Szymon Ociepka

Data wykonania: 22.05.2024

Metoda Eulera

Rozwiązywanie równań różniczkowych

Metody numeryczne

1. Wprowadzenie teoretyczne

Metoda Eulera to jedna z najprostszych metod numerycznych stosowanych do rozwiązywania równań różniczkowych. Została nazwana na cześć szwajcarskiego matematyka, Leonarda Eulera.

$$\frac{dy}{dx}=f(x,y),\ y(x_0)=y_0 \tag{1}$$

1. Równanie dla każdego punktu **(x, y)** określa nachylenie stycznej do rozwiązania które przechodzi przez dany punkt.

2. Kierunek naszej stycznej zmienia się w sposób ciągły od punktu do punktu.

Metoda Eulera polega na przybliżaniu rozwiązania równania różniczkowego za pomocą szeregów kroków. Rozpoczynamy od pewnego punktu początkowego (x_0, y_0), a następnie wykonujemy serię małych kroków do przodu w czasie.

W każdym kroku, staramy się przybliżyć pochodną funkcji y za pomocą różnicy skończonej:

$$y_{n+1}=y_n+h\cdot f(x_n,y_n),\ n=0,1,2,\ldots,N \tag{2}$$

Liczba kroków N wynosi:

$$N=\frac{(b-x_0)}{h} \tag{3}$$

gdzie:

h – to rozmiar kroków,

x_0 – początek badanego zakresu

b – koniec badanego zakresu

y_0 – warunek początkowy

Ważne jest, aby pamiętać, że metoda Eulera jest tylko przybliżeniem. Dokładność metody Eulera zależy od wielkości kroku **h.** Pomimo swojej prostoty, metoda Eulera jest podstawą dla wielu bardziej zaawansowanych metod numerycznych rozwiązywania równań różniczkowych. Jest to zatem ważne narzędzie w arsenale każdego matematyka i inżyniera.

2. Opis implementacji numerycznej

```c
double f(double x, double y) {
    return y; //dana funkcja, w tym przypadku f(x) = y
}
```

Fragment kodu 2.1: funkcja równania różniczkowego

Ta funkcja reprezentuje funkcję w równaniu różniczkowym. W tym przypadku f(x) = y.

```c
for (int i = 1; i < N; i++) {
    y[i] = y[i - 1] + h * f(x[i - 1], y[i - 1]);
    x[i] = x[i - 1] + h;
    printf("y%d = %lf\n", i, y[i]);
}
```

Fragment kodu 2.2: implementacja metody Eulera

Pętla **for** implementuje metodę Eulera. Dla każdego kroku oblicza nową wartość **y[i]** na podstawie poprzedniej wartości **y[i – 1]** i obecnej wartości **x[i – 1]**. Następnie oblicza nową wartość **x[i]** dodając krok **h** do poprzedniej wartości **x[i – 1]**. Następnie **printf** wyświetla obliczoną wartość **y[i]** dla każdego kroku.

3. Testy kodu numerycznego

Test I

Dane:

f(x, y) = y, $x_0 = 0$; $y_0 = 1$; b = 0.4; h = 0.1

Wynik Wolfram Alpha:

$y_1 = 1.1$
$y_2 = 1.21$
$y_3 = 1.331$
$y_4 = 1.4641$

Wynik zaimplementowanego programu:

```
y1 = 1.100000
y2 = 1.210000
y3 = 1.331000
y4 = 1.464100
```

Fragment konsoli 3.1: wynik programu dla testu I

Test II

Dane:

f(x, y) = $2 \cdot x \cdot e^{x}$,

$x_0 = 0$; $y_0 = 0$; b = 2; h = 0.1

Wynik Wolfram Alpha:

y_1 = 0
y_2 = 0.0221034
y_3 = 0.0709595
y_4 = 0.151951
y_5 = 0.271297
y_6 = 0.436169
y_7 = 0.654823
y_8 = 0.936749
y_9 = 1.29284
y_{10} = 1.73556
y_{11} = 2.27922
y_{12} = 2.94014
y_{13} = 3.73696
y_{14} = 4.69098
y_{15} = 5.82644
y_{16} = 7.17094
y_{17} = 8.75592
y_{18} = 10.6171
y_{19} = 12.7949
y_{20} = 15.3356

Wynik zaimplementowanego programu:

```
y1 = 0.000000
y2 = 0.022103
y3 = 0.070960
y4 = 0.151951
y5 = 0.271297
y6 = 0.436169
y7 = 0.654823
y8 = 0.936749
y9 = 1.292835
y10 = 1.735564
y11 = 2.279220
y12 = 2.940137
y13 = 3.736965
y14 = 4.690982
y15 = 5.826438
y16 = 7.170945
y17 = 8.755915
y18 = 10.617057
y19 = 12.794930
y20 = 15.335570
```

Fragment konsoli 3.2: wynik programu dla testu II

Test III

Dane:

f(x, y) = $e^{x-1} - 2 \cdot x$,

$x_0 = 0$; $y_0 = -1$; b = 1; h = 0.2

Wynik Wolfram Alpha:

y_1 = -0.8
y_2 = -0.716254
y_3 = -0.74219
y_4 = -0.872428
y_5 = -1.10256

Wynik zaimplementowanego programu:

```
y1 = -0.800000
y2 = -0.716254
y3 = -0.742190
y4 = -0.872428
y5 = -1.102562
```

Fragment konsoli 3.3: wynik programu dla testu III

Test IV

Dane:

f(x, y) = $\frac{2 \cdot x + y \cdot \sin(x)}{\cos(x)}$, $x_0 = 0$; $y_0 = 0$; b = 2; h = 0.2

Wynik Wolfram Alpha:

y_1 = 0
y_2 = 0.0816271
y_3 = 0.262242
y_4 = 0.588915
y_5 = 1.16949
y_6 = 2.27409
y_7 = 4.76861
y_8 = 13.5929
y_9 = -101.389
y_{10} = -17.6421

Wynik zaimplementowanego programu:

```
y1 = 0.000000
y2 = 0.081627
y3 = 0.262242
y4 = 0.588915
y5 = 1.169492
y6 = 2.274094
y7 = 4.768615
y8 = 13.592944
y9 = -101.389401
y10 = -17.642085
```

Fragment konsoli 3.4: wynik programu dla testu IV

Test V

Dane:

f(x, y) = $2 \cdot y$, $x_0 = 0$; $y_0 = 1$; b = 2; h = 0.25

Wynik Wolfram Alpha:

y_1 = *1.5*
y_2 = *2.25*
y_3 = *3.375*
y_4 = *5.0625*
y_5 = *7.59375*
y_6 = *11.3906*
y_7 = *17.0859*
y_8 = *25.6289*

Wynik zaimplementowanego programu:

```
y1 = 1.500000
y2 = 2.250000
y3 = 3.375000
y4 = 5.062500
y5 = 7.593750
y6 = 11.390625
y7 = 17.085938
y8 = 25.628906
```

Fragment konsoli 3.5: wynik programu dla testu V

4. Analiza wyników

W celu głębszej analizy powstała tabela, w której porównane zostały wyniki zaimplementowanych metod z wynikami programu „Wolfram Alpha” na podstawie powyższych testów.

Tabela 1: wyniki Testu I

wartości	**Wolfram Alpha**	**Program**
x_1	1.1	1.1
x_2	1.21	1.21
x_3	1.331	1.331
x_4	1.4641	1.4641

Tabela 2: wyniki Testu II

wartości	**Wolfram Alpha**	**Program**
x_1	0	0
x_2	0.0221034	0.022103
x_3	0.0709595	0.07096
x_4	0.151951	0.151951
x_5	0.271297	0.271297
x_6	0.436169	0.436169
x_7	0.654823	0.654823
x_8	0.936749	0.936749
x_9	1.29284	1.292835
x_{10}	1.73556	1.735564
x_{11}	2.27922	2.27922
x_{12}	2.94014	2.940137
x_{13}	3.73696	3.736965
x_{14}	4.69098	4.690982
x_{15}	5.82644	5.826438

x_{16}	7.17094	7.170945
x_{17}	8.75592	8.755915
x_{18}	10.6171	10.617057
x_{19}	12.7949	12.79493
x_{20}	15.3356	15.33557

Tabela 3: wyniki Testu III

wartości	**Wolfram Alpha**	**Program**
x_1	-0.8	-0.8
x_2	-0.716254	-0.716254
x_3	-0.74219	-0.74219
x_4	-0.872428	-0.872428
x_5	-1.10256	-1.102562

Tabela 4: wyniki Testu IV

wartości	**Wolfram Alpha**	**Program**
x_1	0	0
x_2	0.0816271	0.0816271
x_3	0.262242	0.262242
x_4	0.588915	0.588915
x_5	1.16949	1.169492
x_6	2.27409	2.274094
x_7	4.76861	4.768615
x_8	13.5929	13.592944
x_9	-101.389	-101.389401
x_{10}	-17.6421	-17.642085

Tabela 5: wyniki Testu V

wartości	**Wolfram Alpha**	**Program**
x_1	*1.5*	*1.5*
x_2	*2.25*	*2.25*
x_3	*3.375*	*3.375*
x_4	*5.0625*	*5.0625*
x_5	*7.59375*	*7.59375*
x_6	*11.3906*	*11.390625*
x_7	*17.0859*	*17.085938*
x_8	*25.6289*	*25.628906*

Dzięki analizie wyników można stwierdzić, że przedstawiona metoda została zaimplementowana poprawnie. Wyniki uzyskane za pomocą zaimplementowanego programu są praktycznie identyczne względem wyników uzyskanych za pomocą programu Wolfram Alpha. Jednakże, występuje parę przypadków gdzie zauważono minimalne rozbieżności, które mogą wynikać z precyzji elementów - liczby zmiennoprzecinkowe o dużej liczbie miejsc po przecinku, zainicjowana stała Eulera używana w programie.
W trakcie obliczeń numerycznych, między innymi w metodzie Eulera obliczania równań różniczkowych, mogą wystąpić błędy zaokrąglenia wynikające z ograniczonej precyzji liczb zmiennoprzecinkowych. Te błędy mogą się kumulować w trakcie obliczeń, prowadząc do zaokrąglania wyników.

5. Wnioski

W tym sprawozdaniu przedstawiono oraz omówiono metodę Eulera służącą do rozwiązywania równań różniczkowych. Pokazano oraz wytłumaczono jej prostą implementację w języku c++. Z analizy przeprowadzonych testów wynika że metoda Eulera jest skutecznym narzędziem do rozwiązywania równań różniczkowych. Testy wykazały, że zastosowana implementacja jest szczególnie efektywna pod względem uzyskanych wyników, natomiast mogą wystąpić minimalne błędy zaokrąglenia wynikające z ograniczonej precyzji liczb zmiennoprzecinkowych.

6. Źródła

- prezentacja z zajęć „lab 11.pdf"
- Beata Pańczyk, Edyta Łukasik, Jan Sikora, Teresa Guziak, Metody numeryczne w przykładach, Politechnika Lubelska, 2012
- www.wolframalpha.com/input?i=use+Euler+method+y%27+%3D+2y%2C+y%280%29+%3D+1%2C+from+0+to+2%2C+h+%3D+0.25