Metoda Rungego–Kutty

Witold Bołt

26 lutego 2006

1 Wstęp

W niniejszym opracowaniu będziemy poszukiwać przybliżonego rozwiązania równania:

$$y^{(4)}(t) + (1 + \sqrt{1+t^2})y^{(3)}(t) + (t+1)y^{(2)}(t) + ty'(t) + e^{1+t}y(t) = 1 + t^2$$

z warunkiem początkowym: $y(0) = y'(0) = y^{(2)}(0) = y^{(3)}(0) = 0$, metodą Rungego-Kutty rzędu 2 na przedziale $[0, 2]$.

Przed omówieniem programu rozwiązującego dane zagadnienie omówimy najpierw teorię wykorzystywaną przez program.

2 Podstawy teoretyczne

2.1 Metoda Rungego–Kutty

W tej części zakładamy, że czytelnik zna i rozumie zasady działania metody Eulera rozwiązywania równań różniczkowych zwyczajnych. Metoda Rungego–Kutty jest bowiem rozwinięciem tej metody. Stosujemy tu dokładnie te same oznaczenia co przy opisie metody Eulera[1].

W metodzie Eulera, do wyznaczenia kolejnych punktów korzystaliśmy z przybliżonej wartości pochodnej, wyliczonej za pomocą ilorazu różnicowego (dla ustalonego h). Podobnie postępujemy w metodzie Rungego–Kutty, z tą różnicą, że tutaj aby wyznaczyć wartość w węźle wykonujemy kilka wyliczeń, biorąc pod uwagę również punkty wewnątrz przedziału $[t_j, t_{j+1}]$. W zależności od tego jak wiele dodatkowych punktów bierzemy pod uwagę, możemy mówić o różnym rzędzie metody. Metoda Eulera jest metodą Rungego–Kutty rzędu 1. Metoda Rungego-Kutty rzędu 2 jest opisana przez następujące wzory:

$$\begin{aligned} k_1 &= h \cdot f(t_n, y_n)\,, \\ k_2 &= h \cdot f(t_n + \frac{h}{2}, y_n + \frac{k_1}{2})\,. \end{aligned}$$

Przy czym kolejne punkty iteracji dostajemy korzystając z wzoru:

$$y_{n+1} = y_n + k_2.$$

[1]Patrz opracowanie do zadania z metody Eulera: `http://www.houp.info/mn/euler.tar.bz2`.

W poniższym rozwiązaniu zastosowano właśnie metodę drugiego rzędu. W zastosowaniach praktycznych stosuje się bardzo często metodę rzędu 4, gdzie wyliczamy kolejno wartości k_1, k_2, k_3, k_4 korzystając z wzorów:

$$\begin{aligned} k_1 &= h \cdot f(t_n, y_n)\,, \\ k_2 &= h \cdot f\Big(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\Big)\,, \\ k_3 &= h \cdot f\Big(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\Big)\,, \\ k_4 &= h \cdot f(t_n + h, y_n + k_3)\,, \end{aligned}$$

a główny krok iteracji liczymy wzorem:

$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}.$$

Teoretycznie można mówić też o metodach wyższego rzędu. Nie są one jednak zbyt często stosowane.

Oczywiście metoda daje nam wzory na wartości przybliżone rozwiązania w poszczególnych punktach. Aby otrzymać funkcję będącą przybliżeniem rozwiązania należy zbudować krzywą łamaną (lub w jakiś inny sposób interpolować rozwiązanie przechodzące przez te punkty).

2.2 Równania liniowe wyższego rzędu

Okazuje się, że istnieje prosta metoda „przerobienia" równania liniowego wyższego rzędu na układ równań rzędu pierwszego. Innymi słowy – każde równanie liniowe jest tożsame z układem (jednego lub więcej) układ liniowych rzędu pierwszego.

Rozważmy zagadnienie postaci:

$$\begin{cases} y^{(n)}(t) + p_1(t)y^{(n-1)}(t) + \ldots + p_{n-1}(t)y'(t) + p_n(t)y(t) = b(t) \\ y(t_0) = x_{01},\ y'(t_0) = x_{02}, \ldots, y^{(n-1)}(t_0) = x_{0n} \end{cases}$$

Na początek wykonamy następujące podstawienie:

$$\begin{cases} x_1(t) = y(t) \\ x_2(t) = y'(t) \\ x_3(t) = y''(t) \\ \vdots \\ x_n(t) = y^{(n-1)}(t) \end{cases}$$

No ale wtedy możemy napisać również:

$$\begin{cases} x_1'(t) = x_2(t) \\ x_2'(t) = x_3(t) \\ \vdots \\ x_{n-1}'(t) = x_n(t) \\ x_n'(t) = y_n(t) = b(t) - p_n(t)x_1(t) - p_{n-1}(t)x_2(t) - \cdots - p_1(t)x_n(t) \\ x_j(t_0) = x_{0j}, \text{dla } j = 1, \ldots, n \end{cases}$$

Jest to układ równań liniowych, który można zapisać w postaci macierzowej:

$$\underbrace{\begin{bmatrix} x_1' \\ x_2' \\ \vdots \\ x_{n-1}' \\ x_n' \end{bmatrix}}_{x'(t)} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ & & \vdots & & \\ 0 & \cdots & \cdots & 0 & 1 \\ -p_n & -p_{n-1} & \cdots & \cdots & -p_1 \end{bmatrix}}_{A(t)} \cdot \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix}}_{x(t)} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ b(t) \end{bmatrix}}_{B(t)}$$

Innymi słowy – otrzymaliśmy równanie liniowe pierwszego rzędu, w którym rolę funkcji A i B pełnią teraz odpowiednie macierze:

$$x'(t) = A(t)x(t) + B(t).$$

Oczywiście zgodnie z podstawieniem, poszukiwanym przez nas rozwiązaniem jest pierwsza współrzędna wektora $x(t)$.

3 Rozwiązanie

3.1 Przekształcenie równania

Zgodnie z tym, co mówiliśmy przed chwilą, nasze równanie można przedstawić w postaci równania liniowego pierwszego rzędu. Poniżej przedstawiono wejściowe równanie zapisane w postaci macierzowej:

$$\begin{bmatrix} x_1' \\ x_2' \\ x_3' \\ x_4' \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -e^{1+t} & -t & -(t+1) & -(1+\sqrt{1+t^2}) \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1+t^2 \end{bmatrix}$$

Mając równanie zapisane w takiej postaci, możemy już stosować metodę Rungego–Kutty, przy czym wyrażenie $f(t, x(t))$ należy w tym przypadku rozumieć jako funkcję, która działa ze zbioru liczby rzeczywistych w zbiór $\mathbb{R}^4$ i wyraża prawą stronę, przedstawionego wyżej równania macierzowego.

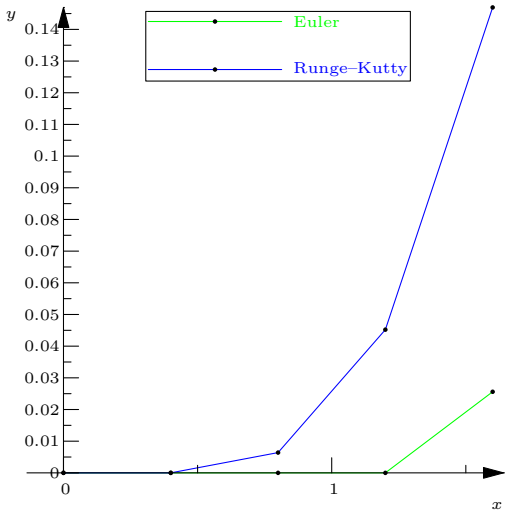
3.2 Program komputerowy realizujący metodę

Program został napisany w języku C. Pobiera on od użytkownika liczbę punktów do wyliczenia. Generuje przybliżone rozwiązanie metodą Eulera oraz Rungego–Kutty rzędu 2 a następnie rysuje wykresy (za pomocą pakietu Asymptote). Poniżej zamieszczono funkcję wyliczającą kolejne punkty z metody Rungego–Kutty. Poza nią, w programie jest też podobna funkcja wyliczająca punkty metodą Eulera, funkcja główna obsługująca wejście i wyjście oraz kilka funkcji pomocniczych do operowania na wektorach *vec*4 oraz funkcja dające kolejne wartości prawej strony naszego zagadnienia.

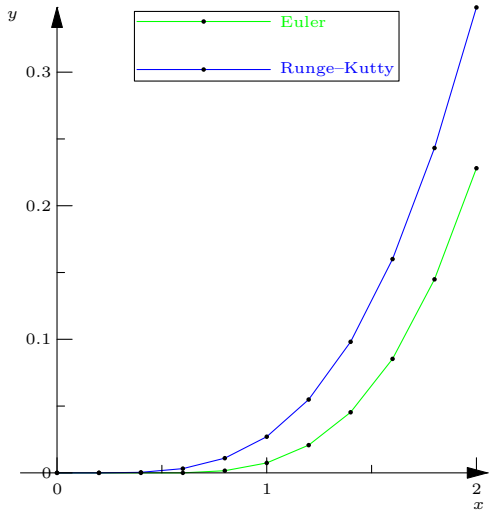
```
// funkcja generuje kolejne punkty metody Rungego--Kutty
// a - punkt początkowy
// x_0 - warunek początkowy
// h - skok :)
// plot - plik wynikowy
// name - nazwa zmiennej w pliku do której zapisać wyniki
void get_rk_points (double a, vec4 x_0, double h, FILE *plot,
                    const char name[32])
{
    double t=a;
    vec4 x;
    copy(x_0,x);
    vec4 x2;
    vec4 k1,k2;

    fprintf(plot,"pair[] %s = {", name);

    while(t<=2.0) {
        fprintf(plot,"(%f,%f),",t,x[0]);
        copy(x,x2);

        f(t,x,k1);
        add(x,0.5*h,k1);

        f(t+h*0.5,x,k2);

        add(x2,h,k2);
        copy(x2,x);
        t+=h;
    }

    fflush(plot);
    fseek(plot,-1,SEEK_CUR);
    fprintf(plot,"};\n");
}
```

Pełny kod źródłowy programu (wraz z komentarzami) można ściągnąć ze strony: `http://www.houp.info/mn/rk.tar.bz2`.

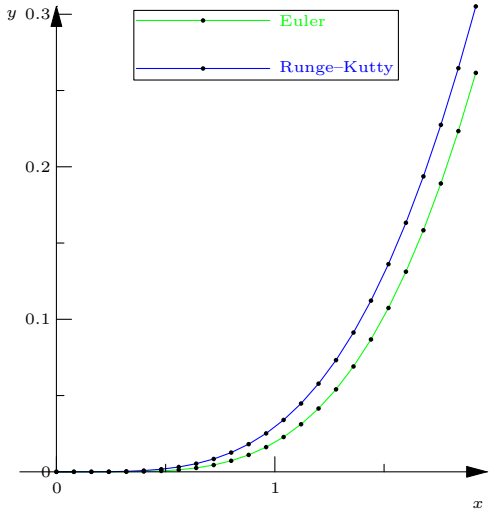
Poniżej przedstawiono wykresy wygenerowane przez program dla $n = 5, 10, 25, 50$:



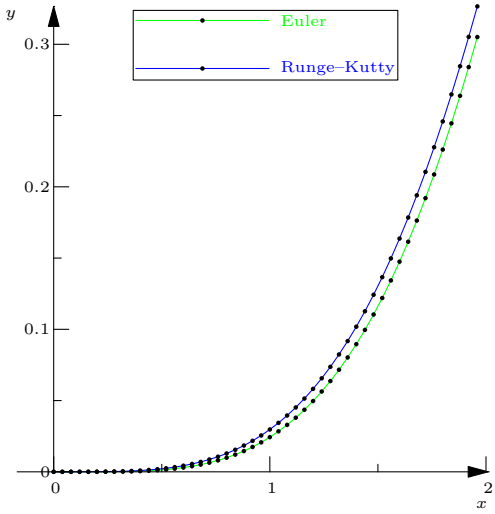
$n = 5$



$n = 10$



$n = 25$



$n = 50$

Źródło:

1. Bołt W., Tartas K.: *Równania różniczkowe*, notatki do wykładu dr hab. A. Augustynowicza, Gdańsk 2004/2005 – `http://www.houp.info/rr/`.

2. Kochański P., Kortyka P.: *Sposoby rozwiązywania prostych równań różniczkowych zwyczajnych*, Dla Szkoły Nauk Ścisłych, Warszawa 1999 – `http://snsinfo.ifpan.edu.pl/skrypty/rown/`