Metody numeryczne
Projekt nr 1

Jan Skwarek
numer albumu: 305734

17 grudnia 2021

1 Treść zadania

Metoda Simpsona obliczania przybliżonej wartości całki $\int_a^b f(x)\mathrm{d}x$, gdzie

$$f(x) = \sum_{k=1}^{n} a_k \sin kx$$

Do obliczania wartości $f(x)$ zastosować metodę Goertzela.

2 Opis metody

2.1 Algorytm Goertzela

W programie, do obliczania wartości funkcji $f(x)$, została użyta metoda Goertzela, znana także Algorytmem Goertzela. Świetnie nadaje się ona do liczenia wartości funkcji typu $f(x) = \sum_{k=1}^{n} a_k \sin kx$, czy $f(x) = \sum_{k=1}^{n} a_k \cos kx$. Przyjmijmy, że jest dany następujący wielomian:

$$w(\lambda) = \sum_{n=0}^{N} a_n \lambda^n,$$

gdzie $a_n \in \mathbb{C}, n = 0, 1, ..., N$, oraz punkt $z = x + iy \in \mathbb{C}$

Za pomocą Algorytmu Goertzela obliczymy właśnie $w(z)$. Najpierw wielomian startowy $w(\lambda) = \sum_{n=0}^{N} a_n \lambda^n$ należy podzielić przez trójmian $(\lambda - z)(\lambda - \overline{z}) = \lambda^2 - \hat{p}\lambda - \hat{q}$. Współczynniki tego trójmianu to $\hat{p} = 2x$ oraz $\hat{q} = -|z|^2$. Możemy to zapisać w ten sposób:

$$w(\lambda) = (\lambda - z)(\lambda - \overline{z}) \sum_{n=2}^{N} b_n \lambda^{n-2} + b_0 + b_1 \lambda$$

Widzimy więc, że z tego wzoru wynika, że $w(z) = b_0 + b_1 z$. Oznacza to, że wartość wielomianu w w punkcie z jest po prostu równa wartości reszty z dzielenia tego wielomianu przez trójmian $\lambda^2 - \hat{p}\lambda - \hat{q}$ w punkcie z. Cała idea algorytmu Goertzela polega właśnie na wyznaczeniu wartości współczynników $b_n, b_{n-1}, ..., b_1, b0$, a następnie policzeniu wartości tej reszty.

Zauważmy jedną ciekawą rzecz. Na początku tej podsekcji napisałem, że Algorytm Goertzela świetnie nadaje się do liczenia wartości funkcji typu $f(x) = \sum_{k=1}^{n} a_k \sin kx$, czy $f(x) = \sum_{k=1}^{n} a_k \cos kx$. Zauważmy, że gdy podstawimy $z = \cos t + i \sin t$, to druga z tych funkcji będzie się równała po prostu $Rew(z)$, natomiast pierwsza (to właśnie ją musimy obliczyć w zadaniu) $Imw(z)$. Bardzo uprości to nasz algorytm, ponieważ nie będziemy musieli wcale się przejmować częścią rzeczywistą $w(z)$.

2.2 Metoda Simpsona

Zajmijmy się teraz drugą z metod, których użyłem do wykonania programu. Metoda Simpsona posłużyła nam do obliczania przybliżonej wartości całki $\int_a^b f(x)\mathrm{d}x$. Metoda Simpsona zwana jest również metodą parabol - stosujemy je jako przybliżenie obliczając sumy wycinków obszarów pod parabolą. Znając wartości y_0, y_1, y_2 funkcji $f(x)$ w 3 punktach x_0, x_1, x_2, (przy czym $x_2 - x_1 = x_1 - x_0 = h$), przybliża się funkcję wielomianem Lagrange'a i całkując w przedziale $[x_0, x_2]$, otrzymuje przybliżoną wartość całki:

$$\int_{x_0}^{x_2} f(x)\mathrm{d}x \approx \frac{h}{3}(y_0 + 4y_1 + y_2)$$

Na tym w skrócie polega metoda Simpsona. Na potrzeby metod numerycznych (po wielu nieoczywistych przekształceniach i założeniach, które nie są istotą tego zadania) wzór wyliczania przybliżonej wartości całki oznaczonej za pomocą metody Simpsona zapisujemy następująco:

$$\int_{x_p}^{x_k} f(x)\mathrm{d}x \approx \frac{x_k - x_p}{6n}(\sum_{i=1}^{n} f_{i-1} + \sum_{i=1}^{n} f_i + \sum_{i=1}^{n} f_{ti})$$

Do obliczeń komputerowych stosujemy jednak efektywniejszy wzór otrzymywania powyższych sum:

$$\int_{x_p}^{x_k} f(x)\mathrm{d}x \approx \frac{x_k - x_p}{6n}(f_0 + f_n + 2\sum_{i=1}^{n-1} f_i + 4\sum_{i=1}^{n} f_{ti})$$

Zgodnie z tym wzorem będę wyznaczać wartość całki w swoim algorytmie.

3 Opis programu obliczeniowego

Mój program obliczeniowy składa się z dwóch funkcji. Opiszę każdą w tej sekcji.

3.1 metodaGoertzela

Funkcja ta przyjmuje następujące argumenty:

- *wspolczynniki* - jest to wektor współczynników wielomianu, musimy pamiętać, że współczynniki iterowane są od n-tego do 0, a więc $wspolczynniki(1) = a_n$. Z kolei ostatni współczynnik w wektorze będzie odpowiadał a_0, przed ostatni a_1 i tak dalej
- t - jest to liczba, dla której obliczamy wartość funkcji

Funkcja zwraca wektor *wartosc*, który jest obliczoną wartością funkcji $f(x) = \sum_{k=1}^{n} a_k \sin kx$ dla konkretnego x (t w naszym algorytmie) i dla pewnych współczynników. Oto zaproponowany przeze mnie w języku programowania MATLAB algorytm:

```
1  % (C) Jan Skwarek
2  function wartosc = metodaGoertzela(wspolczynniki, t)
3      x = cos(t);
4      y = sin(t);
5      len = length(wspolczynniki);
6      % wspolczynniki w tablicy od ostatniego do zerowego
7      p = 2 * x;
8      q = -(x * x + y * y);
9      % w matlabie literujemy od jedynki wiec a_1 bedzie naszym przed
10     % ostatnim wspolczynnikiem
11     if len == 0
12         % lista wspolczynnikow jest pusta
13         error("Lista wspolczynnikow wielomianu jest pusta!")
14     end
15     if len == 1
16         % nie wchodzimy do petli, b_1 to b_n+1, a wiec zero
17         wartosc = 0 * y;
18         return
```

```
    end
    if len == 2
        % ponownie nie wchodzimy do petli, b_n = a_n = b_1
        wartosc = wspolczynniki(1) * y;
        return
    end
    % petla zostala zaprojektowana w ten sposob, ze liczymy w niej dwie
    % kolejne wartosci, a pozniej przechodzimy do ostatniej linijki, w
    % ktorej obliczamy wartosc ostateczna na podstawie tych wyliczanych w
    % petli. Oznacza to, ze nie mozemy wejsc do petli, gdy wiemy ze wykona
    % sie tylko raz. Powinnismy zamiast tego wczesniej zakonczyc program i
    % zwrocic poprawna wartosc. Dlatego wiec powstaly dwa kolejne przypadki
    if len == 3
        wartosc = y * (wspolczynniki(2) + p * wspolczynniki(1));
        return
    end
    if len == 4
        wartoscTmp1 = wspolczynniki(2) + p * wspolczynniki(1);
        wartosc = y * (wspolczynniki(3) + p * wartoscTmp1 + q * wspolczynniki
            (1));
        return
    end
    % wchodzimy do petli, wykona sie ona conajmniej 2 razy (moze sie
    % wykonac rowniez raz, ale nie obliczymy w niej ostatecznej wartosci)
    wartoscTmp1 = 0;
    wartoscTmp2 = wspolczynniki(1);
    % obliczamy wartosci dla kolejnych wspolczynnikow zgodnie z algorytmem
    % Goertzela
    for iterator = 2 : 2 : len - 2
        wartoscTmp1 = wspolczynniki(iterator) + p * wartoscTmp2 + q *
            wartoscTmp1;
        wartoscTmp2 = wspolczynniki(iterator + 1) + p * wartoscTmp1 + q *
            wartoscTmp2;
    end
    % z uwagi na specyfike funkcji powyzszy algorytm pomija rzeczywista
    % czesc wyniku w(z), ktora wyszlaby przy standardowym uzyciu algorytmu
    % Goertzele
    if mod(len, 2) == 1
        wartosc = y * (wspolczynniki(len - 1) + p * wartoscTmp2 + q *
            wartoscTmp1);
        return
    end
    wartosc = y * wartoscTmp2;
```

3.2 metodaSimpsona

Funkcja ta przyjmuje następujące argumenty:

- *wspolczynniki* podobnie jak w poprzedniej funkcji - jest to wektor współczynników wielomianu, musimy pamiętać, że współczynniki iterowane są od n-tego do 0, a więc *wspolczynniki*(1) = a_n. Z kolei ostatni współczynnik w wektorze będzie odpowiadał a_0, przed ostatni a_1 i tak dalej
- a - początek przedziału całkowania, $a \in R$
- b - koniec przedziału całkowania, $b \in R$
- n - liczba punktów przedziałowych, $n \in N$

Funkcja zwraca wektor *wynik*, który jest przybliżoną wartością całki $\int_a^b f(x)\mathrm{d}x$. Oto zaprojektowany przeze mnie w języku programowania MATLAB algorytm:

```
% (C) Jan Skwarek
function wynik = metodaSimpsona(wspolczynniki, a, b, n)
    % wszystkie "trudne" przypadki zwiazane z iloscia wspolczynnikow
    % rozpatrujemy juz w funkcji metodaGoertzela
    wynik = 0;
    % s_t to suma wartosci funkcji w punktach srodkowych, s_t \in R
    s_t = 0;
    % dx to odleglosc miedzy dwoma sasiednimi punktami podzialowymi, dx \in
    % R
    dx = (b - a) / n;
    % standardowa iteracyjna implementacja metody Simpsona
    % iterator to licznik punktow podzialowych, iterator \in N
    for iterator = 1 : n
        % x to pozycja punktu podzialowego, x \in R
        x = a + iterator * dx;
        s_t = s_t + metodaGoertzela(wspolczynniki, x - (dx / 2));
        if iterator < n
            wynik = wynik + metodaGoertzela(wspolczynniki, x);
        end
    end
    % wyznaczamy wartosc calki zgodnie z zaproponowanym wzorem
    wynik = (dx / 6) * (metodaGoertzela(wspolczynniki, a) + metodaGoertzela(
       wspolczynniki, b) + 2 * wynik + 4 * s_t);
```

4 Przykłady obliczeniowe

Wszystkie przykłady powinny się wykonywać w czasie bliskim jednej sekundy nawet na bardzo starych komputerach.

4.1 Przykład 1 - pusty wektor ze współczynnikami

Kilka początkowych przykładów będzie skupiało się na tzw. corner cases - nie wchodzimy do pętli, wektor współczynników jest pusty, zawiera zbyt mało wartości i tym podobne.

```
>> metodaSimpsona([], 2, 10, 3000)
Error using metodaGoertzela (line 13)
Lista wspolczynników wielomianu jest pusta!

Error in metodaSimpsona (line 16)
        s_t = s_t + metodaGoertzela(wspolczynniki, x - (dx / 2));
```

Jak widać, w inpucie podaliśmy funkcji pusty wektor ze współczynnikami, przedział całkowania o początku 2, a końcu 10 oraz 3000 punktów podziałowych. Program zachował się prawidłowo - wyrzucił nam błąd informujący o tym, że lista współczynników jest pusta.

4.2 Przykład 2 - wektor jednoelementowy

```
>> metodaSimpsona([1], 2, 10, 3000)

ans =

     0
```

W inpucie podaliśmy funkcji podobne dane jak w poprzednim przykładzie, z tym że tym razem wektor *wspolczynniki* był jednoelementowy, zawierał 1. Oznacza to, że nasze $a_0 = 1$ (jest to nasz jedyny współczynnik).

$$f(x) = \sum_{k=0}^{0} a_k \sin kx = 0 \qquad a_k = [1_{k=0}]$$

Wartość funkcji to oczywiście zero. Suma zawiera tylko jeden element, który jest iloczynem zerowego współczynnika i zera.

$$\int_2^{10} 0\mathrm{d}x = 0$$

Całka jest oczywiście również równa zeru. Program zachował się prawidłowo.

4.3 Przykład 3 - wektor dwuelementowy

```
>> metodaSimpsona([2, 1], 2, 10, 3000)

ans =

   0.845849385058634
```

W inpucie podaliśmy funkcji podobne dane jak w poprzednim przykładzie, z tym że tym razem wektor *wspolczynniki* był dwuelementowy. Oznacza to, że nasze $a_0 = 1$ oraz $a_1 = 2$. Pamiętamy, że lista *wspolczynniki* zawiera współczynniki iterowane od n do 0, stąd takie, a nie inne ułożenie współczynników w wejściowym wektorze tzw. "od końca".

$$f(x) = \sum_{k=0}^{1} a_k \sin kx = 2\sin x \qquad a_k = [1_{k=0}, 2_{k=1}]$$

$$\int_2^{10} 2\sin x\mathrm{d}x \approx 0.845849385058620$$

Widać więc, że błąd funkcji przy wielomianach co najwyżej pierwszego stopnia jest praktycznie zerowy (tak mała różnica mogła powstać z wielu powodów, chociażby ze specyfiki programu, którego używam do liczenia dokładnych wartości).

```
>> metodaSimpsona([2, 1], 2, 10, 1000000)

ans =

   0.845849385058621
```

Przy milionie punktów podziałowych wyniki są niemal identyczne.

```
    >> metodaSimpsona([2, 1], 2, 10, 10)

ans =

   0.845972014174839
```

Przy zaledwie 10 punktach podziałowych istnieje już pewien niepomijalny błąd.

4.4 Przykład 4 - wektor trzyelementowy

```
>> metodaSimpsona([3, 2, 1], 2, 10, 300000)

ans =

  -0.746739138956921
```

$$f(x) = \sum_{k=0}^{2} a_k \sin kx = 2\sin x + 3\sin 2x \qquad a_k = [1_{k=0}, 2_{k=1}, 3_{k=2}]$$

$$\int_2^{10} (2\sin x + 3\sin 2x)\mathrm{d}x \approx -0.746739138956886$$

Jak widać, przy 300000 punktach podziałowych błąd jest minimalny, wręcz pomijalny.

```
>> metodaSimpsona([3, 2, 1], 2, 10, 1000000)

ans =

  -0.746739138956942
```

Przy milionie wynik jest jeszcze dokładniejszy.

```
>> metodaSimpsona([3, 2, 1], 2, 10, 10)

ans =

  -0.750536502743748
```

Przy 10 punktach podziałowych ponownie widać już niepomijalny błąd.

4.5 Przykład 5 - wektor czteroelementowy

```
>> metodaSimpsona([4, 3, 2, 1], 2, 10, 3000)

ans =

   0.327819310061252
```

$$f(x) = \sum_{k=0}^{3} a_k \sin kx = 2\sin x + 3\sin 2x + 4\sin 3x \qquad a_k = [1_{k=0}, 2_{k=1}, 3_{k=2}, 4_{k=3}]$$

$$\int_2^{10} (2\sin x + 3\sin 2x + 4\sin 3x)\mathrm{d}x \approx 0.327819310060157$$

Mamy do czynienia już z wielomianami trzeciego stopnia. Algorytm ponownie nas nie zawiódł. Błąd jest znowu pomijalnie mały przy jedynie 3000 punktach podziałowych. Sprawdźmy, jak dokładny wynik otrzymamy dla miliona punktów przedziałowych.

```
>> metodaSimpsona([4, 3, 2, 1], 2, 10, 1000000)

ans =

   0.327819310060246
```

Wynik jest praktycznie idealny.

```
>> metodaSimpsona([4, 3, 2, 1], 2, 10, 10)

ans =

   0.338899132689723
```

Natomiast przy 10 punktach podziałowych błąd jest już znaczący.

4.6 Przykład 6 - wektor pięcioelemntowy

```
>> metodaSimpsona([5, 4, 3, 2, 1], 2, 10, 3000)

ans =

   0.979616844868748
```

$$f(x) = \sum_{k=0}^{4} a_k \sin kx = 2\sin x + 3\sin 2x + 4\sin 3x + 5\sin 4x \qquad a_k = [1_{k=0}, 2_{k=1}, 3_{k=2}, 4_{k=3}, 5_{k=4}]$$

$$\int_2^{10} (2\sin x + 3\sin 2x + 4\sin 3x + 5\sin 4x)\mathrm{d}x \approx 0.97961684486472$$

Program ponownie nas nie zawiódł. Błąd jest pomijalny.

```
>> metodaSimpsona([5, 4, 3, 2, 1], 2, 10, 10)

ans =

    1.0243
```

Przy dziesięciu punktach podziałowych błąd już nam znacząco urósł względem poprzednich przykładów.

4.7 Przykład 7 - wektor sześcioelementowy

```
>> metodaSimpsona([6, 5, 4, 3, 2, 1], 2, 10, 3000)

ans =

  -1.185228224237297
```

$$f(x) = \sum_{k=0}^{5} a_k \sin kx = 2\sin x + 3\sin 2x + 4\sin 3x + 5\sin 4x + 6\sin 5x \qquad a_k = [1_{k=0}, 2_{k=1}, 3_{k=2}, 4_{k=3}, 5_{k=4}, 6_{k=5}]$$

$$\int_2^{10} (2\sin x + 3\sin 2x + 4\sin 3x + 5\sin 4x + 6\sin 5x)\mathrm{d}x \approx -1.185228224217562$$

Błąd znów jest pomijalny. W następnym przykładzie zwiększymy nieco liczbę elementów w wektorze (będziemy działać na wielomianach znacznie wyższego stopnia).

```
>> metodaSimpsona([6, 5, 4, 3, 2, 1], 2, 10, 10)

ans =

   -1.4896
```

Przy dziesięciu punktach podziałowych błąd ponownie urósł względem poprzednich przykładów. Aby wyniki były jakkolwiek sensowne, należy używać większej liczby punktów podziałowych.

4.8 Przykład 8 - wektor dziewięcioelementowy

```
>> metodaSimpsona([9, 8, 7, 6, 5, 4, 3, 2, 1], 2, 10, 3000)

ans =

  -0.610286996420571
```

$$f(x) = \sum_{k=0}^{8} a_k \sin kx = 2\sin x + 3\sin 2x + 4\sin 3x + 5\sin 4x + 6\sin 5x + 7\sin 6x + 8\sin 7x + 9\sin 8x$$

$$a_k = [1_{k=0}, 2_{k=1}, 3_{k=2}, 4_{k=3}, 5_{k=4}, 6_{k=5}, 7_{k=6}, 8_{k=7}, 9_{k=8}]$$

$$\int_2^{10} (2\sin x + 3\sin 2x + 4\sin 3x + 5\sin 4x + 6\sin 5x + 7\sin 6x + 8\sin 7x + 9\sin 8x)\mathrm{d}x \approx -0.61028699635605$$

Błąd jest ponownie pomijalnie mały. Spójrzmy jednak co się stanie, gdy wybierzemy mniejszą liczbę punktów przedziałowych.

```
>> metodaSimpsona([9, 8, 7, 6, 5, 4, 3, 2, 1], 2, 10, 10)

ans =

  17.418775200531535
```

Widzimy już, że wartości są zupełnie inne. Przy wielomianach takiego stopnia musimy już używać większej liczby punktów przedziałowych.

5 Analiza wyników obliczeniowych

Podsumowując powyższe rozważania, metoda Simpsona (z funkcją wyliczaną algorytmem Goertzela) daje bardzo poprawne wyniki (z praktycznie zerowym błędem), jeżeli użyjemy odpowiedniej ilości punktów podziałowych. Jak widać, przy wielomianach stopnia 4 i większego błąd jest już znaczny przy mniejszej ilości punktów podziałowych. Ostatni, w teorii najbardziej ekstremalny przykład, dawał przy 10 punktach podziałowych wynik kompletnie innego rzędu (!!!). Należy jednak podkreślić, że algorytm jest na tyle szybki, że spokojnie może liczyć wielomiany wyższego stopnia przy nawet kilkuset tysięcy punktów podziałowych. Wiemy, że kwadratura Simpsona jest dokładna dla wszystkich wielomianów stopnia 3, a więc dla wektorów maksymalnie czteroelementowych. Powyższe obserwacje zdają się ten fakt niejako podkreślać, wyniki są w stanie osiągnąć bardzo dobrą dokładność przy niewielu punktach podziałowych - później, aby wynik był bliski, a nawet prawie idealny wzorcowemu - jesteśmy zmuszeni znacznie zwiększyć liczbę punktów podziałowych.

6 Źródła

- Notatki do Metod Numerycznych autorstwa dr. Iwony Wróbel
- `https://eduinf.waw.pl/inf/alg/004_int/0004.php`
- `https://pl.wikipedia.org/wiki/Metoda_Simpsona`