Laboratorium *Podstaw fizyki teoretycznej.* Implementacja metody RK4.

4 maja 2022

Na laboratorium będziemy numerycznie rozwiązywać równania ruchu, których nie byliśmy w stanie rozwiązać analitycznie na ćwiczeniach. Aby poszukiwane rozwiązania cechowały się wysoką jakością (tj. błędy numeryczne mają być niewielkie) do ich znalezienia musimy użyć odpowiedniej metody. Takie własności ma metoda Rungego-Kutty 4 rzędu (RK4). Można ją stosować zarówno do rozwiązania pojedynczego równania różniczkowego jak i do rozwiązania układu równań różniczkowych 1 rzędu. Istotną kwestią w rozwiązywaniu równań ruchu są warunki początkowe, determinują one całe rozwiązanie, musimy zatem zawsze pamiętać, aby je poprawnie zaimplementować w programie.

1 Równanie różniczkowe zwyczajne 1 rzędu

Zastosowanie algorytmu metody RK4 najłatwiej zobrazować na przykładzie pojedynczego równania różniczkowego

$$\frac{dy}{dt} = f(t,y), \quad WP: \; y(t=0) = y_0 \tag{1}$$

Pierwszym krokiem jest dyskretyzacja zmiennej czasowej. Na osi czasu wprowadzamy siatkę równoodległych węzłów

$$t_i = \Delta t \cdot i, \quad i = 0, 1, \ldots, N, \quad \Delta t = \frac{t_{max}}{N} \tag{2}$$

gdzie: t_{max} to całkowity czas trwania symulacji komputerowej, N jest liczbą kroków czasowych. Jeśli znamy rozwiązanie w chwili t_i (y_i) to rozwiązanie w chwili następnej

$$y_{i+1} = y_i + \frac{\Delta t}{6}\left(k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4\right) \tag{3}$$

gdzie wartości współczynników k_α wyliczamy korzystając z prawej strony RRZ1 tj. $f(t,y)$ licząc kolejno

$$k_1 = f(t_i,\, y_i) \tag{4}$$
$$k_2 = f(t_i + \frac{\Delta t}{2},\, y_i + \frac{\Delta t}{2} k_1) \tag{5}$$
$$k_3 = f(t_i + \frac{\Delta t}{2},\, y_i + \frac{\Delta t}{2} k_2) \tag{6}$$
$$k_4 = f(t_i + \Delta t,\, y_i + \Delta t\, k_3) \tag{7}$$
$$\tag{8}$$

Najprostszy algorytm $RK4$ mógłby wyglądać następująco

```
WP:  y=y0
     t=0
     dt=t_max/N
for i from 1 to N do

        k1 = f(t, y)
        k2 = f(t+dt/2, y+dt/2*k1)
        k3 = f(t+dt/2, y+dt/2*k2)
        k4 = f(t+dt/2, y+dt*k3)

        y=y+dt/6*(k1+2*k2+2*k3+k4)       (tu: y = y_{i+1})
```

```
        t=t+dt

        zapis do pliku:  t, y
end do
```

2 Układ równań różniczkowych 1 rzędu

Rozważmy teraz nieco trudniejszy przypadek. W formalizmie Lagrange'a równania ruchu są zazwyczaj drugiego rzędu. Zapiszmy takie równanie w ogólnej postaci

$$\frac{d^2y}{dt^2} + g(t,y)\,\frac{dy}{dt} + h(t,y) = 0 \tag{9}$$

gdzie $g(t,y)$ i $h(t,y)$ są pewnymi funkcjami czasu i naszego rozwiązania. Wprowadźmy dwie nowe zmienne

$$s_1 = y \tag{10}$$

$$s_2 = \frac{dy}{dt} \tag{11}$$

które pozwolą nam zapisać oryginalne RRZ 2 rzędu

$$\frac{d^2y}{dt^2} = -g(t,y)\,\frac{dy}{dt} - h(t,y) \tag{12}$$

w postaci układu RRZ 1 rzędu

$$\frac{ds_1}{dt} = f_1(t,s_1,s_2) = s_2 \tag{13}$$

$$\frac{ds_2}{dt} = f_2(t,s_1,s_2) = -g(t,s_1)\,s_2 + h(t,s_1) \tag{14}$$

który możemy zapisać w postaci wektorowej

$$\frac{d\vec{s}}{dt} = \vec{f}(t,\vec{s}\,) \tag{15}$$

Równania algorytmu RK4 możemy bardzo łatwo dopasować do naszego układu RRZ1

$$\vec{k}_1 = \vec{f}(t_i,\,\vec{s}_i) \tag{16}$$

$$\vec{k}_2 = \vec{f}(t_i + \frac{\Delta t}{2},\,\vec{s}_i + \frac{\Delta t}{2}\vec{k}_1) \tag{17}$$

$$\vec{k}_3 = \vec{f}(t_i + \frac{\Delta t}{2},\,\vec{s}_i + \frac{\Delta t}{2}\vec{k}_2) \tag{18}$$

$$\vec{k}_4 = \vec{f}(t_i + \Delta t,\,\vec{s}_i + \Delta t\,\vec{k}_3) \tag{19}$$

$$\vec{y}_{i+1} = \vec{y}_i + \frac{\Delta t}{6}\left(\vec{k}_1 + 2\cdot\vec{k}_2 + 2\cdot\vec{k}_3 + \vec{k}_4\right) \tag{20}$$

2.1 Implementacja RK4 dla układu RRZ1

2.1.1 Procedura implementująca RK4

Ponieważ metodę RK4 będziemy wykorzystywać na kilku zajęciach, ułatwimy sobie zadanie jeśli procedurę tę zaimplementujemy w jak najbardziej ogólnej postaci. Wówczas nie będziemy musieli jej zmieniać za każdym razem aby dopasować do aktualnego problemu. Implementację algorytmu danego wzorami (16)-(20) wykonamy korzystając z informacji wyniesionych z kursu języka C.

Poniżej pokazana jest procedura *rk4_vec*, która wykonuje jeden krok $\vec{s}_i \rightarrow \vec{s}_{i+1}$. Jej argumentami są: aktualna chwila czasowa ($t = t_i$), krok czasowy (dt), ilość zmiennych n w układzie równań, tablica przechowująca aktualny wektor rozwiązań ($\boldsymbol{s} = \vec{s}_i$) oraz wskaźnik do funkcji ($f(t,\boldsymbol{s},\boldsymbol{k})$), która wyznacza wartości wektora pochodnych. W procedurze tworzymy tablice $\boldsymbol{k}_1,\ldots,\boldsymbol{k}_4$ (odpowiedniki wektorów $\vec{k}_1,\ldots,\vec{k}_4$) oraz tablicę pomocniczą $\boldsymbol{w}$. Wynik czyli rozwiązania wektora $\vec{s}_{i+1}$ są zwracane w tablicy $\boldsymbol{s}$.

```
void rk4_vec(  double t, double dt, int n, double *s,
               void (*f)(double, double *, double *)  ){
        #define M 1000
        static double k1[M],k2[M],k3[M],k4[M], w[M];
        int i;

        for(i=0;i<n;i++)w[i]=s[i];
        f(t,w,k1);

        for(i=0;i<n;i++)w[i]=s[i]+dt/2*k1[i];
        f(t+dt/2,w,k2);

        for(i=0;i<n;i++)w[i]=s[i]+dt/2*k2[i];
        f(t+dt/2,w,k3);

        for(i=0;i<n;i++)w[i]=s[i]+dt*k3[i];
        f(t+dt,w,k4);

        for(i=0;i<n;i++)s[i]=s[i]+dt/6*(k1[i]+2*k2[i]+2*k3[i]+k4[i]);
        return;
}
```

Uwaga 1: Tablice $\boldsymbol{k}_j$, $j = 1, 2, 3, 4$ tworzymy z atrybutem *static* co oznacza że tworzone są one tylko podczas pierwszego wywołania funkcji. Po zakończeniu działania funkcji *rk4_vec* nadal istnieją w pamięci komputera, więc podczas kolejnego wywołania są gotowe do użycia (nie marnujemy czasu na kolejną alokację pamięci).
Uwaga 2: Parametr M określa maksymalną ilość równań (zmiennych) w rozwiązywanym układzie.

2.1.2 Procedura implementująca liczenie pochodnych

Procedurę *rk4_vec* możemy używać pod warunkiem że w programie mamy zdefinowaną funkcję wyliczającą pochodne

```
 void pochodne( double t, double *s, double *k){

        deklaracje stałych;

        k[0]= f_1(t,s);
        k[1]= f_2(t,s);
        k[2]= f_3(t,s);
           .
           .
           .
        k[n-1]= f_{n-1}(t,s);

        return;
}
```

Uwaga 1: Funkcję *pochodne* tworzymy oddzielnie dla każdego rozwiązywanego układu bo zmieniają się prawe strony tj. definicje $\vec{f}(t, \vec{s})$.
Uwaga 2: W funkcji musimy określić sposób wyznaczania tylu pochodnych ile mamy w układzie RRZ1 czyli $n - 1$.

2.1.3 Program implementujący RK4

Mając do dyspozycji ogólną procedurę dla RK4 oraz funkcję wyliczającą pochodne możemy je wykorzystać w programie którego celem będzie wyznaczenie rozwiązań w kolejnych chwilach czasu

```
 void main(){

        double t, dt, tmax, *s;
        int i, n, N;
        void (*f)(double, double *, double *);   //wskaźnik do funkcji
```

```
//inicjalizacja parametrów:

        n=...;             //ilość zmiennych w układzie RRZ1
        dt=....;
        tmax=...;
        N=(int)tmax/dt;    //ilość kroków czasowych
        t=0;

        f=pochodne;        //przypisujemy wskaźnik do funkcji
        s=(double *) malloc(n*sizeof(double));   //tablica rozwiązań

//warunki początkowe:
                s[0]=...;
                s[1]=...;
                  .
                  .
                  .
                s[n-1]=...;

//symulacja w czasie:
        for(i=1;i<=N;i++){
                rk4_vec(t, dt, n, s, f);
                t=t+dt;
                //zapis wynikow do pliku
        }
        return;
}
```