double *t = g->GetX();
        double *N = g->GetY();

        double *t_exact = _g_exact->GetX();
        double *N_exact = _g_exact->GetY();

```

```
47          double *N_relative_error = new double[nt];
48
49          for(int i=0; i<nt; i++) {
50                  if(t[i] != t_exact[i]) {
51                          cout << "time not match!" << endl;
52                          return nullptr;
53                  }
54          }
55
56          for(int i=0; i<nt; i++) {
57                  N_relative_error[i] = fabs( N[i] - N_exact[i] )/N_exact[i];
58          }
59
60          TGraph *g_relative_error = new TGraph(nt, t, N_relative_error);
61
62          return g_relative_error;
63  }
```

---

(4) Implement the second- and fourth-order Runge-Kutta approximations, and do the same as in (2) and (3). How do the Runge-Kutta results compare to the exact solution and to the Euler approximation results? Perform an analogous error analysis to what you did in (3).

Physics explanation:

For the second-order Runge-Kutta approximation, Figure 4 shows the approximation results with various Δt; Figure 5 shows the absolute and relative deviations between the approximation and the exact results. They increase with increasing Δt. Figure 6 shows the deviations' depnedence on t (or n). The relative deviations increase with increasing t (or n).

For the fourth-order Runge-Kutta approximation, the corresponding plots (Figs. 7, 8, and 9) have the similar behaviors as discussed above.

Both the second- and fourth-order Runge-Kutta approximation are closer to the exact solution than the Euler approximation. The fourth-order Runge-Kutta is even closer to the exact solution than the second-order approximation.

Plots:

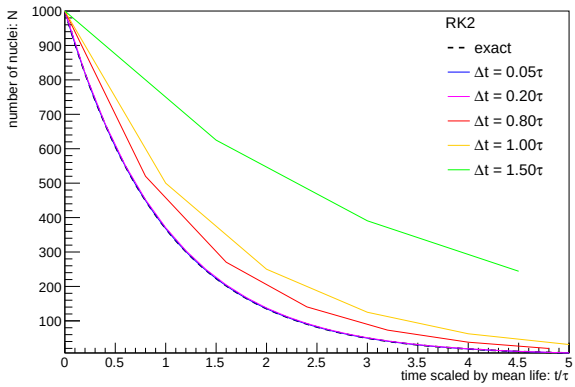

Figure 4: RK2 approximation and the exact solution for the nuclear decay problem with various Δt.

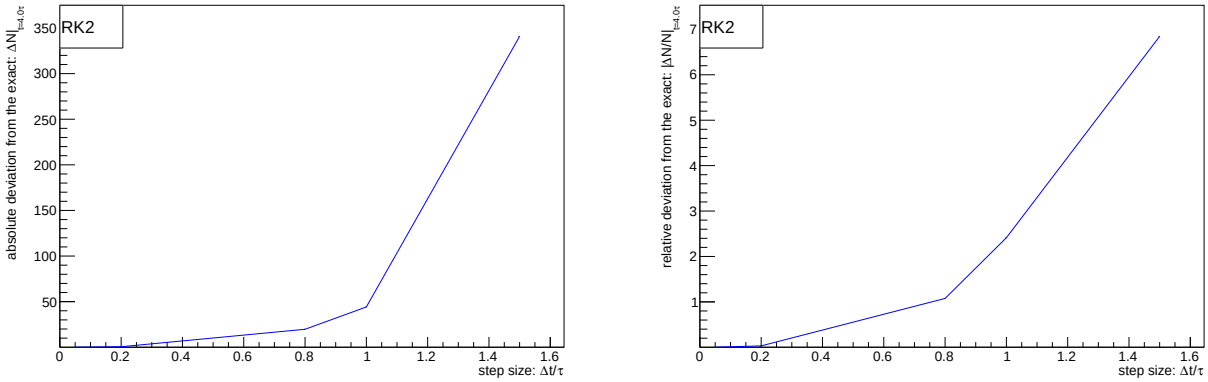

Figure 5: the absolute deviation (left pad) and relative deviation (right pad) between the RK2 approximation and the exact solution depending on Δt at $t = 4\tau$. (for the last point $\Delta t = 1.5\tau$, $t = 3\tau$.)

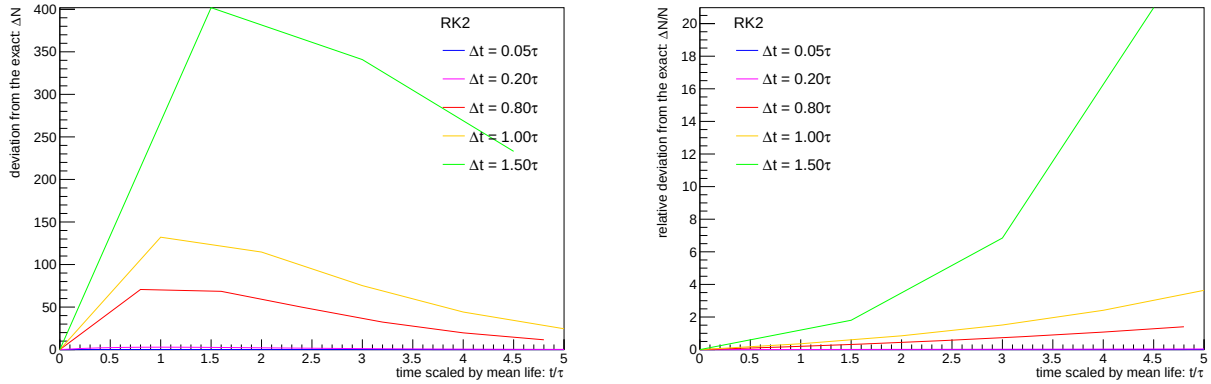

Figure 6: the absolute deviation (left pad) and relative deviation (right pad) between the RK2 approximation and the exact solution depending on t.

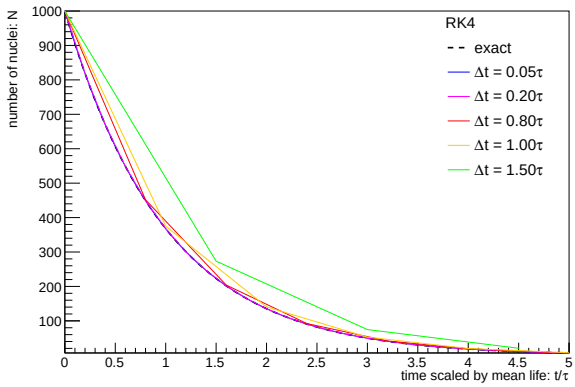

Figure 7: RK4 approximation and the exact solution for the nuclear decay problem with various Δt.

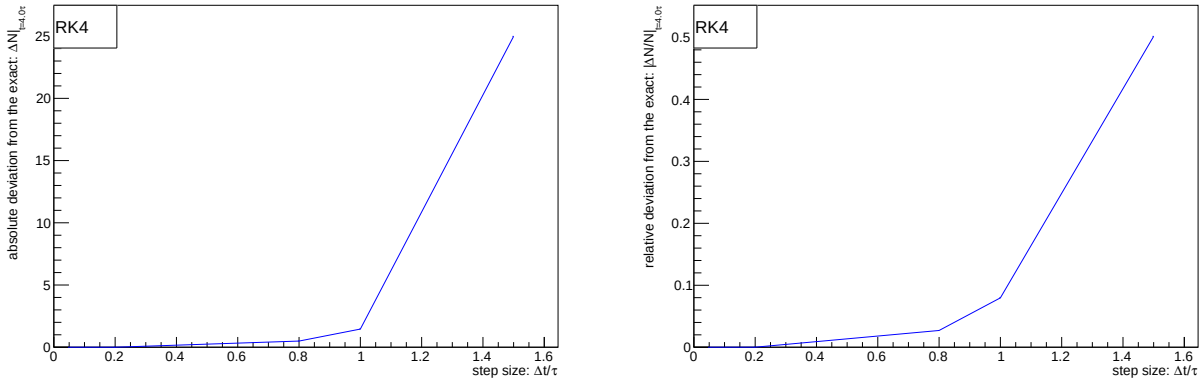

Figure 8: the absolute deviation (left pad) and relative deviation (right pad) between the RK4 approximation and the exact solution depending on Δt at $t = 4\tau$. (for the last point $\Delta t = 1.5\tau$, $t = 3\tau$.)

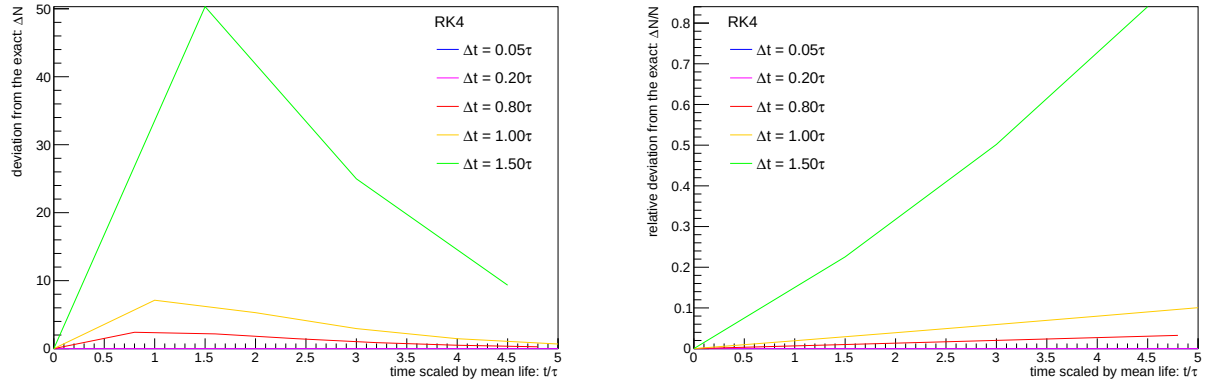

Figure 9: the absolute deviation (left pad) and relative deviation (right pad) between the RK4 approximation and the exact solution depending on t.

Relevant code:

```
double NucleiDecay::fxt(double x, double t) {
        return -x;
}

void NucleiDecay::cal_g_rk2() {

        int nt = (int)((_t_end - _t_start)/_dt)+1;
        double  *N_rk2 = new double[nt];
        double  *t = new double[nt];

        N_rk2[0] = _N0;
        t[0] = _t_start;

        for(int i=1; i<nt; i++) {
                t[i] = t[i-1] + _dt;

                double F1 = fxt(N_rk2[i-1], t[i-1]);
                double F2 = fxt(N_rk2[i-1]+F1*0.5*_dt, t[i-1]+0.5*_dt);
                N_rk2[i] = N_rk2[i-1] + F2*_dt;
        }

        _g_rk2 = new TGraph(nt, t, N_rk2);

        return;
}

void NucleiDecay::cal_g_rk4() {

        int nt = (int)((_t_end - _t_start)/_dt)+1;
        double  *N_rk4 = new double[nt];
        double  *t = new double[nt];

        N_rk4[0] = _N0;
        t[0] = _t_start;

        for(int i=1; i<nt; i++) {
                t[i] = t[i-1] + _dt;

                double F1 = fxt(N_rk4[i-1], t[i-1]);
                double F2 = fxt(N_rk4[i-1]+F1*0.5*_dt, t[i-1]+0.5*_dt);
                double F3 = fxt(N_rk4[i-1]+F2*0.5*_dt, t[i-1]+0.5*_dt);
                double F4 = fxt(N_rk4[i-1]+F3*_dt, t[i-1]+_dt);
                N_rk4[i] = N_rk4[i-1] + ( F1/6 + F2/3 + F3/3 + F4/6 )*_dt;
        }

        _g_rk4 = new TGraph(nt, t, N_rk4);

        return;
}
```

---

(5) Based on your findings above, can you make any general observation regarding the approximation methods used here?

In a nutshell, the findings I get from the activities above are listed as follow:

- For a fixed t, the approximation with smaller Δt has higher accuracy.
- For a specific Δt, the relative deviation $\Delta N/N$ (global deviation) of the approximation to the exact result increases with increasing t and increasing number of steps n. The absolute deviation is not necessary increasing, because the magnitude of the exact solution can also change along time.
- The general accuracy of each approximation can be ranked as: RK4 > RK2 > Euler.

In fact, the Euler approximation is just the first-order Runge-Kutta approximation. Then, the observation regarding the approximation methods used here could be that the accurcy of approximation generally gets better with the increase of the order of the Runge-Kutta methods.

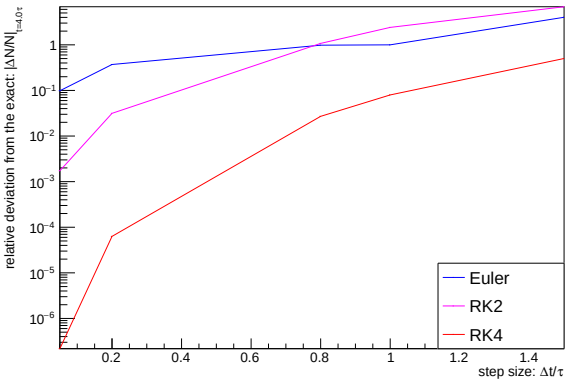

Figure 10: the relative deviations between various approximations and the exact solution depending on Δt at $t = 4\tau$. (for the last point $\Delta t = 1.5\tau$, $t = 3\tau$.)

However, the accuracy difference could get smaller or event inversed with larger Δt. This is also reasonable, because the Taylor analysis tells us that the global errors are given as $\mathcal{O}(\Delta t)$ for RK1 (Euler), $\mathcal{O}(\Delta t^2)$ for RK2, and $\mathcal{O}(\Delta t^4)$ for RK4. When Δt gets larger, the difference (in ratio) between them could become smaller.
