

Arytmetyka zmiennopozycyjna, zadanie i algorytm numeryczny

Dorota Dąbrowska, UKSW

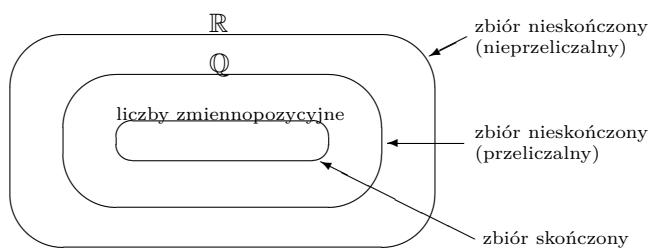
2017/18

Spis treści

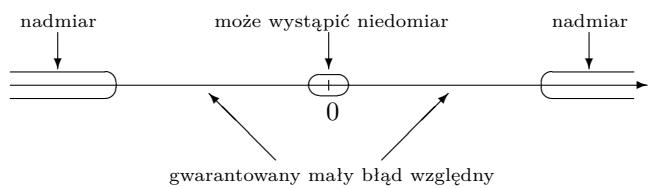
1 Arytmetyka zmiennopozycyjna	2		
1.1 Standardy IEEE 754 i 854	2	2.1 Czym zajmuje się analiza numeryczna	16
1.2 Reprezentacja liczb rzeczywistych	3	2.2 Historia	16
1.3 Zbiór wartości zmiennopozycyjnych	3	2.3 Po co uczymy się metod numerycznych	16
1.4 Reprezentacje standardowe	4	2.4 O błędach zaokrągleń	17
1.5 Rozmieszczenie liczb zmiennopozycyjnych na osi liczbowej	4	2.5 Zadania numeryczne	17
1.6 Formaty liczb zmiennopozycyjnych	6	2.6 Uwarunkowanie zadania	18
1.7 Kodowanie binarnych formatów przenośnych	7	2.7 Algorytmy numerycznie poprawne	20
1.7.1 Sposób kodowania	7	2.8 Algorytmy numerycznie stabilne	22
1.7.2 Dowód poprawności kodowania	8	2.9 Efektywność algorytmu	23
1.8 Reprezentacja liczb rzeczywistych w arytmetyce zmiennopozycyjnej	8		
1.8.1 Zaokrąglenie do najbliższej z wyborem parzystej cyfry	8	Literatura	
1.8.2 Zaokrąglenie do najbliższej z wyborem większego modułu	9	[1] Dryja M., Jankowscy J. i M., <i>Przegląd metod i algorytmów numerycznych</i> . cz. 2, WNT, Warszawa 1982.	
1.8.3 Zaokrąglanie w kierunku $+\infty$	9	[2] Fortuna, Z., Macukow, B., Wąsowski, J., <i>Metody numeryczne</i> . WNT, Warszawa 1993.	
1.8.4 Zaokrąglanie w kierunku $-\infty$	9	[3] Jankowscy J. i M., <i>Przegląd metod i algorytmów numerycznych</i> . cz. 1, WNT, Warszawa 1981.	
1.8.5 Zaokrąglanie w kierunku zera	9	[4] Kielbasiński A., Schwetlick H., <i>Numeryczna algebra liniowa. Wprowadzenie do obliczeń zautomatyzowanych</i> . WNT, Warszawa 1994.	
1.9 Błędy przyporządkowania	10	[5] Kincaid D., Cheney W., <i>Analiza numeryczna</i> . WNT, Warszawa 2006.	
1.10 Operacje	11	[6] Kowalski M., Sikorski C., Stenger F., <i>Selected Topics in Approximation and Computation</i> . Oxford University Press, New York 1995.	
1.11 Wyjątki i ich obsługa	13	[7] Stoer J., Bulirsch R., <i>Wstęp do analizy numerycznej</i> . PWN, Warszawa 1987.	
1.11.1 Dzielenie przez zero	13		
1.11.2 Niepoprawna operacja	13		
1.11.3 Nadmiar	13		
1.11.4 Niedomiar	13		
1.11.5 Niedokładny wynik	13		
1.11.6 Alternatywne sposoby obsługi wyjątków	14		
1.12 Z artykułów pana Kahana	14		
1.12.1 Potrzeba arytmetyki dziesiętnej	14		
2 Uwarunkowanie zadania, numeryczna poprawność i stabilność algorytmu	16		

1 Arytmetyka zmennopozycyjna

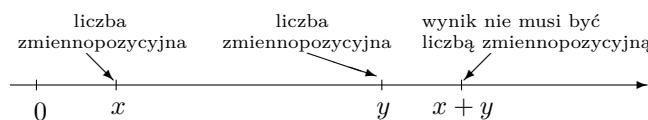
Skończona pamięć komputera sprawia, że jedynie nieliczne spośród liczb rzeczywistych mogą być reprezentowane dokładnie. Sytuacja przedstawia się dużo trudniej niż w przypadku liczb całkowitych, w którym maszyny udostępniają wszystkie wartości z pewnego przedziału. Liczb rzeczywistych jest znacznie więcej — w każdym niepustym, nawet najmniejszym przedziale jest ich nieskończoność wiele. W praktyce komputer operuje jedynie pewnym skończonym podzbiorem liczb wymiernych nazywanym *zborem liczb zmennopozycyjnych*.¹



Skutkiem takiego modelu są błędy reprezentacji danych. Dla większości liczb z pewnego zakresu udało się zagwarantować niewielki błąd względny — typowe jego wielkości to 2^{-24} czy 2^{-53} . Pewne problemy stwarzają otoczenie zera, gdzie błąd względny może być dowolnie duży — mówimy wtedy o zjawisku *niedomiaru*. Również próba reprezentacji liczby o zbyt dużym module prowadzi do kłopotów, zwykle generowany jest sygnał *nadmiaru* i przypisana zostaje specjalna wartość oznaaczająca nieskończoność.



Wykonując działania arytmetyczne, których argumentami są liczby zmennopozycyjne uzyskujemy wyniki rzeczywiste często nie będące liczbami zmennopozycyjnymi. Następuje wówczas zaokrąglenie do liczby zmennopozycyjnej lub, gdy moduł wyniku jest zbyt duży — przypisywany jest symbol $+\infty$ lub $-\infty$.



Nie zawsze wynik działania arytmetycznego jest dobrze określony w zbiorze liczb rzeczywistych. Przykładem może być próba obliczenia pierwiastka kwadratowego z liczby ujemnej, albo podzielenia zera przez zero. Twórcy obowiązującego obecnie standardu arytmetyki zmennopozycyjnej przyjęli zasadę, że każde działanie musi mieć wynik, dlatego wprowadzono specjalną wartość oznaczaną przez NaN , która jest zwieracana w takich wypadkach. Poniżej opiszę obecnie stosowane arytmetyki zmennopozycyjne oraz ich własności.

¹ Używana jest też nazwa *zbior liczb zmienoprzecinkowych*.

1.1 Standardy IEEE 754 i 854

Przez długi czas, bo aż do początku lat osiemdziesiątych ubiegłego wieku, komputery realizowały arytmetyki zmennopozycyjne istotnie różniące się między sobą. Nie zawsze używano systemu dwójkowego.²

Nawet jeśli podstawa liczenia była ta sama, to posługiwano się innymi sposobami kodowania (czyli zapisu liczby jako ciągu bitów). Ten sam program uruchamiany na różnych komputerach mógł dawać odmienne wyniki. Niektóre arytmetyki wymagały „sztuczek”, np. używania wyrażeń typu $(x+x)-x$, by wyniki były sensowne.

W drugiej połowie lat 70-tych zaczęto rozmowy zmierzające do ujednolicenia sposobu przechowywania liczb zmennopozycyjnych. Brali w nich udział początkowo przedstawiciele firm Intel, DEC, Motorola, Zilog, National Semiconductors, IBM. Dużą rolę odegrał William Kahan³ (reprezentował on Intel⁴).

Po 7 latach prac, w 1985 r. ogłoszono standard *IEEE 754-1985*.⁵ Był on dość trudny do realizacji oraz, tak jak wszystkie standardy *IEEE*, „nieobowiązkowy”. Mimo to został od razu (nawet przed rokiem 1984) zaimplementowany przez główne firmy.

W 1987 roku ukazał się standard *IEEE 854-1987*. Dołączono tu arytmetykę dziesiętną, ale zrezygnowano z opisu kodowania. Trzymano się zasady, że każda arytmetyka spełniająca standard poprzedni wypełnia i ten, ale niekoniecznie na odwrót.

Oba standardy tworzone były z myślą nie tylko o producentach sprzętu, ale też twórcach języków programowania i kompilatorów. Jednak w 1997 r. William Kahan pisał o błędym kole, w które „wpadł” standard 754:⁶

- wiele cech standardu nie ma dostatecznego wsparcia ze strony języków programowania i kompilatorów, więc
- cech tych praktycznie nie da się użyć, zatem
- cechy te są niemal nieznane i rzadko zgłasiane jest na nie zapotrzebowanie, co powoduje, że
- cechy te nie mają dostatecznego wsparcia ze strony języków programowania i kompilatorów.

² Obecnie stosowane są na pewnych komputerach też inne systemy liczenia, np. równie wygodny szesnastkowy, czy ósemkowy oraz dziesiętny. Na początku lat sześćdziesiątych ubiegłego wieku na Uniwersytecie Moskiewskim skonstruowano komputer (nazwany Setun), którego najmniejsza jednostka pamięci była trójstanowa („trit” zamiast bitu). Wszystkie obliczenia przeprowadzano w systemie trójkowym. Model wszedł na krótko do produkcji: do 1965 roku wykonano 50 egzemplarzy, po czym zrezygnowano z niego. Badania uniwersyteckie prowadzono dalej do lat siedemdziesiątych. Historia tego komputera jest opisana przez jego współtwórców na stronie www.computer-museum.ru/english/setun.htm.

³ Za swój wkład w tworzenie standardu został w 1989 roku uhonorowany nagrodą Turinga.

⁴ Intel w 1976 r. rozpoczął projekt nowego koprocesora dla mikroprocesorów i8086/8 i i432. Firmie tej zależało na ustanowieniu standardu, zanim prace zostaną ukończone. Pierwszy szkic standardu w dużej mierze korzystał z ustaleń tego projektu. Oczywiście nie zdradzono, w jaki sposób zbudować koprocesor, a jedynie jak wygląda jego arytmetyka i dlaczego (www.cs.berkeley.edu/~wkahan/754story.html).

⁵ IEEE jest skrótem od The Institute of Electrical and Electronics Engineers, USA

⁶ (www.cs.berkeley.edu/~wkahan/754story.html)

W sierpniu 2008 roku, po siedmiu latach pracy, ogłoszono nowy standard: *IEEE 754-2008*. Jest on rozszerzeniem i uścieleniem poprzednich, m.in. włączono do niego arytmetykę dziesiętną wraz z opisem kodowania. We wstępnie pojawia się życzenie, by stał się on podstawą dyskusji między środowiskiem związanym z analizą numeryczną, a projektantami języków programowania. W 2018 roku powinna powstać nowa wersja standardu (każdy standard co 10 lat jest przeglądany pod kątem koniecznych zmian), prawdopodobnie nie będzie się istotnie różniła od poprzedniej.

1.2 Reprezentacja liczb rzeczywistych

Notacja matematyczna dopuszcza wiele różnych sposobów zapisu liczb rzeczywistych, nawet jeśli ograniczymy się do jednego systemu liczenia. Obok ułamków zwykłych ($\frac{3}{4}$) posługujemy się rozwinięciami ($-34.98696\dots$), pozycję przecinka dodatkowo można regulować mnożnikiem będącym całkowitą potęgą podstawy systemu liczenia ($78.765 \cdot 10^{-2}$), niektóre liczby mają swoje przedstawienie symboliczne (e, π), dozwolona jest notacja zawierająca pierwiastkowanie lub inne działania arytmetyczne ($\sqrt{2} + 1$). Taka różnorodność jest przeszkołą w przypadku komputerów — tam konkremtem ciągiowi bitów ma odpowiadać dokładnie jedna liczba i dwa różne ciągi bitów powinny reprezentować dwie różne wartości (drugi warunek nie zawsze jest gwarantowany).

Wśród wielu zapisów jeden stał się podstawą modelu komputerowego. Polega on na przedstawieniu dowolnej liczby rzeczywistej x w postaci:

$$x = \text{znak} \cdot \text{rozwinięcie} \cdot \text{baza}^{\text{wykładnik}},$$

gdzie

1. *znak* może przyjmować wartości $+1$ lub -1 ,
2. *baza* to ustalona liczba naturalna większa od 1 wyznaczająca system liczenia (np. 2 dla dwójkowego, 10 dla dziesiętnego),
3. *rozwinięcie* jest liczbą postaci

$$d_0 d_1 \dots d_k . d_{k+1} d_{k+2} d_{k+3} \dots$$

zapisaną w systemie wyznaczonym przez bazę, przy czym zakładamy, że cyfra d_0 jest różna od zera, o ile $x \neq 0$, dla $x = 0$ przyjmujemy $k = 0$,

4. *wykładnik* jest liczbą całkowitą, również zapisaną w systemie wyznaczonym przez bazę.

Rozwinięcie zawiera wszystkie cyfry znaczące liczby x względem ustalonej bazy, wykładnik zaś reguluje położenie przecinka.

Przykład Następujące liczby są przedstawione w powyższej postaci.

$$\begin{aligned} & +1 \cdot 1.5678231 \cdot 10^{-2}, \\ & -1 \cdot (2.121001)_3 \cdot 3^{(11)_3}, \\ & +1 \cdot 3.333 \dots \cdot 10^1. \end{aligned}$$

Pierwsza i ostatnia liczba zapisana jest w systemie dziesiętnym, środkowa w dwójkowym. Systemy inne niż dziesiętny będziemy odróżniać przez ujęcie rozwinięć w nawiasy zwykłe i użycie indeksu dolnego wskazującego na bazę systemu liczenia. Baza zawsze będzie zapisana w systemie dziesiętnym.

Zauważmy, że zapis ten nie jest jednoznaczny, na przykład

$$\begin{aligned} -1 \cdot 3.21 \cdot 10^0 &= -1 \cdot 321 \cdot 10^{-2}, \\ +1 \cdot 9.999 \dots \cdot 10^{-1} &= +1 \cdot 1 \cdot 10^0, \\ +1 \cdot (1.1)_2 \cdot 2^0 &= +1 \cdot (1.0111 \dots)_2 \cdot 2^0. \end{aligned}$$

Niemniej, przy ustalonej wartości bazy, każdej trójce

$$(\text{znak}; \text{wykładnik}; \text{rozwinięcie})$$

odpowiada dokładnie jedna liczba rzeczywista. Takie uporządkowane trójki będziemy nazywać *reprezentacjami liczb rzeczywistych*. Zauważmy, że każda liczba rzeczywista ma nieskończonie wiele reprezentacji.

1.3 Zbiór wartości zmiennopozycyjnych

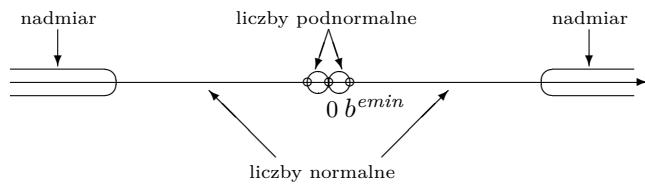
Standard *IEEE 754-2008* definiuje *zbiór wartości zmiennopozycyjnych* posługując się parametrami:

- b — baza, może ona wynosić 2 albo 10,
- p — liczba cyfr użytych do zapisania rozwinięcia,
- $emax$ — maksymalny wykładnik,
- $emin = 1 - emax$ — minimalny wykładnik.

Definicja 1. Elementami zbioru wartości zmiennopozycyjnych są:

1. *liczby zmiennopozycyjne*, czyli liczby postaci
$$(-1)^s \cdot (d_0.d_1d_2 \dots d_{p-1})_b \cdot b^E,$$
gdzie
 - (a) s wynosi 0 lub 1,
 - (b) d_i dla $i = 0, 1, \dots, p-1$ jest dowolna cyfra w systemie liczenia zdefiniowanym przez bazę,
 - (c) $E \in \{emin, emin + 1, \dots, emax\}$,
2. *symbole* $-\infty$ i $+\infty$,
3. *symbole oznaczane przez NaN* (od ang. Not a Number), służące do reprezentacji wyników niepoprawnych działań (np. $\sqrt{-1}$); wyróżnia się dwa rodzaje NaN: takie, które standardowo nie powodują dodatkowych akcji („ciche”, ozn. qNaN) i pozostałe („głośne” — sNaN), wymagające specjalnej obsługi.

Wśród liczb zmiennopozycyjnych wyróżniono *liczby normalne*, czyli te, których moduł jest większy bądź równy b^{emin} i *podnormalne* — niezerowe o module mniejszym niż b^{emin} .



Standard odróżnia $+0$ i -0 , by móc sensownie przeprowadzać działania rozumiane w sensie granic, np.

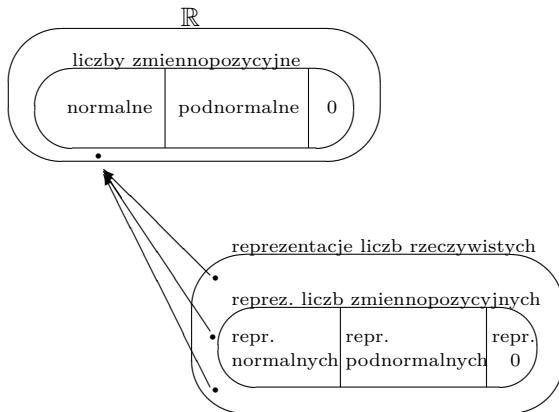
$$\frac{1}{+\infty} = +0, \quad \frac{1}{-0} = -\infty.$$

W większości przypadków „zwykłych” działań liczby $+0$ i -0 są utożsamiane.

Zbiór liczb zmiennopozycyjnych można przedstawić zatem jako sumę trzech rozłącznych zbiorów: liczb normalnych, podnormalnych oraz dwuelementowego o elementach $+0$ i -0 . Liczby zmiennopozycyjne są liczbami rzeczywistymi (dokładniej wymiernymi), można więc je reprezentować przy pomocy uporządkowanych trójkę

(znak; wykładnik; rozwinięcie).

Każda liczba zmiennopozycyjna ma nieskończenie wiele reprezentacji.



Standard wyodrębnia cztery poziomy opisu wartości zmiennopozycyjnych.

Poziom 1. zbiór, który jest aproksymowany, czyli rozszerzony zbiór liczb rzeczywistych $\bar{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$,

Poziom 2. zbiór wartości zmiennopozycyjnych,

Poziom 3. zbiór reprezentacji liczb zmiennopozycyjnych w sumie z $\{+\infty, -\infty, \text{qNaN}, \text{sNaN}\}$,

Poziom 4. zbiór kodów, czyli ciągów bitów reprezentujących wartości zmiennopozycyjne.

1.4 Reprezentacje standardowe

Wśród nieskończoności wielu reprezentacji liczby zmiennopozycyjnej wyróżnię jedną i nazwę ją standardową.

Definicja 2. Reprezentacją standardową liczby zmiennopozycyjnej x należącej do arytmetyki o parametrach b, p, e_{\max} nazywamy reprezentację $((-1)^s; E; (d_0.d_1 \dots d_{p-1})_b)$, która spełnia warunki

- jeśli $x = +0$, to

$$\begin{aligned} s &= 0, \\ E &= e_{\min}, \\ d_0 &= d_1 = \dots = d_{p-1} = 0, \end{aligned}$$

- jeśli $x = -0$, to

$$\begin{aligned} s &= 1, \\ E &= e_{\min}, \\ d_0 &= d_1 = \dots = d_{p-1} = 0, \end{aligned}$$

- jeśli x jest liczbą podnormalną, to

$$\begin{aligned} E &= e_{\min}, \\ d_0 &= 0, \\ \exists i \in \{1, 2, \dots, p-1\} \quad d_i &\neq 0, \end{aligned}$$

- jeśli x jest liczbą normalną, to

$$\begin{aligned} E &\in \{e_{\min}, e_{\min} + 1, \dots, e_{\max}\} \\ d_0 &\neq 0. \end{aligned}$$

Można udowodnić, że każda liczba zmiennopozycyjna ma dokładnie jedną reprezentację standardową.

1.5 Rozmieszczenie liczb zmiennopozycyjnych na osi liczbowej

Arytmetyka zmiennopozycyjna zgodna z opisywanym standardem oferuje skończenie wiele liczb, które są symetrycznie położone względem zera. Narysujmy obecnie ich rozmieszczenie na osi liczbowej. Ze względu na symetrię ograniczymy się jedynie do półosi dodatniej. Ponadto założymy, że $b = 2$. W przypadku $b = 10$ należy postępować analogicznie.

Zajmijmy się najpierw liczbami normalnymi. Rozpatrzmy tylko ich reprezentacje standardowe. Wtedy przecinek w rozwinięciu znajduje się tuż za pierwszą cyfrą znaczącą.

Najmniejszą dodatnią liczbą normalną jest $2^{e_{\min}}$, oznaczmy ją przez MIN. Jej reprezentacją standardową jest

$$(+1; e_{\min}; (1.00 \dots 0)_2).$$

W tabeli przedstawiamy kolejne liczby normalne. Początkowo można tak wybierać reprezentacje, by wykładnik był równy e_{\min} .

wykładnik nik	rozwinięcie $(d_0.d_1 \dots d_{p-1})_2$	liczba
$emin$	1.00...000	$2^{emin}(1.00\dots000)_2 = MIN$
$emin$	1.00...001	$2^{emin}(1.00\dots001)_2 = MIN + h$
$emin$	1.00...010	$2^{emin}(1.00\dots010)_2 = MIN + 2h$
$emin$	1.00...011	$2^{emin}(1.00\dots011)_2 = MIN + 3h$
$emin$	1.00...100	$2^{emin}(1.00\dots100)_2 = MIN + 4h$
\vdots	\vdots	\vdots
$emin$	1.11...110	$2^{emin}(1.11\dots110)_2$
$emin$	1.11...111	$2^{emin}(1.11\dots111)_2$

Liczby te są rozmieszczone równoodlegle z krokiem h .

$$h = 2^{emin}(0.00\dots01)_2 = 2^{emin} \cdot 2^{-(p-1)} = 2^{emin-p+1} = MIN \cdot 2^{-nepozycyjnych}$$

Pozostając przy konwencji dotyczącej położenia przecinka, następujące liczby normalne, muszą mieć większy wykładnik.

wykładnik nik	rozwinięcie $(d_0.d_1 \dots d_{p-1})_2$	liczba
$emin$	0.00...001	$2^{emin}(0.00\dots001)_2 = h$
$emin$	0.00...010	$2^{emin}(0.00\dots010)_2 = 2h$
$emin$	0.00...011	$2^{emin}(0.00\dots011)_2 = 3h$
\vdots	\vdots	\vdots
$emin$	0.11...110	$2^{emin}(0.11\dots110)_2$
$emin$	0.11...111	$2^{emin}(0.11\dots111)_2$

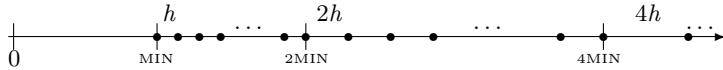
Liczby te są rozmieszczone równoodlegle z krokiem h . Tyle też wynosi odległość między największą z nich a MIN oraz między zerem a najmniejszą z nich. Cała siatka liczb zmien-

wykładnik nik	rozwinięcie $(d_0.d_1 \dots d_{p-1})_2$	liczba
$emin + 1$	1.00...000	$2^{emin+1}(1.00\dots000)_2$
$emin + 1$	1.00...001	$2^{emin+1}(1.00\dots001)_2$
$emin + 1$	1.00...010	$2^{emin+1}(1.00\dots010)_2$
\vdots	\vdots	\vdots
$emin + 1$	1.11...110	$2^{emin+1}(1.11\dots110)_2$
$emin + 1$	1.11...111	$2^{emin+1}(1.11\dots111)_2$

Są one dwa razy większe od swych odpowiedników z pierwszej tabeli, krok też uległ podwojeniu. Zauważmy ponadto, że ostatnia liczba z wykładnikiem $emin$ i pierwsza z $emin + 1$ są odległe o h :

$$\begin{aligned} & 2^{emin+1}(1.00\dots000)_2 - 2^{emin}(1.11\dots111)_2 \\ &= 2^{emin}[(10)_2 - (1.11\dots111)_2] \\ &= 2^{emin}(0.00\dots01)_2 = h. \end{aligned}$$

Podsumowując, w przedziale $[MIN, 2 MIN]$ liczby normalne są rozmieszczone równoodlegle z krokiem h , a w przedziale $[2 MIN, 2^2 MIN]$ z krokiem $2h$. W ogólności przedział $[2^k MIN, 2^{k+1} MIN]$, gdzie $k = 0, 1, \dots, (emax - emin)$, zawiera $2^{p-1} + 1$ liczb normalnych, odległość między każdymi sąsiednimi wynosi $2^k h$. Tworzą one następującą siatkę.



Największą wśród liczb normalnych jest ta, dla której $E = emax$ oraz cyfry d_i są jedynkami. Oznaczmy ją przez MAX i policzmy jej wartość

$$MAX = 2^{emax}(1.11\dots1)_2 = 2^{emax}(2 - 2^{-(p-1)}).$$

Zauważmy, że drugi czynnik należy do przedziału $[1, 2)$, pierwszy szybko rośnie wraz ze wzrostem parametru $emax$.

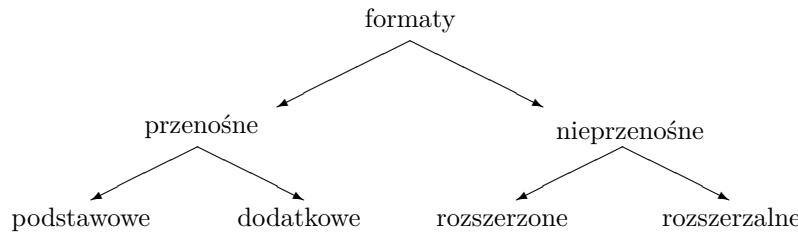
Zajmijmy się obecnie liczbami podnormalnymi dodatnimi. Ponieważ są one mniejsze od 2^{emin} , to można im przypisać reprezentacje postaci

$$(+1; emin; (0.d_1d_2 \dots d_{p-1})_2),$$

co ilustruje poniższa tabela.

1.6 Formaty liczb zmiennopozycyjnych

Standard wprowadza szereg *formatów*, czyli arytmetyk zmiennopozycyjnych różniących się wartościami parametrów. Formaty mogą być implementowane sprzętowo lub programistycznie. Zostały one podzielone na *przenośne* i *nieprzenośne*, w każdej z grup wydzielono dwie podgrupy tak jak na schemacie poniżej.



Formaty *przenośne* mogą służyć do wymiany danych między platformami. Opisano dla nich dokładnie sposób kodowania liczb w postaci ciągu bitów i są one w pełni identyfikowane przez podanie łącznej liczby bitów przeznaczonych na zakodowanie jednej wartości. Wyróżniono tu pięć *podstawowych*: trzy binarne o długościach 32, 64, 128 bitów i dwa dziesiętne 64- i 128-bitowe, oraz *dodatkowe*: 16-bitowy binarny, 32-bitowy dziesiętny oraz k -bitowe binarne i dziesiętne dla $k > 128$ będących wielokrotnością 32. Parametry zawarte są w poniższych tabelach. Standard wymaga, by zaimplementować przynajmniej jeden z formatów podstawowych. Dotyczy to również języków programowania.

Przenośne formaty binarne ($b = 2$)

parametr	binary16 (dodatkowy)	binary32 (podstawowy)	binary64 (podstawowy)	binary128 (podstawowy)	binary{k} (dodatkowy)
p	11	24	53	113	$k - [4 \log_2 k] + 13$
$emax$	15	127	1023	16383	$2^{(k-p-1)} - 1$

Uwaga: $[x]$ oznacza najbliższą do x liczbę całkowitą.

Przenośne formaty dziesiętne ($b = 10$)

parametr	decimal32 (dodatkowy)	decimal64 (podstawowy)	decimal128 (podstawowy)	decimal{k} (dodatkowy)
p	7	16	34	$9k/32 - 2$
$emax$	96	384	6144	$3 \cdot 2^{(k/16+3)}$

Parametry oraz sposób kodowania formatów *nieprzenośnych* mogą zależeć od implementacji. Formaty te podzielono na *rozszerzone* i *rozszerzalne*. Każdy format rozszerzony odpowiada jednemu z podstawowych i musi mieć odpowiednio większe parametry. Standard nie wymaga obowiązkowej implementacji tych standardów, ale zaleca, by języki programowania dostarczały format będący rozszerzeniem największego z dostępnych formatów podstawowych dla każdego zaimplementowanego systemu liczenia. Tabele zawierają wymogi dotyczące parametrów formatów rozszerzonych.

Rozszerzone formaty binarne ($b = 2$)

parametr	extended binary32	extended binary64	extended binary128
$p \geq$	32	64	128
$emax \geq$	1023	16383	65535

Rozszerzone formaty dziesiętne ($b = 10$)

parametr	extended decimal64	extended decimal128
$p \geq$	22	40
$emax \geq$	6144	24576

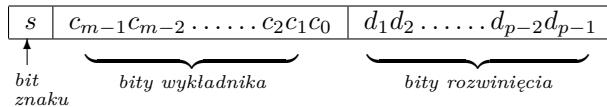
Formaty rozszerzalne, to takie, w których wielkość parametrów p i $emax$ może być kontrolowana przez użytkownika. Znowu nie są one obowiązkowe, ale zalecane.

1.7 Kodowanie binarnych formatów przenośnych

Termin *kodowanie* oznacza tu sposób przedstawienia wartości zmiennopozycyjnej jako ciągu bitów. W tym punkcie zajmujemy się tylko binarnymi formatami przenośnymi. Kodowanie dziesiętne jest trudniejsze, pominiemy je.

1.7.1 Sposób kodowania

Na zapis jednej wartości zmiennopozycyjnej przeznacza się $k = m + p$ bitów. Pierwszy bit jest bitem znaku, w następnych m bitach zapisany jest wykładnik, a ostatnie $p - 1$ bitów przeznaczonych jest na rozwinięcie.



Liczba k , która oznacza liczbę bitów przeznaczonych na kodowanie pojawia się zawsze w nazwie arytmetyki. Parametr p jest ustalony dla każdej arytmetyki. Liczbę m można więc obliczyć następująco

$$m = k - p.$$

Łatwo sprawdzić, że dla wszystkich formatów przenośnych zachodzą równości:

$$emax = 2^{m-1} - 1,$$

$$emin = 1 - emax = 2 - 2^{m-1}.$$

Sposób interpretacji ciągu bitów nie jest określony jedną regułą — wyróżniamy pięć przypadków.

Przypadek 1 — bity wykładnika zawierają same jedynki, a bity rozwinięcia same zera. W zależności od bitu znaku taki ciąg bitów koduje symbol specjalny $+\infty$ lub $-\infty$.

Przypadek 2 — bity wykładnika zawierają same jedynki, a bity rozwinięcia nie zawierają samych zer. Każdy taki ciąg bitów koduje NaN.⁷

Przypadek 3 — bity wykładnika i rozwinięcia zawierają same zera. W zależności od bitu znaku taki ciąg bitów koduje $+0$ lub -0 .

Przypadek 4 — bity wykładnika zawierają same zera, a bity rozwinięcia nie zawierają samych zer. Taki ciąg bitów reprezentuje liczbę podnormalną o wartości

$$(-1)^s \cdot (0.d_1d_2\dots d_{p-1})_2 \cdot 2^{emin}.$$

Przypadek 5 — bity wykładnika nie zawierają ani samych zer, ani samych jedynek. Taki ciąg bitów reprezentuje liczbę normalną o wartości

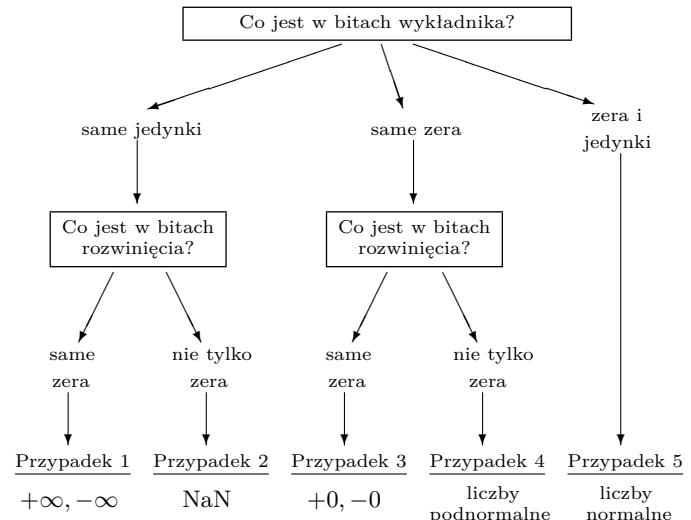
$$(-1)^s \cdot (1.d_1d_2\dots d_{p-1})_2 \cdot 2^E,$$

gdzie

$$E = (c_{m-1}c_{m-2}\dots c_2c_1c_0)_2 - emax.$$

⁷Dokładniej, standard zaleca, by dla $d_1 = 1$ kodowany był qNaN, a dla $d_1 = 0$ sNaN.

Zauważmy, że w przypadku 4 przed przecinkiem stoi zawsze 0, a w przypadku 5 zawsze 1. W ostatnim przypadku sposób obliczenia wykładnika E pozwala na uzyskanie zarówno wartości dodatnich jak i ujemnych przy uniknięciu podwójnej reprezentacji wykładnika równego零.



Przykład Rozpatrzmy arytmetykę binary16. Wtedy

$$\begin{aligned} p &= 11, & emax &= 15, \\ m &= 5, & emin &= -14. \end{aligned}$$

W ciągu bitów

1000 1111 0000 0000

pierwszy: 1 jest bitem znaku, następne pięć: 00011 są bitami wykładnika, a pozostałe dziesięć: 1100000000 to bity rozwinięcia. Bity wykładnika nie zawierają ani samych zer, ani samych jedynek, jest to zatem liczba normalna. Wartość wykładnika wynosi $(00011)_2 - 15 = -12$, a całej liczby

$$(-1)^1 \cdot (1.11)_2 \cdot 2^{-12} = -7 \cdot 2^{-14}.$$

Ciąg bitów

0000 0011 0000 0000

reprezentuje liczbę podnormalną o wartości

$$(-1)^0 \cdot (0.11)_2 \cdot 2^{-14} = 3 \cdot 2^{-16}.$$

Ciągi

1111 1100 0000 0000
0111 1111 0000 0000

mają w bitach wykładnika same jedynki, reprezentują zatem symbole specjalne. Pierwszy z nich to $-\infty$, gdyż bity ułamka są zerami, a znak ujemny. Drugi to NaN, bo wśród bitów ułamka niektóre mają wartość 1.

1.7.2 Dowód poprawności kodowania

Należy obecnie udowodnić, że kodowanie jest poprawne, czyli że każdej wartości zmiennopozycyjnej odpowiada pewien kod i każdemu kodowi wartość zmiennopozycyjna. Zauważmy, że wystarczy rozpatrzyć jedynie liczby zmiennopozycyjne.

Zacznijmy od Przypadku 5. Największą wartość wykładnika E uzyskamy, gdy wszystkie bity c_i zawierają jedynki, z wyjątkiem najmłodszego (w tym przypadku wykładnik z samymi jedynkami nie jest dozwolony)

$$\begin{aligned}(111\ldots10)_2 - emax &= 2^m - 2 - emax \\ &= 2(2^{m-1} - 1) - emax \\ &= 2emax - emax = emax.\end{aligned}$$

Najmniejszą wartość wykładnika E otrzymamy, gdy odpowiadające wykładnikowi bity wypełnione są zerami, oprócz znów najmłodszego

$$(000\ldots01)_2 - emax = 1 - emax = emin.$$

Zatem możliwe do otrzymania są dokładnie te wykładniki E , które należą do zbioru

$$\{emin, emin + 1, \dots, emax - 1, emax\}.$$

Liczby normalne są z definicji nie mniejsze niż 2^{emin} , każda możliwa zatem przedstawić w postaci

$$(-1)^s \cdot (1.d_1d_2\ldots d_{p-1})_2 \cdot 2^E \text{ dla } emin \leq E \leq emax, E \in \mathbb{Z}.$$

Zatem zbiór liczb normalnych jest równy zbiorowi wartości, które można otrzymać w Przypadku 5.

Liczby podnormalne są mniejsze od 2^{emin} , zatem dają się przedstawić w postaci

$$(-1)^s \cdot (0.d_1d_2\ldots d_{p-1})_2 \cdot 2^{emin},$$

występującej w opisie Przypadku 4.

1.8 Reprezentacja liczb rzeczywistych w arytmetyce zmiennopozycyjnej

Arytmetyka zmiennopozycyjna ma za zadanie modelować przestrzeń liczb rzeczywistych. Dysponując siatką opisaną wcześniej możemy sensownie aproksymować liczby jedynie z pewnego przedziału, przy czym różnica między liczbą $x \in \mathbb{R}$ a jej przybliżeniem może rosnąć wraz ze wzrostem $|x|$.

Przyjmijmy, że dane jest $x \in \mathbb{R}$. Przez $rd(x)$ będziemy oznaczać wartość zmiennopozycyjną, która przybliża x . Może to być liczba zmiennopozycyjna, $+\infty$ lub $-\infty$.

Standard definiuje pięć sposobów zaokrąglania liczb rzeczywistych do wartości zmiennopozycyjnych:

1. do najbliższej z wyborem parzystej cyfry,
2. do najbliższej z wyborem większego modułu,
3. w kierunku $+\infty$,

4. w kierunku $-\infty$,

5. w kierunku zera.

Arytmetyki dziesiętne muszą realizować wszystkie pięć sposobów, binarne mogą pominąć zaokrąglanie do najbliższej z wyborem większego modułu. Domyslnym sposobem w przypadku arytmetyk binarnych jest zaokrąglanie do najbliższej z wyborem parzystej cyfry. Ten sam sposób jest zalecany dla arytmetyk dziesiętnych, ale ostateczna decyzja jest pozostawiona twórcom implementacji. Sposoby nie będące domyslnym mają być dostępne dla użytkownika poprzez ustawianie parametru, który jest specjalnie ku temu przeznaczony. Parametr nie musi mieć wartości stałej dla całego programu, ale może być zmieniany dla pewnych jego części.

Sposób zaokrąglania	Arytmetyki binarne	Arytmetyki dziesiętne
do najbliższej z wyborem parzystej cyfry	obowiązkowy domyślny	obowiązkowy zalecany domyślny
do najbliższej z wyborem większego modułu	nieobowiązkowy	obowiązkowy
w kierunku $+\infty$	obowiązkowy	obowiązkowy
w kierunku $-\infty$	obowiązkowy	obowiązkowy
w kierunku zera	obowiązkowy	obowiązkowy

W poniższych punktach omówimy każdy z tych sposobów.

1.8.1 Zaokrąglenie do najbliższej z wyborem parzystej cyfry

Wartość $rd(x)$ wyznaczamy następująco.

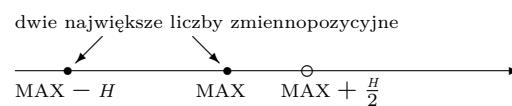
1. Jeśli $|x|$ przekracza zakres arytmetyki, a dokładniej $|x| \geq b^{emax}(b - \frac{1}{2}b^{1-p})$, to

$$rd(x) = \begin{cases} +\infty & \text{gdy } x > 0, \\ -\infty & \text{gdy } x < 0. \end{cases}$$

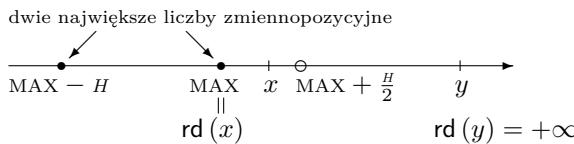
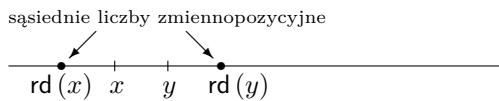
2. Jeśli $|x| < b^{emax}(b - \frac{1}{2}b^{1-p})$ to $rd(x)$ jest równe najbliższej do x liczbie zmiennopozycyjnej.

3. W przypadku, gdy x leży w połowie odległości między dwiema sąsiednimi liczbami zmiennopozycyjnymi, to wybierana jest ta, której cyfra d_{p-1} rozwinięcia w reprezentacji standardowej jest parzysta.

Mogą udowodnić, że liczba $b^{emax}(b - \frac{1}{2}b^{1-p})$ jest równa $\text{MAX} + \frac{H}{2}$, gdzie H jest odległością między dwiema największymi liczbami zmiennopozycyjnymi.



Poniższe rysunki ilustrują różne przypadki położenia liczb $x, y \in \mathbb{R}$ oraz odpowiadające im wartości $\text{rd}(x), \text{rd}(y)$.



1.8.2 Zaokrąglenie do najbliższej z wyborem większego modułu

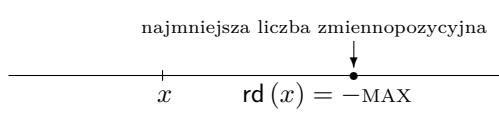
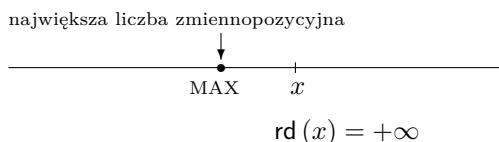
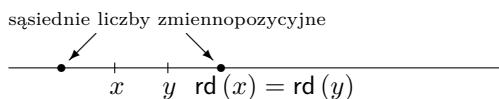
Sposób ten różni się od poprzedniego jedynie w przypadku, gdy x leży w połowie odległości między dwiema sąsiednimi liczbami zmennopozycyjnymi. Tutaj wybierana jest ta, która ma większy moduł.⁸

1.8.3 Zaokrąglanie w kierunku $+\infty$

Wartością $\text{rd}(x)$ jest liczba zmennopozycyjna, która spełnia oba poniższe warunki

1. jest większa lub równa x ,
2. jest najmniejsza ze wszystkich liczb spełniających warunek 1.

Jeśli $x > \text{MAX}$ to nie istnieje żadna liczba zmennopozycyjna spełniająca warunek pierwszy. Wtedy przyjmujemy $\text{rd}(x) = +\infty$.



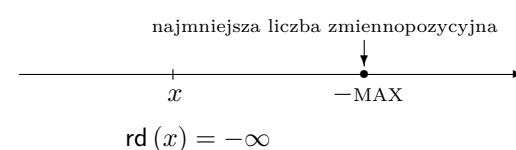
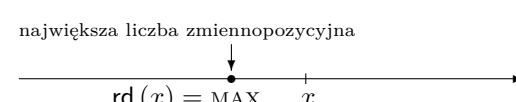
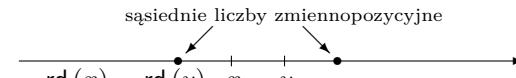
1.8.4 Zaokrąglanie w kierunku $-\infty$

Działa on analogicznie jak poprzednio. Wartością $\text{rd}(x)$ jest liczba zmennopozycyjna, która spełnia oba poniższe warunki

1. jest mniejsza lub równa x ,

2. jest największa ze wszystkich liczb spełniających warunek 1.

Jeśli $x < -\text{MAX}$ to nie istnieje żadna liczba zmennopozycyjna spełniająca warunek pierwszy. Wtedy przyjmujemy $\text{rd}(x) = -\infty$.

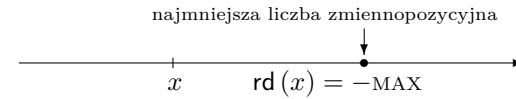
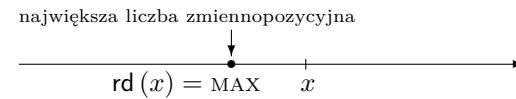
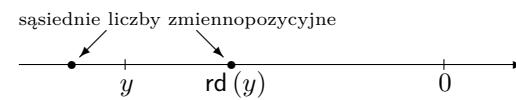
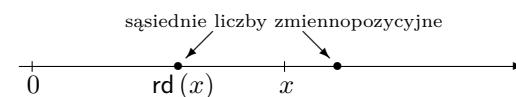


1.8.5 Zaokrąglanie w kierunku zera

Wartością $\text{rd}(x)$ jest liczba zmennopozycyjna, która spełnia oba poniższe warunki

1. moduł tej liczby jest mniejszy lub równy $|x|$,
2. leży najbliżej x ze wszystkich liczb spełniających warunek 1.

Zauważmy, że sposób ten dla $x \geq 0$ działa jak zaokrąglanie w kierunku $-\infty$, a dla $x < 0$ jak zaokrąglanie w kierunku ∞ .



Przykład Rozpatrzmy arytmetykę **binary16** (o parametrach $b = 2, p = 11, emax = 15$ oraz $emin = -14$).

Znajdźmy $\text{rd}(x)$ dla $x = (111.111000011)_2$. W tym celu przekształćmy najpierw liczbę x do postaci takiej, w której rozwińcie ma dokładnie jedną niezerową cyfrę przed przecinkiem:

$$x = +(1.111110000'11)_2 \cdot 2^2.$$

⁸ Ten sposób prowadzi czasem do rezultatów, których intuicyjnie się nie spodziewamy.

Liczba x nie jest liczbą normalną, bo ma za dużo cyfr znaczących. Kreskę ' postawiono po 11 cyfrach znaczących. Zauważmy, że x leży pomiędzy liczbami normalnymi

$$\begin{aligned} l &= (1.1111100000)_2 \cdot 2^2 \\ p &= (1.1111100001)_2 \cdot 2^2, \end{aligned}$$

które są sąsiednimi liczbami zmiennopozycyjnymi.

Jeśli stosujemy zaokrąglenie do najbliższej (jedno z dwóch opisanych), to aby wyznaczyć $\text{rd}(x)$ musimy stwierdzić, czy x jest mniejsze, czy też nie od liczby leżącej dokładnie pośrodku tamtych, czyli

$$s = \frac{1}{2}(l + p) = (1.1111100000'1)_2 \cdot 2^2.$$

Porównując x i s zauważamy, że $x > s$, więc

$$\text{rd}(x) = p = +(1.1111100001)_2 \cdot 2^2.$$

Okazuje się, że obliczanie liczby s nie jest w ogóle potrzebne.

Zauważmy bowiem, że

- liczby z przedziału (s, p) są postaci $(1.1111100000'1\dots)_2 \cdot 2^2$, gdzie wielokropki zastępuje dowolny ciąg cyfr, który nie jest stale równy 0,
- liczby z przedziału $[l, s]$ są zaś postaci $(1.1111100000'0\dots)_2 \cdot 2^2$, tu wielokropki zastępuje dowolny ciąg cyfr.

W przypadku ogólnym wystarczy więc spojrzeć na 12 cyfr znaczącej liczby x :

- jeśli byłaby ona równa 0, to wynikiem przybliżenia byłoby l ,
- jeśli ta cyfra wynosi 1 i wśród dalszych cyfr jest choć jedna jedynka, to wynikiem jest p ,
- jeśli ta cyfra wynosi 1 i wszystkie następne wynoszą zero, to wynik zależy od tego, czy zaokrąglamy z wyborem parzystej cyfry czy z wyborem większego modułu.

Przykład Rozważmy arytmetykę taką, jak w poprzednio. Wyznaczmy $\text{rd}(x)$ dla $x = (1.1111100000'1)_2 \cdot 2^2$ stosując oba rodzaje zaokrąglenia do najbliższej. Tym razem x leży dokładnie pośrodku dwóch sąsiednich liczb zmiennopozycyjnych: $l = (1.1111100000)_2 \cdot 2^2$ i $p = (1.1111100001)_2 \cdot 2^2$.

Jeśli stosujemy zaokrąglenie do najbliższej z wyborem parzystej cyfry, to wynikiem będzie mniejsza z nich, bo jej cyfra d_{10} jest parzysta. W przypadku zaokrąglenia do najbliższej z wyborem większego modułu $\text{rd}(x)$ będzie równe większej, bo ma ona większy moduł.

1.9 Błędy przyporządkowania

Zbadajmy obecnie jak dobrze $\text{rd}(x)$ przybliża x . Będziemy używać dwóch wielkości do pomiaru błędów.

Definicja 3. *Błąd bezwzględny* to wielkość

$$\text{r}_B(x) = x - \text{rd}(x).$$

Błąd względny jest określony dla $x \neq 0$ następująco

$$\text{r}_W(x) = \frac{|\text{rd}(x) - x|}{|x|} = \frac{|\text{r}_B(x)|}{|x|}.$$

Oszacujmy obie wielkości dla różnych x przy założeniu, że posługujemy się zaokrągleniem do najbliższej (obojętnie czy z wyborem parzystej cyfry, czy większego modułu). Inne sposoby zaokrąglania rozpatruje się analogicznie.

Przypadek 1: $x = 0$ Liczba $x = 0$ jest reprezentowana dokładnie, czyli $\text{rd}(0) = 0$ i wtedy $\text{r}_B(0) = 0$, a błąd względny nie jest określony.

Przypadek 2: $0 < |x| < \text{MIN}$ Błąd bezwzględny można oszacować przez

$$|\text{r}_B(x)| \leq \frac{h}{2} = 2^{emin} 2^{-p} = \text{MIN} 2^{-p}.$$

Jest on bardzo mały, przykładowo dla arytmetyki **binary32** wynosi 2^{-150} , a dla **binary64** 2^{-1075} . Natomiast błąd względny może być dowolnie duży. Dla liczb bliskich 0, a dokładniej takich $x \neq 0$, że $\text{rd}(x) = 0$ (czyli spełniających nierówność $|x| < \frac{h}{2}$) wynosi 1 — cała informacja o liczbie x ulega zniszczeniu.

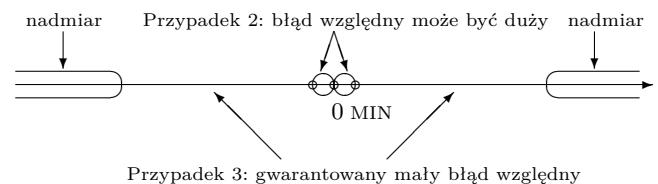
Przypadek 3: $\text{MIN} \leq |x| < \text{MAX} + \frac{H}{2}$ Oszacujmy maksymalny błąd bezwzględny, jaki może się pojawić. Jeśli $|x| \in [2^k \text{MIN}, 2^{k+1} \text{MIN})$ dla pewnego $k = 0, 1, \dots, emax - emin$, to

$$\begin{aligned} |\text{r}_B(x)| &\leq 2^k \frac{h}{2} \\ &= 2^k 2^{emin} 2^{-p} \\ &= 2^k \text{MIN} 2^{-p} \\ &\leq |x| 2^{-p}. \end{aligned}$$

Błąd względny można oszacować następująco

$$\text{r}_W(x) = \frac{|\text{r}_B|}{|x|} \leq 2^{-p}.$$

Jego maksymalna wielkość zależy od parametru p — im większe p tym mniejszy „najgorszy” błąd względny, jaki może się pojawić. Ponieważ p jest dość dużą liczbą to błąd względny jest zawsze mały dla $\text{MIN} \leq |x| < \text{MAX} + \frac{H}{2}$. W sensie względnym utrata informacji o liczbie x jest niewielka.



Definicja 4. Liczbę 2^{-p} nazywamy *precyzją arytmetyki*. Będziemy ją oznaczać przez ν .

Precyzję arytmetyki można też zdefiniować w inny, równoważny sposób, co jest przedmiotem jednego z zadań.

Przykład Weźmy liczbę $x = \text{MIN} + \frac{h}{2}$. Wtedy błąd bezwzględny wynosi

$$|r_B(x)| = \frac{h}{2}.$$

Ponieważ $\frac{h}{2} = \text{MIN} \nu$, to $|x| = \text{MIN}(1 + \nu)$ i

$$\begin{aligned} |r_B(x)| &= \text{MIN} \nu \\ &= |x| \frac{\text{MIN}}{|x|} \nu \\ &= |x| \frac{\text{MIN}}{\text{MIN}(1 + \nu)} \nu \\ &= |x| \frac{1}{1 + \nu} \nu. \end{aligned}$$

Błąd względny ma zatem wartość

$$r_W(x) = \frac{1}{1 + \nu} \nu,$$

która tylko nieznacznie różni się od ν . Przykład ten wykazuje, że oszacowania błędu względnego są bardzo dokładne.

1.10 Operacje

Duża część standardu poświęcona jest opisowi operacji, które powinny być udostępnione. Wprowadzony jest ich podział ze względu na pewne cechy, co zostanie tu pominięte. Każda operacja jest dokładnie zdefiniowana dla wszystkich wartości argumentów, w szczególności jeśli argument jest zmiennopozycyjny, to wynik operacji musi być określony zarówno wtedy, gdy jest on liczbą zmiennopozycyjną, jak i wtedy, gdy jest jednym z symboli $+\infty, -\infty, \text{qNaN}, \text{sNaN}$.

Podstawową zasadę dotyczącą wykonywania operacji można sformułować jak następuje.

Jeżeli wynikiem operacji jest liczba, to powinna być ona wyznaczona w taki sposób, jakby najpierw była obliczona dokładnie (nieskończony zakres i nieskończenie wiele cyfr rozwinięcia), a następnie zaokrąglona zgodnie z obowiązującym sposobem do liczby zmiennopozycyjnej (ewentualnie do $+\infty$ lub $-\infty$).

W praktyce wystarczy najpierw obliczyć wynik z dostatecznie dużą dokładnością, a nie nieskończoną. Sprzętowo wykorzystuje się do tego odpowiednio „większej” arytmetyki. Następnie konwertuje się go do arytmetyki docelowej.

Przez $\text{fl}(\text{wyrażenie})$ będę oznaczać wartość *wyrażenia* obliczonego w konkretnej arytmetyce zmiennopozycyjnej, przy czym jeśli *wyrażenie* zawiera wiele operacji, to są one wykonywane po kolejno zgodnie z zapisem i wynik każdej z nich jest zaokrąglany do wartości zmiennopozycyjnej. *Wyrażenie* może być zapisane zgodnie z konwencją matematyczną albo regułami stosowanymi w ustalonym języku programowania.

Przykład Napis $\text{fl}(2/3 + \sqrt{2})$ oznacza liczbę zmiennopozycyjną c , którą otrzymamy gdybyśmy

1. obliczyli dokładnie $2/3$ i wynik zaokrąglili do liczby zmiennopozycyjnej a ,
2. obliczyli dokładnie $\sqrt{2}$ i wynik zaokrąglili do liczby zmiennopozycyjnej b ,
3. obliczyli dokładnie $a + b$ i wynik zaokrąglili do liczby zmiennopozycyjnej c .

Ten sam rezultat otrzymamy pisząc $\text{fl}(2/3 + \sqrt{2})$ zgodnie z konwencją w C.

Standard wymaga, by realizowane były operacje arytmetyczne dodawania, odejmowania, mnożenia, dzielenia, wyciągania pierwiastka kwadratowego, połączonego mnożenia z dodawaniem. Ostatnia operacja jest trójargumentowa, będącą oznaczaną przez FMA od ang. FusedMultiplyAdd. Z definicji $\text{FMA}(x, y, z) = x * y + z$. Operacja ta nie jest równoważna następującym po sobie operacjom mnożenia i dodawania, bo FMA generuje tylko jeden błąd zaokrągleń.

Poprawna arytmetyka zmiennopozycyjna w myśl podstawowej zasady spełnia dla x, y, z będących liczbami zmiennopozycyjnymi równości:

$$\begin{aligned} \text{fl}(x + y) &= \text{rd}(x + y), \\ \text{fl}(x - y) &= \text{rd}(x - y), \\ \text{fl}(x * y) &= \text{rd}(x * y), \\ \text{fl}(x/y) &= \text{rd}(x/y), \quad \text{dla } x \neq 0, \\ \text{fl}(\sqrt{x}) &= \text{rd}(\sqrt{x}), \quad \text{dla } x \geq 0, \\ \text{fl}(\text{FMA}(x, y, z)) &= \text{rd}(x * y + z). \end{aligned}$$

Przykład Rozpatrzmy następujący program w języku C.

```
#include<stdio.h>
#include<math.h>

main(){
    float x=pow(2,-148);
    x=x/8;
    if (x==0.0)
        printf("ZERO");
    else
        printf("NIE_ZERO");
}
```

Załóżmy, że zastosowano zaokrąglanie do najbliższej oraz, że typ **float** jest zgodny z formatem podstawowym **binary32**. Wtedy liczby

$$8 = (1.00000000000000000000000000)_2 \cdot 2^3$$

oraz

$$2^{-148} = (0.000000000000000000000000010)_2 \cdot 2^{-126}$$

są niezerowymi liczbami zmiennopozycyjnymi. Dokładny wynik dzielenia $2^{-148}/2^3$ wynosi 2^{-151} , ale w tej arytmetyce będzie reprezentowany przez 0 i na ekranie pojawi się napis ZERO.

Nie przedstawię tutaj wszystkich definicji operacji i reguł nimi rządzących. Są one zawarte w standardzie. Dla przykładu uczynię to w przypadku dodawania dla formatów binarnych, aby ukazać jak szczegółowy jest standard.

Przykład Operator dodawania jest dwuargumentowy. Argumenty i wynik nie muszą być tego samego formatu, co więcej dodawanie ma być zaimplementowane dla wszystkich możliwych kombinacji formatów o tej samej podstawie liczenia. Jeśli argumenty są liczbami zmiennopozycyjnymi, to wynik jest definiowany przez podstawową regułę (wiersz pierwszy w tabeli). Wszystkie inne przypadki są zawarte w pozostałych wierszach tabeli. Symbol z oznacza, że argument lub wynik jest liczbą zmiennopozycyjną. Wystąpienie ∞ (bez znaku) oznacza $+\infty$ lub $-\infty$, podobnie NaN zastępuje qNaN lub sNaN . Przecinek zastępuje spójnik „lub”. Dolna część tabeli zawiera przypadki symetryczne do tych, które umieszczone w środkowej części.

arg1	arg2	wynik arg1+arg2
z	z	$z, +\infty, -\infty$
$z, +\infty$	$+\infty$	$+\infty$
$z, -\infty$	$-\infty$	$-\infty$
$+\infty$	$-\infty$	qNaN
z, ∞, NaN	NaN	qNaN
$+\infty$	$z, +\infty$	$+\infty$
$-\infty$	$z, -\infty$	$-\infty$
$-\infty$	$+\infty$	qNaN
NaN	z, ∞, NaN	qNaN

Dodatkowo sformułowane są zasady dotyczące znaku zera, gdy oba argumenty i/lub wynik są zerami.

arg1	arg2	wynik arg1+arg2
$+0$	$+0$	$+0$
-0	-0	-0
$+0$	-0	$\begin{cases} -0 & \text{przy zaokr. w kier. } -\infty, \\ +0 & \text{w p.p.} \end{cases}$
-0	$+0$	$\begin{cases} -0 & \text{przy zaokr. w kier. } -\infty, \\ +0 & \text{w p.p.} \end{cases}$

Podobne reguły dotyczą sytuacji, gdy przynajmniej jeden z argumentów jest niezerowy, a wynikiem jest zero. Jeśli argumenty i wynik są różnych formatów, to niekoniecznie argumenty muszą mieć przeciwnie znaki. Wtedy wynikiem jest zero z takim samym znakiem, co argumenty. Gdy argumenty są przeciwnych znaków, to wynikiem jest $+0$, z wyjątkiem sytuacji, gdy stosowane jest zaokrąglanie w kierunku $-\infty$. Podsumowuje to poniższa tabela, zauważmy, że obejmuje ona również przypadek obu argumentów zerowych. Z zakładamy jedynie, że argumenty są takie, by wynikiem dodawania było zero.

znak arg1	znak arg2	wynik arg1+arg2 (będący zerem)
$+$	$+$	$+0$
$-$	$-$	-0
$+$	$-$	$\begin{cases} -0 & \text{przy zaokr. w kier. } -\infty, \\ +0 & \text{w p.p.} \end{cases}$
$-$	$+$	$\begin{cases} -0 & \text{przy zaokr. w kier. } -\infty, \\ +0 & \text{w p.p.} \end{cases}$

Na koniec wymienię, jakiego typu operacje są zdefiniowane w standardzie.

- operacje arytmetyczne: dodawanie, odejmowanie, mnożenie, dzielenie, wyciąganie pierwiastka kwadratowego, FMA, wyznaczanie reszty z dzielenia,
 - operacje konwersji z i do różnych typów całkowitych, formatów zmiennopozycyjnych, ciągów znaków; możliwy jest wybór rodzaju zaokrągleń,
 - operatory kopowania wartości z zachowaniem znaku, z zamianą znaku na przeciwny, z opuszczeniem znaku, z zamianą znaku ta taki sam, jaki ma drugi argument,
 - operatory porównania (jest ich 22), różnią się między innymi tym, jak traktowane są symbole specjalne,
 - operatory sprawdzające, który standard spełnia arytmetyka (z 1985 r. czy z 2008 r.),
 - operatory pozwalające sprawdzić z jaką wartością mamy do czynienia (liczbą normalną, podnormalną, zerem, symbolem specjalnym), jaka jest podstawa liczenia, jaki jest znak wartości,
 - operatory sprawdzające, zapamiętujące i zmieniające stan flag, które informują o zajściu sytuacji wyjątkowej.
- Oprócz tych standard zaleca dodatkowe operacje, nie są one obowiązkowe. Mają one realizować:
- funkcje wykładnicze,
 - funkcje logarytmiczne,
 - funkcje potęgowe,
 - funkcje trygonometryczne i odwrotne do trygonometrycznych,
 - funkcje hiperboliczne,
 - funkcję obliczającą $1/\sqrt{x}$,
 - funkcję obliczającą $\sqrt{x^2 + y^2}$,
 - funkcję obliczającą $(1+x)^n$,
 - funkcje obliczające sumę, sumę modułów, sumę kwadratów, iloczyn elementów wektora,
 - funkcje obliczające różne kombinacje sum i iloczynów elementów dwóch wektorów,
 - funkcje pozwalające w trakcie działania programu zmieniać sposoby zaokrągleń.

1.11 Wyjątki i ich obsługa

Wyjątkiem nazywamy nietypowe zdarzenie, które może być związane z pewnym błędem lub nie, jest wykrywalne przez sprzęt albo oprogramowanie i wymaga często specjalnej obsługi. Przykładem takiego zdarzenia jest dzielenie przez zero lub wyciąganie pierwiastka kwadratowego z liczby ujemnej, albo napotkanie końca pliku podczas odczytu. W momencie rozpoznania podejmowane są specjalne kroki. Może to być zakończenie działania bieżącego programu i wygenerowanie odpowiedniego sygnału przerwania wewnętrznego, co spowoduje obsługę przez program z systemu operacyjnego. To rozwiązanie jest typowe dla starszych języków programowania. Innym sposobem jest dostarczenie specjalnego podprogramu (lub podprogramów do wyboru) przez twórcę kompilatora, albo twórcę programu, o ile język jest wyposażony w odpowiednie narzędzia. Kolejną możliwość polega na zignorowaniu wyjątku.

Standard definiuje pięć rodzajów wyjątków. Mogą być one obsługiwane w sposób domyślny lub inny. Z każdym z nich związana jest *flaga*, czyli zmienna, która przyjmuje dwa stany, umownie nazywane *podniesieniem* i *opuszczeniem* flagi. Domyślnie, zgłoszenie wyjątku powoduje podniesienie flagi. Opuszczenie flagi jest możliwe tylko na życzenie użytkownika.

Program rozpoczyna działanie z opuszczonymi flagami chyba, że „*odziedziczył*” flagi po innym programie. Standard wymaga, by użytkownik miał możliwość sprawdzania i zapamiętywania stanu flag oraz opuszczania i podnoszenia ich. Ponadto reguluje wiele innych szczegółów związanych z flagami.

1.11.1 Dzielenie przez zero

Wyjątek jest zgłaszany w przypadku operacji o argumentach będących liczbami zmiennopozycyjnymi i dokładnym wyniku równym $+\infty$ lub $-\infty$. Przykładem jest działanie $1 / +0$. Domyślna obsługa polega na podniesieniu odpowiedniej flagi.

1.11.2 Niepoprawna operacja

Zgłoszenie następuje wtedy i tylko wtedy, gdy nie można doстarczyć żadnego użytecznego wyniku. Dzieje się tak, gdy argumenty nie są poprawne dla danej operacji. Wynikiem (jeśli jest zmiennopozycyjny) powinien być **qNaN**. Oto przykłady (jeśli nie jest zaznaczony znak przy nieskończoności lub zerze, to można wziąć dowolny):

1. działania arytmetyczne, w których jednym z argumentów jest **sNaN**,
2. $0 * \infty, \infty * 0,$
3. $\text{FMA}(0, \infty, z), \text{FMA}(\infty, 0, z)$, chyba, że $z = \text{qNaN}$, w ostatnim przypadku, to czy jest to niedozwolona operacja zależy od implementacji,
4. $+\infty + (-\infty), +\infty - (+\infty), \text{FMA}(+\infty, 1, -\infty),$
5. $0/0, \infty/\infty,$
6. \sqrt{x} dla $x < 0$.

Domyślna obsługa polega na podniesieniu odpowiedniej flagi.

1.11.3 Nadmiar

Podczas zaokrąglania może być zgłoszony wyjątek nadmiaru. Występuje on w następujących sytuacjach. Wyobraźmy sobie, że siatka liczb zmiennopozycyjnych została rozszerzona do nieskończenie wielu wartości poprzez rezygnację z ograniczenia wartości wykładnika E . Dla ustalonej liczby x obliczamy reprezentację x dysponując taką siatką (postępujemy zgodnie z obowiązującym sposobem zaokrąglania). Nie pojawią się wtedy wartości $+\infty$ lub $-\infty$, lecz zawsze będzie to pewna liczba. Jeśli tak obliczona reprezentacja przekroczy co do modułu MAX to dochodzi do nadmiaru.

Domyślnie program obsługuje podnosi odpowiednią flagę i dodatkowo zgłasza wyjątek „*niedokładny wynik*”.

1.11.4 Niedomiar

Do niedomiaru dochodzi wówczas, gdy otrzymano bardzo mały niezerowy wynik. W formatach binarnych stwierdzone to może być na jeden z poniższych sposobów:

1. *po zaokrągleniu* — rozszerzamy siatkę liczb zmiennopozycyjnych do nieskończenie wielu wartości poprzez rezygnację z ograniczenia wartości wykładnika E ; dla ustalonej liczby x obliczamy reprezentację x dysponując taką siatką (postępujemy zgodnie z obowiązującym sposobem zaokrąglania); jeśli tak uzyskany wynik jest niezerowy i mniejszy co do modułu od b^{emin} , to wykrywamy niedomiar,
2. *przed zaokrągleniem* — niedomiar jest wykrywany, jeśli dokładny wynik jest niezerowy i mniejszy co do modułu od b^{emin} .

Wymaga się, by wybrany sposób był konsekwentnie używany w przypadku wszystkich formatów binarnych i wszystkich rodzajów zaokrągleń.

Jeśli format jest dziesiętny, to niedomiar jest stwierdzany przed zaokrągleniem.

Domyślna obsługa tego zdarzenia jest następująca. Jeśli po stwierdzeniu niedomiaru dokładny wynik nie jest liczbą zmiennopozycyjną (czyli musi być zaokrąglony), to podniesiona jest odpowiednia flaga i dodatkowo zgłaszany wyjątek niedokładny wynik. Jeśli jest liczbą zmiennopozycyjną, to nic nie jest robione (flaga pozostaje opuszczona). To jedyny przypadek, kiedy rozpoznanie wyjątku nie powoduje podniesienia flagi.

1.11.5 Niedokładny wynik

Zgłaszanego jest wtedy, gdy uzyskany po zaokrągleniu wynik jest różny od wyniku dokładnego. Domyślna obsługa polega na podniesieniu odpowiedniej flagi.

1.11.6 Alternatywne sposoby obsługi wyjątków

Twórcy standardu sugerują, by języki programowania dostarczały oprócz domyślnego inne sposoby obsługi wyjątków. Wymienione takie, które

1. działają tak jak domyślne, ale bez podnoszenia żadnych flag,
2. działają tak jak domyślne, ale twórcy języka decydują, kiedy flagi są podnoszone,
3. działają tak jak domyślne i dodatkowo zapamiętują informację o wyjątku,
4. zastępują wynik operacji wyrażeniem podanym przez użytkownika,
5. w przypadku niedomiaru zastępują wynik zerem lub b^{emin} lub $-b^{emin}$ według pewnych reguł, równocześnie podnosząc flagę i zgłaszając wyjątek niedokładny wynik.

1.12 Z artykułów pana Kahana

Na stronie www.cs.berkeley.edu/~wkahan znajduje się szereg artykułów autorstwa W. Kahana, które poruszają tematy związane z arytmetyką zmiennopozycyjną. Artykuł www.cs.berkeley.edu/~wkahan/Mindless.pdf rozpoczyna się poniższym stwierdzeniem.

„Dane zmiennopozycyjne pojawiają się w niezmiernych ilościach, a programy numeryczne jak nigdy dotąd rosną, stają się ambitne i skomplikowane, podczas gdy ich użytkownicy (uśredniając) coraz mniej wiedzą o numerycznej analizie błędów, chociaż ich wiedza nie jest mniejsza niż ich poprzedników w dziedzinach, o które dbają, by się ich nauczyć. Konsekwencją jest to, że anomalie numeryczne zwykle pozostają niezauważone, a jeśli już, to na ogół błędnie zdiagnozowane. Na szczęście większość z nich nie ma znaczenia. W ogóle większość obliczeń nie ma znaczenia.

Obliczenia zmiennopozycyjne stały się tak tanie, że często nie są wiele warte. Powiększający się ogrom, głównie przypadkowych użytkowników, wykorzystuje je zwykle w rozrywce i grach. Anomalie powodowane błędami zaokrągleń zamigoczą przed oczami zbyt szybko, by zostały zauważone, a jeśli zostaną, to jedynie awansują do „cech” opisanych być może na jakimś forum tak

«Nie trzeba szukać i składać w ofierze żadnej dziewczyny wiedźmie strzegącej bramy do poziomu siedemnastego: wiedźmę dotknij katatonii, gdy zaoferujemy jej dokładnie 13.875 dolarów.»

Powiększa się jednak liczba obliczeń, które są ważne i to bardzo, choć nie przybywa ich tak szybko jak gier. Ale coraz mniejszej liczbie programistów i użytkowników wykształcenie i doświadczenie umożliwia znalezienie i usunięcie numerycznych anomalii. Są one niewidoczne w tekście programu, gdyby tak nie było, ich nazwy zalałyby wszystko inne. Widzimy je umysłem i w modelu obliczeń zbudowanym dla celów analizy błędów zaokrągleń.

Analiza błędów przyciąga coraz mniej studentów i rzadziej staje się ścieżką kariery naukowej. Dlatego prawie wszyscy użytkownicy i programiści obliczeń zmiennopozycyjnych wymagają pomocy nie tyle w przeprowadzaniu analizy błędów (nie będą tego robić), jak w ustaleniu, który błąd zaokrągleń jest przyczyną ich zgryzoty i gdzie. Potem nastąpi, o ile to możliwe, przypisanie winy i pozbicie się problemu.”

1.12.1 Potrzeba arytmetyki dziesiętnej

Arytmetyka dziesiętna zmiennopozycyjna zgodna ze standardem *IEEE 754-2008* jest sporo trudniejsza do realizacji niż binarna. Działa też nieco wolniej. Niektórzy zadają pytanie, czy i kiedy jest ona użyteczna. Poniżej zamieszczam przykłady ilustrujące jej przydatność. Wskazują one pewne anomalie, których by nie było, gdyby zastosowano arytmetykę dziesiętną. Przykłady dotyczą arkusza kalkulacyjnego EXCEL 2000 i są wzięte ze strony www.cs.berkeley.edu/~wkahan/Mindless.pdf, paragraf „Błędy zaprojektowane tak, by ich nie znaleźć”.

EXCEL 2000 posługuje się arytmetyką binarną, dla której $p = 53$ (odpowiada ona **binary64**). Wyniki prezentowane są w systemie dziesiętnym — dokonywane są odpowiednie konwersje. Twórcy arkusza chcą, by użytkownik miał wrażenie, że EXCEL posługuje się arytmetyką dziesiętną.

Przykład Obliczamy wyrażenia opisane w drugiej kolumnie, przy czym x , y , z , w zastępujemy odwołaniami do wcześniej obliczonych komórek. Format komórek z obliczonymi wynikami jest ustawiony na **Naukowy** z 16 cyframi znaczącymi.

Nr	Wyrażenie	Obliczony wynik
1	1.2345123451234512345	1,234512345123450E+00
2	$x=4/3$	1,333333333333330E+00
3	$y=x-1$	3,333333333333330E-01
4	$z=y*3$	1,000000000000000E+00
5	$w=z-1$	0,000000000000000E+00
6	$w*2^{52}$	0,000000000000000E+00
7	$(4/3-1)*3-1$	0,000000000000000E+00
8	$((4/3-1)*3-1)$	-2,220446049250310E-16
9	$((4/3-1)*3-1)*2^{52}$	-1,000000000000000E+00

W linii pierwszej sprawdzamy, ile faktycznie wyświetlanych jest znaczących cyfr dziesiętnych. Okazuje się, że maksymalnie 15, jeśli zażyczymy więcej, to zostaną one zastąpione zerami.

Wynikiem dzielenia $4/3$ jest $1,33333333333330E+00$ (po przecinku jest 14 trójkę). Odjęcie 1 powoduje pojawić się piętnastej trójki!

W linii czwartej spodziewamy się raczej wyniku $9,9999999999990E+00$ niż $1,000000000000000E+00$. W kolejnych dwóch liniach sprawdzamy, czy jest to „prawdziwa” jedynka czyli, czy na dalekich pozycjach po przecinku (w systemie binarnym) nie znajdują się niezerowe cyfry.

W liniach siódmej i ósmej wykonujemy te same działania co powyżej, ale „zbiorczo”, tzn. bez odwoływanego się do poprzednich komórek. Dodanie nawiasów zewnętrznych zmieniło wynik! Linia 9 ukazuje, że zmiana dotyczy ostatniej cyfry

przechowywanej w arytmetyce binarnej, którą posługuje się EXCEL.

Warto porównać jak zachowują się w tych sytuacjach inne arkusze. Przykładowo OpenOffice.org 2.0.4 zachowuje się podobnie, ale nie zmienia wyniku po dodaniu zewnętrznych nawiasów.

Przykład Liczby zmiennopozycyjne, którymi faktycznie posługuje się EXCEL są postaci

$$(-1)^s 2^E (d_0.d_1 d_2 \dots d_{52})_2,$$

gdzie $s, d_i \in \{0, 1\}$, a wykładnik E jest liczbą całkowitą z przedziału $[-1022, 1023]$. Zauważmy, że liczby $\frac{1}{2}, 1$ i 2 są reprezentowane dokładnie, bo

$$\frac{1}{2} = 2^{-1} (1.\underbrace{00\dots 0}_{52})_2,$$

$$1 = 2^0 (1.\underbrace{00\dots 0}_{52})_2,$$

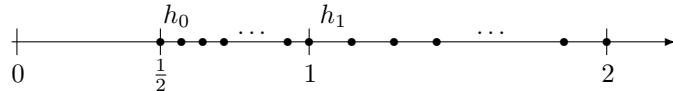
$$2 = 2^1 (1.\underbrace{00\dots 0}_{52})_2.$$

Co więcej, w przedziale $[\frac{1}{2}, 1]$ liczby zmiennopozycyjne są z rozmieszczone równoodlegle z krokiem

$$h_0 = 2^{-1} 2^{-52} = 2^{-53},$$

a w $[1, 2]$ z krokiem

$$h_1 = 2^0 2^{-52} = 2^{-52}.$$



Rozważmy sąsiednie liczby zmiennopozycyjne postaci

$$1 - ah_0 = 1 - a/2^{53}$$

dla $a = 13, 12, \dots, 5$. Okazuje się, że są one wszystkie wypisywane przez EXCEL w ten sam sposób, jako $9,99999999999990E-01$ mimo, że są odróżnialne w wewnętrznej reprezentacji binarnej. Podobnie liczby zmiennopozycyjne postaci

$$1 - bh_0 = 1 - b/2^{53}$$

dla $b = 4, 3, 2, 1, 0$ oraz liczby

$$1 + bh_1 = 1 + b/2^{52}$$

dla $b = 1, 2, \dots, 22$ są „utożsamione” i wypisane jako $1,00000000000000E+00$.

Nie jest to zjawisko dziwne, bo aby zagwarantować różne reprezentacje dziesiętne dla różnych liczb zmiennopozycyjnych opisanego wyżej formatu, trzeba użyć przynajmniej 17 dziesiętnych cyfr znaczących. Dlaczego EXCEL nie pozwala więc na wypisanie 17 cyfr? Prawdopodobnie projektant chciał zagwarantować, że liczba wpisana przez użytkownika po wypisaniu będzie wyglądać dokładnie tak samo. W tej arytmetyce

jest to zachowane tylko wtedy, gdy dopuścimy co najwyżej 15 cyfr wpisywanych. Przy wpisywanych 17 cyfrach i tylu wypisywanych mogą pojawić się różnice.

Czy liczby tak samo wypisywane przez EXCEL są utożsamione? Okazuje się, że nie.

liczba x	$1 - 8/2^{53}$	$1 - 7/2^{53}$
wypisane x	$9,99999999999990E-01$	$9,99999999999990E-01$
x-1	$-8,881784197001250E-16$	$0,000000000000000E+00$
sign(x-1)	$-1,000000000000000E+00$	$-1,000000000000000E+00$
x<1	PRAWDA	PRAWDA
x=1	FAŁSZ	FAŁSZ

Obliczenie $x-1$ daje w wynik ujemny dla pierwszej liczby, a zero dla drugiej. Jednak w obu przypadkach stwierdzono, że liczby są mniejsze od 1.

Oto kolejne przykłady. Cztery rozważane wartości są wypisywane jako $1,000000000000000E+00$.

liczba x	$1 - 1/2^{53}$	$1 - 0/2^{52}$
wypisane x	$1,000000000000000E+00$	$1,000000000000000E+00$
x=1	PRAWDA	PRAWDA
floor(x)	$1,000000000000000E+00$	$1,000000000000000E+00$
x-1	$0,000000000000000E+00$	$0,000000000000000E+00$
sign(x-1)	$-1,000000000000000E+00$	$0,000000000000000E+00$
acos(x)	$1,490116119384770E-08$	$0,000000000000000E+00$

liczba x	$1 + 1/2^{52}$	$1 + 8/2^{52}$
wypisane x	$1,000000000000000E+00$	$1,000000000000000E+00$
x=1	PRAWDA	PRAWDA
floor(x)	$1,000000000000000E+00$	$1,000000000000000E+00$
x-1	$0,000000000000000E+00$	$1,776356839400250E-15$
sign(x-1)	$1,000000000000000E+00$	$1,000000000000000E+00$
acos(x)	#LICZBA!	#LICZBA!

We wszystkich czterech przypadkach sprawdzenie warunku, czy liczby są równe 1 daje odpowiedź pozytywną oraz zaokrągleniem w dół tych liczb jest 1. Dzieje się tak mimo, że w wewnętrznej reprezentacji binarnej dwie są większe, a jedna mniejsza od 1.

Przy odejmowaniu jedynki jeden z wyników jest różny od zera. Badanie znaku różnicy liczby i jedynki daje trzy możliwe odpowiedzi, zgodne z reprezentacją binarną. Próba obliczenia wartości funkcji acos dwukrotnie się nie powodzi — zgłoszony jest błąd wykroczenia poza dziedzinę (tak zresztą jest). Dlaczego jednak część funkcji zdaje się korzystać z wartości reprezentowanych w pamięci w systemie dwójkowym (np. acos), a inne z wartości takich, jakie pojawiają się na ekranie (np. floor)? Jak wykonywane jest odejmowanie?

Podobnie, choć nieidentycznie, zachowuje się arkusz OpenOffice.org 2.0.4. Oprócz powyższych zjawisk, nie radzi sobie z wypisaniem liczby $1 - 6/2^{53}$, na ekranie pojawia się $10,00000000000000E+00$.

Przykład Aby lepiej zilustrować niespójność oblicznia funkcji sign oraz odejmowania rozważmy liczby zmiennopozycyjne, które są wypisywane jako $1,00000000000010E+00$ (są one postaci $1 + b/2^{52}$ dla $b = 23, 24, \dots, 67$).

liczba x	$1 + 37/2^{52}$
wypisane x	1,000000000000010E+00
$x=1,00000000000001$	PRAWDA
$x-1,00000000000001$	-1,776356839400250E-15
$(x-1,00000000000001)$	-1,776356839400250E-15
$\text{sign}(x-1,00000000000001)$	-1,000000000000000E+00

liczba x	$1 + 38/2^{52}$
wypisane x	1,00000000000000010E+00
x=1,00000000000001	PRAWDA
x-1,00000000000001	0,000000000000000E+00
(x-1,00000000000001)	-1,554312234475220E-15
sign(x-1,0000000000000001)	-1,000000000000000E+00

liczba x	$1 + 45/2^{52}$
wypisane x	1,00000000000000010E+00
x=1,0000000000000001	PRAWDA
x-1,0000000000000001	0,000000000000000E+00
(x-1,0000000000000001)	0,000000000000000E+00
sign(x-1,0000000000000001)	0,000000000000000E+00

liczba x	$1 + 46/2^{52}$
wypisane x	1,00000000000000010E+00
x=1,0000000000000001	PRAWDA
x-1,0000000000000001	0,000000000000000E+00
(x-1,0000000000000001)	2,220446049250310E-16
sign(x-1,0000000000000001)	1,000000000000000E+00

liczba x	$1 + 53/2^{52}$
wypisane x	1,00000000000000010E+00
x=1,0000000000000001	PRAWDA
x-1,0000000000000001	1,776356839400250E-15
(x-1,0000000000000001)	1,776356839400250E-15
sign(x-1,0000000000000001)	1,000000000000000E+00

Trudno zgadnąć jakimi prawami rządzą się te operacje.

Przykład Testujemy teraz funkcję `round`. Rozważamy tu wartości zmienopozycyjne, które EXCEL wypisuje jako $1,02450000000000E+03$. Są to liczby postaci $1024.5 + c/2^{42}$ dla $c = -22, -21, \dots, 22$.

liczba x	$1024.5 - 3/2^{42}$	$1024.5 - 2/2^{42}$	$1024.5 - 0/2^{42}$
round(x)	1025	1025	1025
round(x-25)	999	1000	1000
round(x-925)	99	99	100

We wszystkich przedstawionych w tabeli przypadkach zaokrąglane liczby leżą albo pośrodku dwóch sąsiednich liczb całkowitych, albo bliżej mniejszej z nich. Dlaczego te nie leżące pośrodku raz zaokrąglane są w góre, a innym razem w dół?

Najłatwiejszym sposobem pozbycia się przedstawionych anomalii byłoby zastąpienie arytmetyki binarnej dziesiątną. Jej niepodważalną zaletą jest to, że liczby kodowane w bitach i wyświetlane na ekranie są takie same.

Innym (zastępczym) rozwiązańiem byłoby wyświetlanie zamiast obecnych piętnastu, siedemnaście cyfr, ale zezwolenie na wprowadzanie jedynie piętnastu. Gwarantowałoby to, że różne liczby zmiennopozycyjne miałyby odmienny wygląd, niestety czasami dziwny (np. po wpisaniu 8.04 ukazywałoby się 8.0399999999999991), co jednak byłoby poprawne pod względem numerycznym.

2 Uwarunkowanie zadania, numeryczna poprawność i stabilność algorytmu

2.1 Czym zajmuje się analiza numeryczna

Analiza numeryczna zajmuje się oszacowaniem dokładności wyników obliczeń obarczonych błędami. Źródłami błędów mogą być:

- błędy modelu (konieczność zastąpienia zbyt złożonych rzeczywistości uproszczonym modelem matematycznym),
 - niedokładności danych wejściowych (np. pomiarów),
 - błędy zaokrągleń (liczby rzeczywiste reprezentowane przez skończoną ilość bitów),
 - błędy aproksymacji (konieczność zastąpienia zbyt długiego procesu procesem dostatecznie krótkim, np. zastąpienie szeregu nieskończonego sumą skończoną).

Szczególny nacisk kładzie się na badanie, jak zachowują się algorytmy realizowane w arytmetyce komputera oraz na tworzenie wiarygodnych algorytmów — pod względem dokładności produkowanych wyników oraz dobrych ze względu na koszty: czasowy i pamięciowy.

2.2 Historia

Rozwój metod numerycznych rozpoczął się kilkaset lat temu (przykładem zastosowań „pozakomputerowych” są poprawki do tablic, przybliżone obliczanie całek przy pomocy kwadratur). Burzliwy rozwój nastąpił w ostatnich siedemdziesięciu latach, wraz z rozwojem komputerów. Istotny postęp dokonał się na początku lat sześćdziesiątych — Wilkinson wprowadził metodę analizy przez konstrukcję pozornych zaburzeń. Warto podkreślić, że w latach 40-tych, 50-tych i jeszcze 60-tych komputery były używane głównie do obliczeń naukowych, stąd wzięło się powodzenie analizy numerycznej. Metody numeryczne wykorzystywane są w obliczeniach inżynierijnych (wojskowe, telekomunikacyjne, modelowanie powierzchni), ekonomicznych (optymalizacja), medycznych (zastosowania grafiki komputerowej, np. tomografia) itp.

2.3 Po co uczymy się metod numerycznych

Istnieje wiele świetnie opracowanych pakietów programów numerycznych, mogłoby się wydawać zbędne studiowanie tej dziedziny. Jednak

- potrzeba umieć interpretować otrzymane wyniki,
 - trzeba potrafić wybrać wśród podobnych programów ten, który najlepiej pasuje do rozwiązywanego problemu,
 - w praktyce nie zawsze wystarczy sięgnąć po gotową receptę.

- metody numeryczne pozwalają nabierać nawyku uważnego spojrzenia na z pozoru proste zadania informatyczne, np. wystrzegać się sytuacji prowadzących do nadmiaru, unikać niepotrzebnych działań arytmetycznych.

Przykład Gdy chcemy wyznaczyć odległość punktu (x, y) od początku układu współrzędnych i zastosujemy algorytm

`sqrt(pow(x, 2)+pow(y, 2))`

to dość łatwo wyjdziemy poza zakres arytmetyki. Stosując inny algorytm możemy tego uniknąć.

2.4 O błędach zaokrągleń

Przypomnijmy, że jeśli reprezentujemy liczbę rzeczywistą x w arytmetyce zmiennopozycyjnej, to może zajść jedna z poniższych sytuacji (zakładamy zaokrąglanie do najbliższej).

- $x = 0$
Wtedy $r_B(0) = 0$, a błąd względny nie jest określony.
- $0 < |x| < \text{MIN}$
Błąd bezwzględny można oszacować przez

$$|r_B(x)| \leq \text{MIN } \nu.$$

Jest on bardzo mały. Natomiast błąd względny może być dowolnie duży.

- $\text{MIN} \leq |x| < \text{MAX} + \frac{H}{2}$
Wtedy

$$\begin{aligned} |r_B(x)| &\leq |x|\nu, \\ r_W(x) &\leq \nu. \end{aligned}$$

Okazuje się, że ostatnią nierówność można w sposób równoważny zapisać następująco

$$rd(x) = x(1 + \varepsilon) \quad \text{dla pewnego } |\varepsilon| \leq \nu.$$

- $|x| \geq \text{MAX} + \frac{H}{2}$
Wtedy $rd(x) = +\infty$.

Analogiczne własności dotyczą działań arytmetycznych, bo założenia wynik działania w arytmetyce fl ma być zaokrągleniem dokładnego wyniku.⁹ Zatem jeśli x i y są zmiennopozycyjne, to

$$\begin{aligned} fl(x + y) &= rd(x + y), \\ fl(x - y) &= rd(x - y), \\ fl(x * y) &= rd(x * y), \\ fl(x/y) &= rd(x/y), \quad \text{dla } y \neq 0, \\ fl(sqrt(x)) &= rd(\sqrt{x}), \quad \text{dla } x \geq 0, \\ fl(FMA(x, y, z)) &= rd(x * y + z). \end{aligned}$$

Analizy błędów przeprowadzane dla zadań i algorytmów wykonywane są przy założeniu, że wszystkie pojawiające się

⁹Sformułowanie „arytmetyka fl ” jest synonimem wyrażenia „arytmetyka zmiennopozycyjna”.

liczby leżą w przedziale $\text{MIN} \leq |x| < \text{MAX} + \frac{H}{2}$. Nie uwzględnia się więc zjawiska nadmiaru, ani niedomiaru (są one zwykle omawiane osobno, jako szczególnie przypadki danych). Dla celów analiz przyjmuje się zatem

$$\begin{aligned} \exists_{|\varepsilon| \leq \nu} \quad rd(x) &= x(1 + \varepsilon), \\ \exists_{|\varepsilon| \leq \nu} \quad fl(x + y) &= (x + y)(1 + \varepsilon), \\ \exists_{|\varepsilon| \leq \nu} \quad fl(x - y) &= (x - y)(1 + \varepsilon), \\ \exists_{|\varepsilon| \leq \nu} \quad fl(x * y) &= (x * y)(1 + \varepsilon), \\ \exists_{|\varepsilon| \leq \nu} \quad fl(x/y) &= (x/y)(1 + \varepsilon), \quad \text{dla } x \neq 0, \\ \exists_{|\varepsilon| \leq \nu} \quad fl(sqrt(x)) &= (\sqrt{x})(1 + \varepsilon), \quad \text{dla } x \geq 0, \\ \exists_{|\varepsilon| \leq \nu} \quad fl(FMA(x, y, z)) &= (x * y + z)(1 + \varepsilon). \end{aligned}$$

2.5 Zadania numeryczne

Rozwiązywanie problemu numerycznego polega na wyznaczaniu wyników $w_1, w_2, \dots, w_m \in \mathbb{R}$ na podstawie danych $d_1, d_2, \dots, d_n \in \mathbb{R}$, przy czym zależność między danymi i wynikami jest ustalona. Liczby n i m są naturalne i charakteryzuje odpowiednio rozmiary danych wejściowych i wyjściowych.

Definicja 5. *Zadanie numeryczne* jest to funkcja ϕ o dziedzinie zawartej w \mathbb{R}^n i przeciwdziedzinie w \mathbb{R}^m , gdzie m i n są liczbami naturalnymi:

$$\begin{aligned} \phi : D &\rightarrow \mathbb{R}^m, \quad D \subset \mathbb{R}^n, \\ \phi(d_1, d_2, \dots, d_n) &= [w_1, w_2, \dots, w_m]^T. \end{aligned}$$

Minimalnym wymogiem dotyczącym funkcji ϕ jest jej ciągłość. W innym przypadku niewielkie zaburzenie danych (z którym musimy się przecież liczyć, bo występują błędy zaokrągleń) prowadziły do dużych zmian wyników.

Nie każde zadanie można rozwiązać przy pomocy komputera, tzn. nie zawsze wynik, uzyskany nawet przy zastosowaniu najlepszych algorytmów, będzie wiarygodny.

Przykład (Wilkinson) Rozpatrzmy zadanie obliczania wszystkich pierwiastków wielomianu w dwudziestego stopnia

$$w(x) = a_{20}x^{20} + \dots + a_1x + a_0$$

na podstawie danych współczynników $a_i \in \mathbb{R}$ dla $i = 0, 1, \dots, n$. Zadanie można tu potraktować jako funkcję

$$\phi : \mathbb{R}^{21} \rightarrow \mathbb{R}^{40},$$

gdyż wielomian stopnia 20-go posiada dokładnie 20 pierwiastków zespolonych, o ile policzymy je z uwzględnieniem krotności. Zatem potrzebujemy dwóch liczb rzeczywistych do opisu jednego pierwiastka.

Dla zilustrowania problemów „wrażliwości” zadań na dane wybierzmy wielomian w , którego pierwiastkami są liczby $1, 2, \dots, 20$

$$\begin{aligned} w(x) &= (x - 1)(x - 2)(x - 3) \dots (x - 19)(x - 20) \\ &= a_{20}x^{20} + a_{19}x^{19} + a_{18}x^{18} + \dots + a_1x + a_0. \end{aligned}$$

Jego współczynniki można obliczyć, w szczególności

$$a_{19} = -(1 + 2 + \dots + 20) = -210.$$

Zaburzymy teraz nieznacznie tylko jeden z jego współczynników: zamiast $a_{19} = -210$ weźmiemy $\tilde{a}_{19} = a_{19}(1 - \varepsilon)$ przy $\varepsilon = (2^{23} \cdot 210)^{-1}$. Otrzymamy wielomian \tilde{w}

$$\tilde{w}(x) = a_{20}x^{20} + \tilde{a}_{19}x^{19} + a_{18}x^{18} + \dots + a_1x + a_0.$$

niewiele różniący się od w .

Okazuje się jednak, że ten nowy wielomian ma już miejsca zerowe zespolone i najbliższymi liczbie 15 są pierwiastki $13.992358137 \pm 2.518830070i$. Błąd wzgledny wyniku dla pierwiastka 15 wynosi zatem

$$\begin{aligned} & \frac{|13.992358137 \pm 2.518830070i - 15|}{|15|} \\ &= \frac{\sqrt{1.007641863^2 + 2.51883007^2}}{15} > 0.05. \end{aligned}$$

Przypomnijmy, że błąd wzgledny tylko jednej z danych wynosi zaledwie

$$\frac{|\tilde{a}_{19} - a_{19}|}{|a_{19}|} = \frac{1}{2^{23} \cdot 210} < 0.06 \cdot 10^{-8}.$$

Zatem niewielki błąd danej przeniósł się na wynik z mnożnikiem rzędu 10^8 .

2.6 Uwarunkowanie zadania

Jeśli niewielkie wzgledne zmiany danych zadania powodują niewielkie wzgledne zmiany jego rozwiązań to zadanie takie nazywamy *dobrze uwarunkowanym* (dla tych danych).

Podczas badania uwarunkowania zadania (jak już było wspomniane) zakładamy zawsze, że reprezentacje danych i wyników są liczbami normalnymi. Nie dochodzi więc ani do nadmiaru, ani niedomiaru, w szczególności wynik zadania nie jest równy 0.

Rozpatrzmy zatem pewne zadanie

$$\phi : D \rightarrow \mathbb{R}^m, \quad D \subset \mathbb{R}^n,$$

i pewne dane (dokładne)

$$d_1, d_2, \dots, d_n.$$

Ich reprezentacje zmiennopozycyjne można przedstawić jako

$$d_1(1 + \varepsilon_1), d_2(1 + \varepsilon_2), \dots, d_n(1 + \varepsilon_n),$$

gdzie $|\varepsilon_i| \leq \nu$ dla $i = 1, 2, \dots, n$. Są one zaburzone z małym błędem wzglednym:

$$\frac{|d_i(1 + \varepsilon_i) - d_i|}{|d_i|} = |\varepsilon_i| \leq \nu.$$

Badanie uwarunkowania polega na badaniu, jak te małe wzgledne błędy danych wpłyną na wyniki, a dokładniej na błąd wzgledny wyników:

$$\frac{\|\phi(d_1(1 + \varepsilon_1), \dots, d_n(1 + \varepsilon_n)) - \phi(d_1, d_2, \dots, d_n)\|_p}{\|\phi(d_1, d_2, \dots, d_n)\|_p},$$

gdzie $p \in [1, \infty]$. Zauważmy, że w przypadku $m = 1$ normy są zastąpione modułami.

Choć w konkretnej arytmetyce liczby ε_i są dla danych d_i wyznaczone w sposób jednoznaczny, my będziemy dopuszczać wszystkie możliwe zaburzenia spełniające warunek $|\varepsilon_i| \leq \nu$, czyli badać wielkość

$$U(d_1, \dots, d_n) = \sup_{\substack{|\varepsilon_i| \leq \nu \\ i=1, \dots, n}} \frac{\|\phi(d_1(1 + \varepsilon_1), \dots, d_n(1 + \varepsilon_n)) - \phi(d_1, \dots, d_n)\|_p}{\|\phi(d_1, d_2, \dots, d_n)\|_p}.$$

Po pierwsze, taka analiza jest prostsza. Po drugie błędy mogą pochodzić również z innych źródeł, niż reprezentacja liczb rzeczywistych w pamięci komputera: np. mogą to być błędy pomiarów. Takie podejście jest zatem ogólniejsze.

Definicja 6. Liczbę

$$\text{cond}_p(d_1, d_2, \dots, d_n) = \frac{1}{\nu} U(d_1, d_2, \dots, d_n)$$

nazywamy *p-tym wskaźnikiem uwarunkowania zadania numerycznego* $\phi : D \rightarrow \mathbb{R}^m$, $D \subset \mathbb{R}^n$ dla danych d_1, \dots, d_n , gdzie

$$U(d_1, d_2, \dots, d_n) = \sup_{\substack{|\varepsilon_i| \leq \nu \\ i=1, \dots, n}} \frac{\|\phi(d_1(1 + \varepsilon_1), \dots, d_n(1 + \varepsilon_n)) - \phi(d_1, \dots, d_n)\|_p}{\|\phi(d_1, d_2, \dots, d_n)\|_p}.$$

Wskaźnik uwarunkowania informuje o tym, z jakim maksymalnie mnożnikiem błędy wzgledne danych mogą się przesunąć na błędy wzgledne wyniku. Jeśli cond_p dla danych d_1, d_2, \dots, d_n przyjmuje niewielką wartość (np. 2, 10, 100), to zadanie jest dobrze uwarunkowane. Określenie, kiedy wskaźnik uwarunkowania jest niewielki zależy od precyzji arytmetyki i żądanej dokładności wyników. Wskaźnik uwarunkowania może być również pomocny przy wyborze arytmetyki: sugerować użycie silniejszej, gdy jego wielkość nie jest bardzo mała. W sposób nieformalny liczbę

$$\log_2(\nu) - \log_2(\text{cond}_p(d_1, d_2, \dots, d_n))$$

można interpretować jako liczbę cyfr znaczących wyniku, którym można ufać.

Dla wielu zadań dokładne wyznaczenie $U(d_1, \dots, d_n)$ jest bardzo trudne, zadowalamy się zwykle w miarę dokładnymi oszacowaniami tej wielkości. Zatem staramy się znaleźć liczby $A(d_1, \dots, d_n, \nu)$ i $B(d_1, \dots, d_n, \nu)$ różniące się co najwyżej kilu- lub kilkudziesięciokrotnie od siebie takie, że

$$\nu B(d_1, \dots, d_n, \nu) \leq U(d_1, \dots, d_n) \leq \nu A(d_1, \dots, d_n, \nu).$$

Zdarza się, że liczbę $A(d_1, \dots, d_n, \nu)$, lub częściej jej przybliżenie nie zależące od ν i wyrażające się w miarę czytelnym wzorem, przyjmujemy za wskaźnik uwarunkowania. Jest to podyktowane czasem tradycją dotyczącą konkretnego zadania i zwykle wygodą.

Podkreślimy, że uwarunkowanie jest cechą samego zadania, niezależną od metody jego rozwiązania. Przed przystąpieniem

do szukania algorytmów, badamy uwarunkowanie, by wieǳieć, czy w ogóle jest sens rozwiązywać zadanie przy pomocy komputera. Być może trzeba zawęzić dziedzinę zadania. Możemy również uzyskać pierwsze wskazówki, jakiej arytmetyki powinniśmy użyć. Ponadto niektóre zadania można przeformułować tak, by uzyskać zadanie równoważne matematycznie, ale dużo lepiej uwarunkowane.

Przykład Zbadajmy uwarunkowanie zadania $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}$ obliczania wartości wyrażenia $\phi(d_1, d_2) = d_1 + d_2$. Ponieważ przeciwdziedziną funkcji ϕ jest \mathbb{R} , to zamiast norm można użyć wartości bezwzględnej. Założymy, że $|\varepsilon_1|, |\varepsilon_2| \leq \nu$, wtedy

$$\begin{aligned} & \frac{|\phi(d_1(1 + \varepsilon_1), d_2(1 + \varepsilon_2)) - \phi(d_1, d_2)|}{|\phi(d_1, d_2)|} \\ &= \frac{|d_1(1 + \varepsilon_1) + d_2(1 + \varepsilon_2) - (d_1 + d_2)|}{|d_1 + d_2|} \\ &= \frac{|d_1\varepsilon_1 + d_2\varepsilon_2|}{|d_1 + d_2|} \leq \frac{|d_1\varepsilon_1| + |d_2\varepsilon_2|}{|d_1 + d_2|} \\ &= \frac{|d_1||\varepsilon_1| + |d_2||\varepsilon_2|}{|d_1 + d_2|} \leq \frac{|d_1|\nu + |d_2|\nu}{|d_1 + d_2|} = \frac{|d_1| + |d_2|}{|d_1 + d_2|}\nu \end{aligned}$$

Oszacowanie to jest dokładne, bo dla $\varepsilon_1 = \text{sign}(d_1)\nu$ oraz $\varepsilon_2 = \text{sign}(d_2)\nu$ wszystkie nierówności stałyby się równościami.

Zatem

$$U(d_1, d_2) = \nu \frac{|d_1| + |d_2|}{|d_1 + d_2|}$$

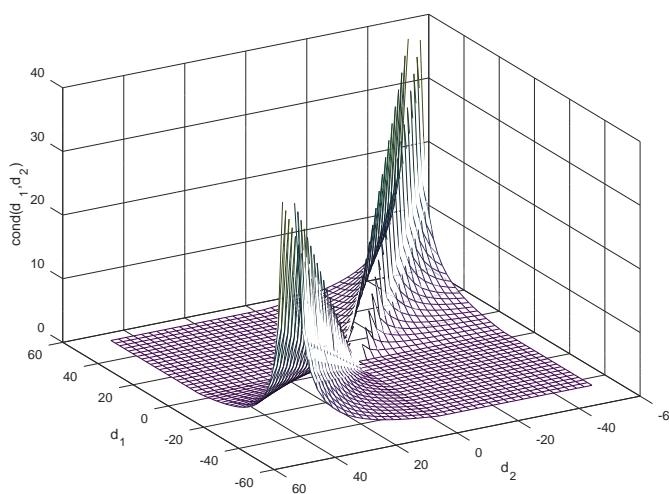
i

$$\text{cond}(d_1, d_2) = \frac{|d_1| + |d_2|}{|d_1 + d_2|}.$$

Uwarunkowanie zadania zależy od tego, czy $\text{cond}(d_1, d_2)$ jest duży, czy mały. Jeśli d_1 oraz d_2 są jednakowych znaków, to $\text{cond}(d_1, d_2) = 1$. Zadanie jest wtedy bardzo dobrze uwarunkowane. Jeśli jednak d_1 i d_2 mają różne znaki, to cond może być duże. Dzieje się to wtedy, gdy d_1 i d_2 są stosunkowo duże i mają podobne wartości modułów, np. gdy $d_1 > \frac{1}{2}$ i $d_2 = -(d_1 + \nu)$. Wtedy

$$\text{cond}(d_1, d_2) = \frac{d_1 + d_1 + \nu}{\nu} = \frac{2d_1 + \nu}{\nu} > \frac{1 + \nu}{\nu}$$

i błąd wzgledny wyniku może być rzędu $1 + \nu$, co oznacza utratę 100% informacji. Poniżej zamieszczam wykres funkcji $\text{cond}(d_1, d_2)$.



Przykład Zbadajmy uwarunkowanie zadania obliczania wyrażenia

$$S = S(a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_n) = \sum_{i=1}^n a_i b_i,$$

gdzie liczby a_i, b_i są rzeczywiste takie, że $S \neq 0$. Zatem dane dokładne to $a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_n$. Dane lekko zaburzone to $a_i(1 + \alpha_i), b_i(1 + \beta_i)$ dla $i = 1, 2, \dots, n$, gdzie $|\alpha_i| \leq \nu, |\beta_i| \leq \nu$. Wynik dla danych dokładnych to

$$S = \sum_{i=1}^n a_i b_i,$$

a dla danych zaburzonych

$$\tilde{S} = \sum_{i=1}^n a_i(1 + \alpha_i)b_i(1 + \beta_i).$$

Aby zbadać uwarunkowanie badamy błąd wzgledny rozwiązania

$$\begin{aligned} \frac{|\tilde{S} - S|}{|S|} &= \left| \frac{\sum_{i=1}^n a_i b_i (1 + \alpha_i)(1 + \beta_i) - \sum_{i=1}^n a_i b_i}{\sum_{i=1}^n a_i b_i} \right| \\ &= \left| \frac{\sum_{i=1}^n a_i b_i (1 + \alpha_i + \beta_i + \alpha_i \beta_i) - \sum_{i=1}^n a_i b_i}{\sum_{i=1}^n a_i b_i} \right| \\ &= \left| \frac{\sum_{i=1}^n a_i b_i (\alpha_i + \beta_i + \alpha_i \beta_i)}{\sum_{i=1}^n a_i b_i} \right| \\ &\leq \frac{\sum_{i=1}^n |a_i||b_i|(|\alpha_i| + |\beta_i| + |\alpha_i||\beta_i|)}{|\sum_{i=1}^n a_i b_i|} \\ &\leq \frac{\sum_{i=1}^n |a_i b_i|}{|\sum_{i=1}^n a_i b_i|} (2\nu + \nu^2) \\ &= (2 + \nu) \frac{\sum_{i=1}^n |a_i b_i|}{\left| \sum_{i=1}^n a_i b_i \right|} \nu. \end{aligned}$$

Zatem

$$U(a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_n) \leq (2 + \nu) \frac{\sum_{i=1}^n |a_i b_i|}{\left| \sum_{i=1}^n a_i b_i \right|} \nu.$$

Z drugiej strony, biorąc $\alpha_i = \beta_i = \text{sign}(a_i b_i)\nu$ otrzymujemy, że wszystkie wyrazy w sumie

$$\sum_{i=1}^n a_i b_i (\alpha_i + \beta_i + \alpha_i \beta_i)$$

są nieujemne, zatem

$$\begin{aligned}
 & U(a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_n) \\
 & \geq \left| \frac{\sum_{i=1}^n a_i b_i (\alpha_i + \beta_i + \alpha_i \beta_i)}{\sum_{i=1}^n a_i b_i} \right| \\
 & = \frac{\sum_{i=1}^n a_i b_i (\alpha_i + \beta_i + \alpha_i \beta_i)}{\left| \sum_{i=1}^n a_i b_i \right|} \\
 & = \frac{\sum_{i=1}^n a_i b_i (2 \operatorname{sign}(a_i b_i) \nu + \nu^2)}{\left| \sum_{i=1}^n a_i b_i \right|} \\
 & = \frac{\sum_{i=1}^n (2|a_i b_i| \nu + a_i b_i \nu^2)}{\left| \sum_{i=1}^n a_i b_i \right|} \\
 & \geq \frac{\sum_{i=1}^n (2|a_i b_i| \nu - |a_i b_i| \nu^2)}{\left| \sum_{i=1}^n a_i b_i \right|} \\
 & = (2 - \nu) \frac{\sum_{i=1}^n |a_i b_i|}{\left| \sum_{i=1}^n a_i b_i \right|} \nu.
 \end{aligned}$$

Ponieważ oszacowania górne i dolne są bliskie sobie, to za wskaźnik uwarunkowania można przyjąć

$$\operatorname{cond} = \operatorname{cond}(a_1, \dots, a_n, b_1, \dots, b_n) = 2 \frac{\sum_{i=1}^n |a_i b_i|}{\left| \sum_{i=1}^n a_i b_i \right|}.$$

Małe błędy względne α_i, β_i danych mogą się przenieść na wynik z mnożnikiem cond. Jeśli wielkość cond będzie duża, to błąd względny rozwiązania będzie duży i nie ma sensu rozwiązywać zadania, o ile nie użyjemy bardzo silnej arytmetyki.

Zbadajmy wielkość cond. Jeśli składniki $a_i b_i$ są jednakowego znaku, to cond = 2 i zadanie jest bardzo dobrze uwarunkowane. Można jednak tak dobrać składniki $a_i b_i$, że są one dużymi liczbami co do wartości bezwzględnej i mają takie znaki, że suma $\sum_{i=1}^n a_i b_i$ jest bliska零. Wtedy cond może być bardzo duży i zadanie źle uwarunkowane.

2.7 Algorytmy numerycznie poprawne

Jeśli po zbadaniu uwarunkowania zadania wiemy, że jest sens do rozwiązywać w dostępnej arytmetyce, przystępujemy do opracowania algorytmu. Algorytm, podobnie jak zadanie, jest funkcją $\mathcal{A} : \mathbb{R}^n \supset D_1 \rightarrow \mathbb{R}^m$, gdzie liczby naturalne m i n są wyznaczone przez zadanie. Dodatkowo przyjmujemy, że dla każdego algorytmu znana jest kolejność wykonywania wszystkich działań i że jest ich skończenie wiele.¹⁰

¹⁰ Traktowanie algorytmu jako funkcji przyporządkowującej danym wynikom jest niewystarczające. Jednak formalne ujęcie całej definicji (czyli

Gdyby działania wykonywane były dokładnie i liczby rzeczywiste przechowywane bez błędów zaokrągleń algorytmy produkowałyby dokładne rozwiązania (co oznaczałoby, że $\mathcal{A} = \phi$), lub ich dobre aproksymacje ($\mathcal{A} \approx \phi$), w zależności od tego, czy zadanie daje się rozwiązać w skończonej liczbie kroków, czy nie. Musimy jednak się liczyć z błędami zniekształcającymi obliczenia i pogodzić się z tym, że niemożliwe jest uzyskanie dokładnego wyniku. Gotowy algorytm należy więc umieć ocenić, czy produkuje on dopuszczalne wyniki.

Zastanówmy się jak dokładny może być dla danego zadania wynik obliczony w arytmetyce zmienopozycyjnej. Zauważmy, że

1. zamiast zadania o dokładnych danych, będziemy rozwiązywali numerycznie zadanie o danych zaburzonych przez błędy reprezentacji,
2. gdybyśmy nawet znali dokładne rozwiązanie dla tych danych, to w arytmetyce fl byłoby ono reprezentowane na ogólny sposób przybliżony.

Z tych względów za numerycznie najwyższej jakości uznamy takie algorytmy, dla których obliczone w arytmetyce fl rozwiązanie jest nieco zaburzonym rozwiązaniem zadania o nieco zaburzonych danych. Takie algorytmy nazywamy *numerycznie poprawnymi*. Zakładamy przy tym, że nie dochodzi ani do nadmiaru, ani do niedomiaru, a zaburzenia są mierzone w sposób względny.

Definicja 7. Niech $\text{fl}(\mathcal{A}(d_1, \dots, d_n))$ oznacza wynik algorytmu \mathcal{A} dla wektora danych (d_1, \dots, d_n) obliczony w arytmetyce fl. Algorytm \mathcal{A} jest *numerycznie poprawny* jeśli dla każdego wektora danych można dobrać liczby σ_i, δ_i tak, by spełniona była zależność

$$\begin{aligned}
 \text{fl}(\mathcal{A}(d_1, \dots, d_n)) &= \\
 &= \begin{bmatrix} 1 + \sigma_1 & & 0 \\ & \ddots & \\ 0 & & 1 + \sigma_m \end{bmatrix} \phi(d_1(1 + \delta_1), \dots, d_n(1 + \delta_n)),
 \end{aligned}$$

przy czym muszą zachodzić nierówności $|\sigma_i| \leq C_i \nu$, $|\delta_i| \leq K_i \nu$ dla niewielkich stałych K_i, C_i niezależnych od danych. Liczby K_i, C_i te nazywamy *wskaźnikami kumulacji*.¹¹

Przyjrzyjmy się powyższej równości. Przypominam, że $\phi(d_1(1 + \delta_1), \dots, d_n(1 + \delta_n))$ jest wektorem przestrzeni \mathbb{R}^m i oznacza dokładny wynik (a raczej m składowych wyników) zadania wyznaczony dla danych nieco zaburzonych. Zaburzenia są wprowadzone przez mnożniki $(1 + \delta_i)$, a ich małość gwarantować mają nierówności $|\delta_i| \leq K_i \nu$ (o ile K_i są małe). Stojąca obok macierz diagonalna pozwala na zanotowanie faktu, że każdy z m wyników składowych może być też nieco zaburzony — po wymnożeniu przy i -tym wyniku składowym pojawi się $(1 + \sigma_i)$. Prawą stronę równości można zatem przeczytać następująco

włączenie kolejności działań) bardzo ją komplikuje. Pozostawiam więc podejście nieformalne. Myślę, że jest wystarczająco intuicyjne.

¹¹ Stosowane są też inne definicje numerycznej poprawności, np. niższa.

nieco zaburzone rozwiążanie zadania dla danych nieco zaburzonych.

Prawa strona nie zawiera pojęcia algorytmu. Pojawia się ono jedynie po lewej — zgodnie z definicją $\text{fl}(\mathcal{A}(d_1, \dots, d_n))$ oznacza wynik algorytmu \mathcal{A} dla danych (d_1, \dots, d_n) obliczony w arytmetyce fl .

Własność numerycznej poprawności algorytmów udowadnia się stosując technikę zwaną *konstrukcją pozornych zaburzeń*. Sprowadza się ona do wykazania, że błędy wytwarzane w poszczególnych krokach algorytmu są równoważne skonstruowanym sztucznie błędem danych początkowych.

Przykład Obliczając w arytmetyce fl wartość wielomianu $w(x, y) = x^2 + 4y^2$ możemy się posłużyć algorytmem

$$\mathcal{A}(x, y) = (x * x) + ((4 * y) * y);$$

Zakładamy, że nie powstanie nadmiar ani niedomiar, a działania są wykonywane od lewej do prawej i zgodnie z nawiasami. Podczas wykonywania algorytmu najpierw musimy zastąpić dane oryginalne x oraz y danymi zaburzonymi $\text{rd}(x) = x(1 + \varepsilon_x)$ i $\text{rd}(y) = y(1 + \varepsilon_y)$. Podczas wykonania każdego z działań znów pojawia się błędy zaokrągleń, wyrażone przez $\varepsilon_1, \varepsilon_2, \varepsilon_3$. Wiemy ponadto, że $|\varepsilon_x|, |\varepsilon_y|, |\varepsilon_1|, |\varepsilon_2|, |\varepsilon_3| \leq \nu$. Zatem

$$\begin{aligned} \text{fl}(\mathcal{A}(x, y)) &= [x(1 + \varepsilon_x) \cdot x(1 + \varepsilon_x)(1 + \varepsilon_1) \\ &\quad + 4y(1 + \varepsilon_y) \cdot y(1 + \varepsilon_y)(1 + \varepsilon_2)](1 + \varepsilon_3) \\ &= [(x(1 + \varepsilon_x)\sqrt{1 + \varepsilon_1})^2 + 4(y(1 + \varepsilon_y)\sqrt{1 + \varepsilon_2})^2](1 + \varepsilon_3) \\ &= [(x(1 + \delta_1))^2 + 4(y(1 + \delta_2))^2](1 + \delta_0) \\ &= (1 + \delta_0)w(x(1 + \delta_1), y(1 + \delta_2)) \end{aligned}$$

gdzie

$$\begin{aligned} 1 + \delta_0 &= 1 + \varepsilon_3, \\ 1 + \delta_1 &= (1 + \varepsilon_x)\sqrt{1 + \varepsilon_1}, \\ 1 + \delta_2 &= (1 + \varepsilon_y)\sqrt{1 + \varepsilon_2}. \end{aligned}$$

Definicja 8. Niech $\text{fl}(\mathcal{A}(\vec{d}))$ oznacza wynik algorytmu \mathcal{A} dla wektora danych \vec{d} obliczony w arytmetyce fl . Algorytm \mathcal{A} jest *numerycznie poprawny* jeśli dla każdego wektora danych \vec{d} można dobrać wektor danych $\tilde{\vec{d}}$ tak, by spełniona była zależność

$$\frac{\|\text{fl}(\mathcal{A}(\vec{d})) - \phi(\tilde{\vec{d}})\|}{\|\phi(\tilde{\vec{d}})\|} \leq K\nu,$$

przy czym $\tilde{\vec{d}}$ musi spełniać nierówność

$$\frac{\|\vec{d} - \tilde{\vec{d}}\|}{\|\vec{d}\|} \leq C\nu.$$

Liczby K, C są niezależnymi od danych niewielkimi stałymi, nazywamy je *wskaznikami kumulacji*. Zakładamy przy tym, że w powyższych nierównościach nie dochodzi do dzielenia przez zero. Oznaczonemu wspólnym symbolem normy w przestrzeniach zawierających odpowiednio przeciwdziedzinę i dziedzinę zadania ϕ nie muszą być takie same.

Niektórzy uważają algorytm za numerycznie poprawny, gdy można znaleźć dostatecznie silną arytmetykę (czyli dostatecznie małe ν) taką, że powyższa definicja zachodzi. Dopuszczają też dowolną wielkość stałych kumulacji.

Zauważmy, że mnożenie przez potęgi dwójki jest wykonywane dokładnie (powiększenie wykładnika). By udowodnić numeryczną poprawność algorytmu pozostaje jeszcze wykazać, że $|\delta_i| \leq K_i\nu$ dla $i = 0, 1, 2$ przy stałych K_0, K_1 i K_2 niewielkich w stosunku do ν , najlepiej bliskich jedności. W sposób oczywisty $K_0 = 1$. Oszacujmy wielkość $|\delta_1|$

$$\begin{aligned} |\delta_1| &= |(1 + \varepsilon_x)\sqrt{1 + \varepsilon_1} - 1| \\ &= \frac{|(1 + \varepsilon_x)^2(1 + \varepsilon_1) - 1|}{(1 + \varepsilon_x)\sqrt{1 + \varepsilon_1} + 1} \\ &= \frac{|2\varepsilon_x + \varepsilon_x^2 + \varepsilon_1 + 2\varepsilon_1\varepsilon_x + \varepsilon_1\varepsilon_x^2|}{(1 + \varepsilon_x)\sqrt{1 + \varepsilon_1} + 1} \\ &\leq \frac{2|\varepsilon_x| + |\varepsilon_x|^2 + |\varepsilon_1| + 2|\varepsilon_1||\varepsilon_x| + |\varepsilon_1||\varepsilon_x|^2}{(1 + \varepsilon_x)\sqrt{1 + \varepsilon_1} + 1} \\ &\leq \frac{3\nu + 3\nu^2 + \nu^3}{(1 - \nu)^{\frac{3}{2}} + 1} \\ &= \nu \frac{3 + 3\nu + \nu^2}{(1 - \nu)^{\frac{3}{2}} + 1}. \end{aligned}$$

Biorąc

$$K_1 = \frac{3 + 3\nu + \nu^2}{(1 - \nu)^{\frac{3}{2}} + 1} \approx \frac{3}{2}$$

otrzymujemy, że $|\delta_1| \leq K_1\nu$. Analogicznie można udowodnić, że $K_2 \approx \frac{3}{2}$. Zastąpiliśmy błędy zaokrągleń (parametry $\varepsilon_x, \varepsilon_y, \varepsilon_1, \varepsilon_2$ i ε_3) zaburzeniami danych (parametry δ_1 i δ_2) oraz wyniku (δ_0). Zaburzenia te są niewielkie. Oczywiście zastępowanie jest pozorne — faktycznie podczas obliczeń występują błędy zaokrągleń. Jednak przedstawiona analiza pozwala stwierdzić, że obliczony powyższym algorytmem wynik jest nieco zaburzonym rozwiązaniem zadania sąsiadniego, czyli zadania o nieco zaburzonych danych. Tak więc algorytm jest numerycznie poprawny.

Zwrómy uwagę, na to że jeśli w powyższym przykładzie założylibyśmy, że dane są reprezentowane dokładnie, czyli $\varepsilon_x = \varepsilon_y = 0$, to wskaźniki kumulacji byłyby mniejsze o 1. Jest to ogólna reguła i przy dowodach numerycznej poprawności dla uproszczenia zakłada się, że dane nie są obarczone błędami reprezentacji. Tak też będziemy czynić dalej.

Przykład Przeanalizujmy jeden z algorytmów rozwiążających zadanie

$$S = S(a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_n) = \sum_{i=1}^n a_i b_i$$

dla danych a_i i b_i , $i = 1, 2, \dots, n$. Posługujemy się algorytmem \mathcal{A} :

```
S=0.0;
for(i=1;i<=n;i++)
    S=S+a[i]*b[i];
```

Przymajemy, że dane są umieszczone w tablicach **a** i **b**, oraz zakładamy, że są reprezentowane dokładnie. Wykonanie mnożenia i dodawania wewnątrz pętli powoduje błędy zaokrągleń, zatem

$$\text{fl}(\mathcal{A}(a_1, \dots, a_n, b_1, \dots, b_n)) = \{\dots \{ \{ a_1 b_1 (1 + \varepsilon_1)$$

$$\begin{aligned} & +a_2 b_2(1+\varepsilon_2)\}(1+\delta_2) \\ & +a_3 b_3(1+\varepsilon_3)\}(1+\delta_3)+\dots \\ & +a_n b_n(1+\varepsilon_n)\}(1+\delta_n). \end{aligned}$$

Po przeprowadzeniu przekształceń otrzymujemy

$$\text{fl}(\mathcal{A}(a_1, \dots, a_n, b_1, \dots, b_n)) = \sum_{i=1}^n a_i b_i (1 + E_i),$$

gdzie

$$1 + E_i = (1 + \varepsilon_i) \prod_{j=i}^n (1 + \delta_j), \quad \delta_1 = 0.$$

Dowodzi się, że

$$|E_i| \leq M_i(n - i + 2)\nu,$$

dla stałych M_i bliskich jedności. Obliczony wynik można interpretować jako dokładne rozwiązanie zadania dla trochę zaburzonych danych, czyli

$$S(a_1, a_2, \dots, a_n, b_1(1 + E_1), b_2(1 + E_2), \dots, b_n(1 + E_n)).$$

Stałe kumulacji wynoszą $K_i = M_i(n - i + 2)$. Nie będą one małe dla bardzo dużych n , czyli takich, że ν zbliża się do jedności. Zauważmy jeszcze, że błędy zaokrągleń mogą być inaczej „podzielone” pomiędzy dane a_i i b_i .

Algorytmy numeryczne poprawne są najlepszej możliwej jakości. Więcej nie można wymagać. Nie każdy jednak algorytm jest tak dobry. Należy więc ustalić jakie algorytmy są do zaakceptowania, podać minimalną własność, która musi być spełniona. Jest nią *numeryczna stabilność*, która jest omówiona w następnym punkcie.

2.8 Algorytmy numerycznie stabilne

Definicja 9. Dane jest zadanie $\phi : \mathbb{R}^n \supset D \rightarrow \mathbb{R}^m$. Optymalnym poziomem błędu zadania ϕ dla danych d nazywamy wielkość

$$\text{OPB}(d) = \nu \|\phi(d)\| + \sup_{\|d - \tilde{d}\| \leq \nu \|d\|} \|\phi(d) - \phi(\tilde{d})\|.$$

Normy są dowolne i mogą być różne dla przestrzeni danych \mathbb{R}^n oraz wyników \mathbb{R}^m .

Co wyraża optymalny poziom błędu? Pierwszy jego składnik odpowiada za maksymalny błąd bezwzględny, jaki może powstać podczas reprezentacji wyników. Obliczone wyniki muszą być przecież reprezentowane w pamięci komputera. Ponieważ zakładamy, że nie zachodzi zjawisko nadmiaru ani niedomiaru, to wiemy, że dla liczby rzeczywistej x błąd bezwzględny reprezentacji nie przekracza $\nu|x|$ (porównaj punkt 2.4). Ta nierówność jest prawdziwa dla każdego wyniku z osobna, jednak zamiast sumować maksymalne błędy wyników, co odpowiadałoby zastosowaniu normy wektorowej dla $p = 1$, stosujemy podejście ogólniejsze i dopuszczały dowolną normę wektorową. Dla niektórych zadań zmiana normy ułatwia analizę.

Drugi składnik określa, jak w najgorszym przypadku błędy danych mogą przenieść się na wynik zadania. Rozpatrujemy znowu błąd bezwzględny. Wielkość $\|\phi(d) - \phi(\tilde{d})\|$ wyraża odległość w sensie normy między wynikami dla danych dokładnych d i wynikami dla danych zaburzonych \tilde{d} . Wśród takich odległości szukamy największej rozpatrując dane zaburzone, które leżą blisko d w sensie takim, że są obarczone błędem na tym samym poziomie, co błąd reprezentacji d w pamięci komputera. Tę ostatnią bliskość rozpatrujemy znów w sensie normy: bierzemy takie \tilde{d} , które spełniają nierówność $\|d - \tilde{d}\| \leq \nu \|d\|$. Tym razem odległość nie jest bezwzględna, lecz względna, bo dla $d \neq 0$

$$\|d - \tilde{d}\| \leq \nu \|d\| \iff \frac{\|d - \tilde{d}\|}{\|d\|} \leq \nu.$$

Błąd, który jest rzędu $\text{OPB}(d)$ musi się pojawić podczas rozwiązywania zadania dowolną metodą. Zauważmy, że $\text{OPB}(d)$ nie zależy od algorytmu. Jeśli algorytm nie produkuje błędu znaczaco większego niż $\text{OPB}(d)$, to nazywamy go *numerycznie stabilnym*.

Definicja 10. Algorytm \mathcal{A} jest *numerycznie stabilny dla danych ze zbioru $B \subset \mathbb{R}^n$* jeśli dla dostatecznie silnej arytmetyki oraz dla każdego $d \in B$, błąd bezwzględny przezeń wytworzony jest jedynie K -krotnie większy niż $\text{OPB}(d)$, gdzie K jest niezbyt dużą stałą. Można to zapisać następująco

$$\forall_{d \in B} \quad |\text{fl}(\mathcal{A}(d)) - \phi(d)| \leq K \cdot \text{OPB}(d).$$

Inaczej mówiąc, błąd bezwzględny w algorytmie numerycznie stabilnym jest na poziomie nieuniknionego błędu rozwiązania wynikającego z przybliżonej reprezentacji danych i wyniku. Wielkość stałej K służy do oceny jakości algorytmu — im mniejsza, tym algorytm lepszy.

Udowodniono, że każdy algorytm numerycznie poprawny jest numerycznie stabilny. Nie jest prawdziwe stwierdzenie odwrotne, bo są algorytmy numerycznie stabilne, które nie są numerycznie poprawne.

Przykład Rozpatrzmy zadanie obliczania kwadratu liczby rzeczywistej d ,

$$\phi(d) = d^2.$$

W celu jego rozwiązania posłużymy się algorytmem:

$$\mathcal{A}(d) = d * d.$$

Sprawdźmy, czy jest on numerycznie stabilny. W tym celu wyznaczmy optymalny poziom błędu.

$$\begin{aligned} \text{OPB}(d) &= \nu |\phi(d)| + \sup_{|d - \tilde{d}| \leq \nu |d|} |\phi(d) - \phi(\tilde{d})| \\ &= \nu d^2 + \sup_{|d - \tilde{d}| \leq \nu |d|} |d^2 - \tilde{d}^2|. \end{aligned}$$

Obliczmy wartość supremum. Ponieważ

$$\tilde{d}^2 - d^2 = (d - \tilde{d})^2 - 2d((d - \tilde{d})),$$

to stosując nierówność trójkąta mamy

$$\begin{aligned} \sup_{|d-\tilde{d}| \leq \nu|d|} |d^2 - \tilde{d}^2| &\leq \sup_{|d-\tilde{d}| \leq \nu|d|} (|d - \tilde{d}|^2 + 2|d||d - \tilde{d}|) \\ &= \nu^2 d^2 + 2\nu d^2 \\ &= \nu(\nu + 2)d^2 \end{aligned}$$

Z drugiej strony jeśli $\tilde{d} = d + \nu d$ to $|d^2 - \tilde{d}^2| = \nu(\nu + 2)d^2$ oraz $|d - \tilde{d}| \leq \nu|d|$. Zatem powyższe supremum wynosi $\nu(\nu + 2)d^2$ oraz

$$\text{OPB}(d) = \nu(3 + \nu)d^2.$$

Teraz należy dokonać oszacowania błędu bezwzględnego wytworzzonego podczas obliczeń algorytmu.

$$\begin{aligned} |\text{fl}(A(d)) - \phi(d)| &= |(d(1 + \varepsilon_1)d(1 + \varepsilon_1))(1 + \varepsilon_2) - d^2| \\ &= d^2|2\varepsilon_1 + \varepsilon_2 + \varepsilon_1^2 + 2\varepsilon_1\varepsilon_2 + \varepsilon_1^2\varepsilon_2| \\ &\leq d^2\nu(3 + 3\nu + \nu^2) \\ &= K_1 \cdot \text{OPB}(d), \end{aligned}$$

dla $K_1 = \frac{3+3\nu+\nu^2}{3+\nu} \approx 1$. Algorytm \mathcal{A} jest numerycznie stabilny dla wszystkich danych.

2.9 Efektywność algorytmu

Dla wielu zadań znane są różne metody ich rozwiązywania. Spośród nich chcielibyśmy wybrać najlepszą. Powstaje problem, jakich kryteriów używać. Oczywiście jednym z nich, i zwykle najważniejszym, jest jakość algorytmu. Drugim jest koszt danej metody (czasowy i pamięciowy).

Za operacje dominujące przy analizie algorytmów numerycznych przyjmuje się zazwyczaj działania zmiennopozycyjne: dodawanie, odejmowanie, mnożenie, dzielenie i pierwiastkowanie. Zwykle oblicza się trzy wielkości

1. liczbę dodawań i odejmowań,
2. liczbę mnożeń i dziileń,
3. liczbę pierwiastkowań.

Okazuje się bowiem, że czas wykonania dodawania/odejmowania jest istotnie krótszy niż mnożenia/dzielenia, a ten znów krótszy niż pierwiastkowania. Chociaż technicznie nic nie stoi na przeszkodzie, by niemal zrównać czasy tych działań, to wielu producentów sprzętu tego nie robi. Stosują oni prostsze procesory tłumacząc, że zwykle komputer wykonuje mniej mnożeń/dziileń niż dodawań/odejmowań i nie warto przepłacać.

Szukaniem algorytmów optymalnych (np. pod względem czasowym) zajmuje się dział analizy numerycznej zwany *analityczną złożonością obliczeniową*. Znanych jest jednak stosunkowo mało algorytmów optymalnych.

Wielomiany

Dorota Dąbrowska, UKSW

2017/18

Spis treści

3 Bazy wielomianowe	
3.1 Wybrane bazy wielomianowe	24
3.2 Własności wielomianów Czebyszewa	26
3.3 Węzły równoodległe i Czebyszewa	27
3.4 Norma supremum w przestrzeni funkcji ciągłych na przedziale domkniętym	27
3.5 Miary skośności baz	28
4 Algorytmy obliczania wartości wielomianu	29
4.1 Algorytm Hornera dla bazy naturalnej	29
4.2 Uogólniony algorytm Hornera	30
4.3 Algorytmy dla bazy Czebyszewa	30
4.4 Algorytm dla bazy Lagrange'a	31
5 Algorytmy zamiany baz	32
5.1 Jeszcze raz o skośności bazy Newtona	33
5.2 Baza Lagrange'a → baza Newtona	33
5.2.1 Algorytm pierwszy — dobrej jakości .	33
5.2.2 Algorytm różnic dzielonych	35
5.2.3 Algorytm różnic dzielonych w przypadku węzłów równoodległych	37
5.3 Baza naturalna → baza Czebyszewa	37
5.4 Baza Czebyszewa → baza naturalna	39
5.5 Baza naturalna → baza Lagrange'a	39
5.6 Baza naturalna → baza Newtona	39
5.7 Baza Newtona → baza naturalna	39
6 Obliczanie pochodnych wielomianów	39
6.1 Bezpośrednie obliczenie pochodnej	39
6.2 Obliczanie pochodnych wielomianu algorymem Hornera	39
6.3 Algorytm Shaw-Trauba	40
3 Bazy wielomianowe	
24 Wielomian jest funkcją, którą można zapisać w postaci	
$w(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0.$	
Będziemy się zajmowali głównie wielomianami rzeczywistymi, czyli takimi, że $a_i \in \mathbb{R}$, $i = 0, \dots, n$ oraz $x \in \mathbb{R}$. Jeśli $a_n \neq 0$ to n jest stopniem wielomianu.	
W analizie numerycznej wielomiany mają liczne zastosowania — choćby w interpolacji, aproksymacji, czy całkowaniu numerycznym. Głównymi operacjami, jakie się na wielomianach wykonuje są: obliczanie wartości i pochodnej wielomianu w punkcie, dodawanie wielomianów, mnożenie przez liczbę.	
Tu dochodzimy do kluczowego problemu: jak dobrze przechowywać wielomiany. Dobrze — to znaczy tak, by łatwo było dokonywać operacji oraz by jakość otrzymanych wyników była zadowalająca.	
Zbiór wielomianów rzeczywistych stopnia nie większego niż n ze zwyczajowymi działaniami dodawania wielomianów i mnożenia wielomianu przez liczbę tworzy przestrzeń liniową nad ciałem \mathbb{R} . Będziemy oznaczać tę przestrzeń przez Π_n .	
3.1 Wybrane bazy wielomianowe	
Jeśli dokonamy wyboru bazy w przestrzeni Π_n , to każdy wielomian można jednoznacznie w niej zapisać. W pamięci komputera wystarczy zapamiętać jego współczynniki. ¹	
Z punktu widzenia matematyki jest właściwie wszystko jedno, którą bazę wybierzemy — to tylko kwestia zapisu, wielomian przedstawiony w tej, czy innej bazie pozostaje tym samym wielomianem. Okazuje się, że zachodzą istotne różnice, związane z jakością i wygodą obliczeń, gdy rozważamy algorytmy numeryczne. Przedstawię poniżej cztery bazy. Nie zamieszczam dowodów (nietrudnych), że rzeczywiście są to bazy.	
Baza naturalna (inaczej potęgowa)	
Bazę Π_n tworzą wielomiany	
$1, x, x^2, x^3, \dots, x^n.$	
Współczynniki	
a_0, a_1, \dots, a_n	

¹W algebrze liniowej używa się raczej sformułowania *współrzędne* w bazie, ale w stosunku do wielomianów zachowam tradycyjne nazewnictwo.

wyznaczają w tej bazie wielomian

$$w(x) = a_0 + a_1x + a_2x^2 \dots + a_nx^n = \sum_{i=0}^n a_i x^i.$$

Baza Newtona By ją zdefiniować należy najpierw wybrać n punktów (tzw. węzłów) x_1, x_2, \dots, x_n . Są one dowolne i nie muszą być różne. Wielomiany

$$1, x - x_1, (x - x_1)(x - x_2), \dots, \prod_{i=1}^n (x - x_i)$$

tworzą bazę przestrzeni Π_n . Współczynniki

$$b_0, b_1, \dots, b_n$$

definiują w tej bazie wielomian

$$\begin{aligned} w(x) &= b_0 + b_1(x - x_1) + b_2(x - x_1)(x - x_2) + \dots \\ &\quad + b_n(x - x_1)(x - x_2) \dots (x - x_n) \\ &= \sum_{k=0}^n b_k \prod_{i=1}^k (x - x_i). \end{aligned}$$

Baza Czebyszewa Elementami tej bazy są *wielomiany Czebyszewa*, które zdefiniowane są w sposób rekurencyjny

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_k(x) &= 2xT_{k-1}(x) - T_{k-2}(x), \quad \text{dla } k = 2, 3, \dots \end{aligned}$$

czyli bazą jest zestaw wielomianów

$$T_0, T_1, \dots, T_n.$$

Współczynniki

$$c_0, c_1, \dots, c_n$$

wyznaczają w tej bazie wielomian

$$w(x) = c_0T_0(x) + c_1T_1(x) + \dots + c_nT_n(x) = \sum_{i=0}^n c_i T_i(x).$$

Baza Lagrange'a Określamy ją dla $n+1$ różnych punktów (tzw. węzłów) x_0, x_1, \dots, x_n . Elementami bazy są *funkcje Lagrange'a*:

$$l_0, l_1, \dots, l_n,$$

które są zdefiniowane równościami

$$l_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^n \frac{x - x_j}{x_i - x_j} \quad i = 0, 1, \dots, n.^2$$

Współczynniki

$$d_0, d_1, \dots, d_n$$

²Ta baza istotnie różni się od pozostałych — wszystkie funkcje bazowe są dokładnie stopnia n , a poprzednio stopnie wielomianów bazowych rosły od 0 do n . Funkcje l_i wydają się dość skomplikowane. Nie powstały one jako kaprys pana Lagrange'a, lecz pojawiły się podczas rozwiązywania zadania interpolacji (Lagrange'a), o której będzie mowa później.

wyznaczają w tej bazie wielomian

$$\begin{aligned} w(x) &= d_0l_0(x) + d_1l_1(x) + \dots + d_nl_n(x) \\ &= \sum_{i=0}^n d_il_i(x) \\ &= \sum_{i=0}^n d_i \prod_{\substack{j=0 \\ j \neq i}}^n \frac{x - x_j}{x_i - x_j}. \end{aligned}$$

Przykład Dany jest wielomian

$$w(x) = 1 - 2x^2 + 4x.$$

Szukamy jego współczynników w różnych bazach przestrzeni Π_3 .³

1. Baza naturalna: $1, x, x^2, x^3$. Zauważmy, że

$$w(x) = 1 \cdot 1 + x \cdot 4 + x^2 \cdot (-2) + x^3 \cdot 0,$$

zatem współczynniki w bazie naturalnej przestrzeni Π_3 wynoszą $a_0 = 1, a_1 = 4, a_2 = -2$ i $a_3 = 0$.

2. Baza Newtona dla węzłów $x_1 = -1, x_2 = 0, x_3 = 1$ to: $1, x+1, (x+1)x, (x+1)x(x-1)$. Szukamy współczynników b_0, b_1, b_2, b_3 spełniających równość

$$w(x) = 1 \cdot b_0 + (x+1) \cdot b_1 + (x+1)x \cdot b_2 + (x+1)x(x-1) \cdot b_3$$

dla dowolnego $x \in \mathbb{R}$. Po uporządkowaniu otrzymujemy równość

$$1 + 4x - 2x^2 = b_0 + b_1 + (b_1 + b_2 - b_3)x + b_2x^2 + b_3x^3.$$

Stąd porównując współczynniki przy odpowiednich potęgach zmiennej x uzyskujemy układ równań

$$\begin{cases} b_0 + b_1 = 1, \\ b_1 + b_2 - b_3 = 4, \\ b_2 = -2, \\ b_3 = 0, \end{cases}$$

którego rozwiązaniem są liczby $b_3 = 0, b_2 = -2, b_1 = 6, b_0 = -5$. Można łatwo sprawdzić, że $w(x) = -5 + 6(x+1) - 2x(x+1)$.

3. Baza Czebyszewa: T_0, T_1, T_2, T_3 . Z definicji

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_2(x) &= 2xT_1(x) - T_0(x) = 2x^2 - 1, \\ T_3(x) &= 2xT_2(x) - T_1(x) = 4x^3 - 3x. \end{aligned}$$

Szukamy współczynników c_0, c_1, c_2 i c_3 takich, że

$$w(x) = c_0T_0(x) + c_1T_1(x) + c_2T_2(x) + c_3T_3(x),$$

czyli spełniających dla każdego $x \in \mathbb{R}$ równość

$$1 + 4x - 2x^2 = c_0 + c_1x + c_2(2x^2 - 1) + c_3(4x^3 - 3x).$$

³Ten wielomian jest też elementem Π_2, Π_4, Π_5 itd. Można go zatem zapisywać w bazach każdej z tych przestrzeni. Nie należy jednak do Π_1 i w żadnej z baz Π_1 nie da się go przedstawić.

Po uporządkowaniu i porównaniu współczynników przy odpowiednich potęgach x otrzymujemy układ równań

$$\begin{cases} c_0 - c_2 = 1, \\ c_1 - 3c_3 = 4, \\ 2c_2 = -2, \\ 4c_3 = 0. \end{cases}$$

Jego rozwiązanie to $c_3 = 0$, $c_2 = -1$, $c_1 = 4$ i $c_0 = 0$.

4. Baza Lagrange'a zdefiniowana węzłami $-1, 1, -2, 2$. Funkcjami bazowymi są tu

$$\begin{aligned} l_0(x) &= \frac{(x-1)(x+2)(x-2)}{(-1-1)(-1+2)(-1-2)}, \\ l_1(x) &= \frac{(x+1)(x+2)(x-2)}{(1+1)(1+2)(1-2)}, \\ l_2(x) &= \frac{(x+1)(x-1)(x-2)}{(-2+1)(-2-1)(-2-2)}, \\ l_3(x) &= \frac{(x+1)(x-1)(x+2)}{(2+1)(2-1)(2+2)}. \end{aligned}$$

Szukamy współczynników takich, że

$$w(x) = d_0l_0(x) + d_1l_1(x) + d_2l_2(x) + d_3l_3(x).$$

Obliczanie współczynników dla bazy Lagrange'a wydaje się dość pracochłonne (gdybyśmy chcieli do powyższego równania wstawiać wzory opisujące funkcje bazowe Lagrange'a, uporządkować równanie i porównać odpowiednie współczynniki). Można jednak współczynniki wyznaczyć znacznie szybciej. Nie zdradzę w tym miejscu jak — proszę o zastanowienie się. Trzeba znaleźć szczególną własność bazy Lagrange'a związaną z węzłami. Uzyskanie rozwiązania tego przykładu w pamięci zajęło mi około 1 min. (16 mnożeń i 8 dodawań). Wynikami są $d_0 = -5$, $d_1 = 3$, $d_2 = -15$, $d_3 = 1$.

3.2 Własności wielomianów Czebyszewa

O wielomianach Czebyszewa można pisać wiele — powstały na ich temat prace doktorskie. Wybrałam parę ich właściwości, które powinny przybliżyć Państwu te ciekawe funkcje.

1. Przyglądając się zamieszczonej definicji

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_k(x) &= 2xT_{k-1}(x) - T_{k-2}(x), \quad \text{dla } k = 2, 3, \dots, \end{aligned}$$

i wypisując kilka początkowych wielomianów

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_2(x) &= 2x^2 - 1, \\ T_3(x) &= 4x^3 - 3x, \\ T_4(x) &= 8x^4 - 8x^2 + 1, \end{aligned}$$

łatwo zgadnąć, że współczynnik przy najwyższej potędze x wielomianu T_n (dla $n \geq 1$) wynosi 2^{n-1} oraz, że

wszystkie współczynniki tych wielomianów, gdybyśmy je zapisali w bazie naturalnej, są całkowite. Te własności można udowodnić stosując proste rozumowanie indukcyjne. Stosowanie bazy naturalnej do zapisu wielomianów Czebyszewa w pamięci komputera nie ma sensu — po pierwsze po co, a po drugie, dla stosunkowo niewielkich n doszłoby do nadmiaru.

2. Funkcje T_n są dla n parzystych funkcjami parzystymi (czyli takimi, że $\forall x \in \mathbb{R} T_n(-x) = T_n(x)$), a dla n nieparzystych nieparzystymi (spełniającymi warunek $\forall x \in \mathbb{R} T_n(-x) = -T_n(x)$). Zatem dla n parzystego wykres T_n jest symetryczny względem osi OY , a nieparzystego symetryczny względem punktu $(0, 0)$.
3. Dla $x \in [-1, 1]$ wielomiany T_n można przedstawić (zupełnie nieintuicyjnym) wzorem

$$T_n(x) = \cos(n \arccos x).$$

4. Istnieje też inny nierekurencyjny wzór opisujący T_n

$$T_n(x) = \frac{(x + \sqrt{x^2 - 1})^n + (x - \sqrt{x^2 - 1})^n}{2}.$$

Jego poprawność można udowodnić przez indukcję. Ci, którzy znają teorię równań rekurencyjnych mogą go wprowadzić bezpośrednio. Zauważmy bowiem, że podstawiając do definicji wielomianów Czebyszewa $a_n = T_n(x)$ otrzymujemy

$$\begin{aligned} a_0 &= 1, \\ a_1 &= x, \\ a_k &= 2xa_{k-1} - a_{k-2}, \quad \text{dla } k = 2, 3, \dots \end{aligned}$$

Jest to równanie jednorodne o stałych współczynnikach (jeśli potraktujemy x jako parametr). Znane są metody jego rozwiązania.

5. Wielomian T_n posiada dokładnie n różnych pierwiastków rzeczywistych, wszystkie leżą w przedziale $(-1, 1)$. Są nimi liczby

$$\cos \frac{(2k+1)\pi}{2n} \quad \text{dla } k = 0, 1, \dots, n-1.$$

Rozmieszczone są one symetrycznie względem zera i im bliżej zera, tym rzadziej.

6. Wielomian posiada dokładnie $n-1$ ekstremów lokalnych: po jednym pomiędzy sąsiednimi miejscami zerowymi, są nimi liczby

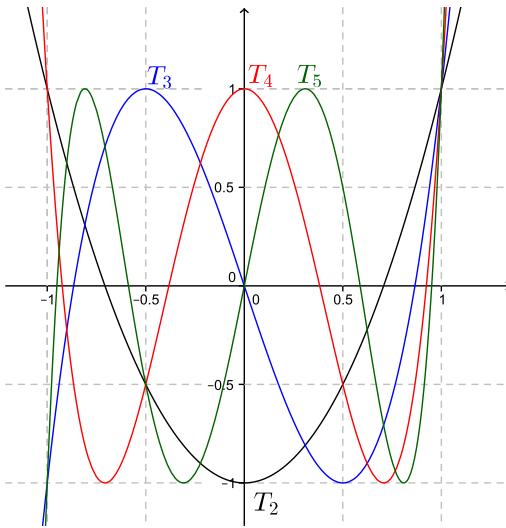
$$\cos \frac{k\pi}{n} \quad \text{dla } k = 1, \dots, n-1.$$

W tych ekstremach funkcja przyjmuje naprzemiennie wartości 1 i -1

$$T_n\left(\cos \frac{k\pi}{n}\right) = (-1)^k \quad \text{dla } k = 1, \dots, n-1.$$

7. $T_n(-1) = (-1)^n$, $T_n(1) = 1$.

8. Wykresy paru wielomianów Czebyszewa zamieszczono poniżej. Zauważmy, że im większe n , tym T_n częściej oscyluje. Proszę też zwrócić uwagę na wspomniane wcześniej symetrie.



3.3 Węzły równoodległe i Czebyszewa

Bazy Newtona i Lagrange'a w Π_n wymagają ustalenia węzłów — dla bazy Newtona potrzeba n zupełnie dowolnych, a Lagrange'a $n + 1$ różnych punktów. Wśród wielu możliwych wyborów dwa sposoby są szczególne.

Definicja 1. Węzłami równoległymi w przedziale $[a, b]$ nazywamy liczby $x_0 < x_1 < x_2 \dots < x_n$ takie, że $x_0 = a$, $x_n = b$, a pozostałe dzielą przedział $[a, b]$ na n równych części.

Poniższy rysunek przedstawia 10 węzłów równoodległych w przedziale $[a, b]$. Zauważmy, że wtedy $n = 9$.

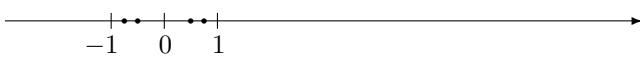


Definicja 2. Węzłami Czebyszewa w przedziale $[a, b]$ nazywamy liczby $x_0 < x_1 < x_2 \dots < x_n$, które są przeskalowanymi pierwiastkami wielomianu T_{n+1} do przedziału $[a, b]$, czyli takimi, które są wynikiem złożenia operacji

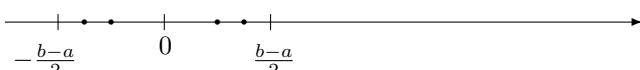
1. jednokładności względem 0 z czynnikiem $\frac{b-a}{2}$,
2. przesunięcia o wektor (jednowymiarowy) $\frac{a+b}{2}$.

Poniżej przedstawiam jak wyznaczyć 4 węzły Czebyszewa w przedziale $[a, b]$.

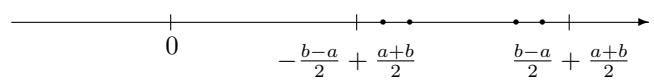
Pierwiastki T_4 :



Węzły po wykonaniu jednokładności:



Węzły po wykonaniu przesunięcia:



Fakt 1. Węzły Czebyszewa są pierwiastkami $(n+1)$ -go przeskalowanego wielomianu Czebyszewa

$$\tilde{T}_{n+1}(x) = T_{n+1} \left(\frac{x - \frac{a+b}{2}}{\frac{b-a}{2}} \right).$$

3.4 Norma supremum w przestrzeni funkcji ciągłych na przedziale domkniętym

Rozważmy zbiór wszystkich funkcji ciągłych o dziedzinie, która jest przedziałem domkniętym $[a, b]$. Zbiór ten wraz ze zwyczajowymi działaniami dodawania funkcji i mnożenia funkcji przez liczbę tworzy przestrzeń liniową nad ciałem \mathbb{R} . Będziemy oznaczać tą przestrzeń przez $C[a, b]$.

W przestrzeni tej można zdefiniować normę następująco (i udowodnić, że rzeczywiście jest to norma).

Definicja 3. Normą supremum w przestrzeni $C[a, b]$ nazywamy funkcję

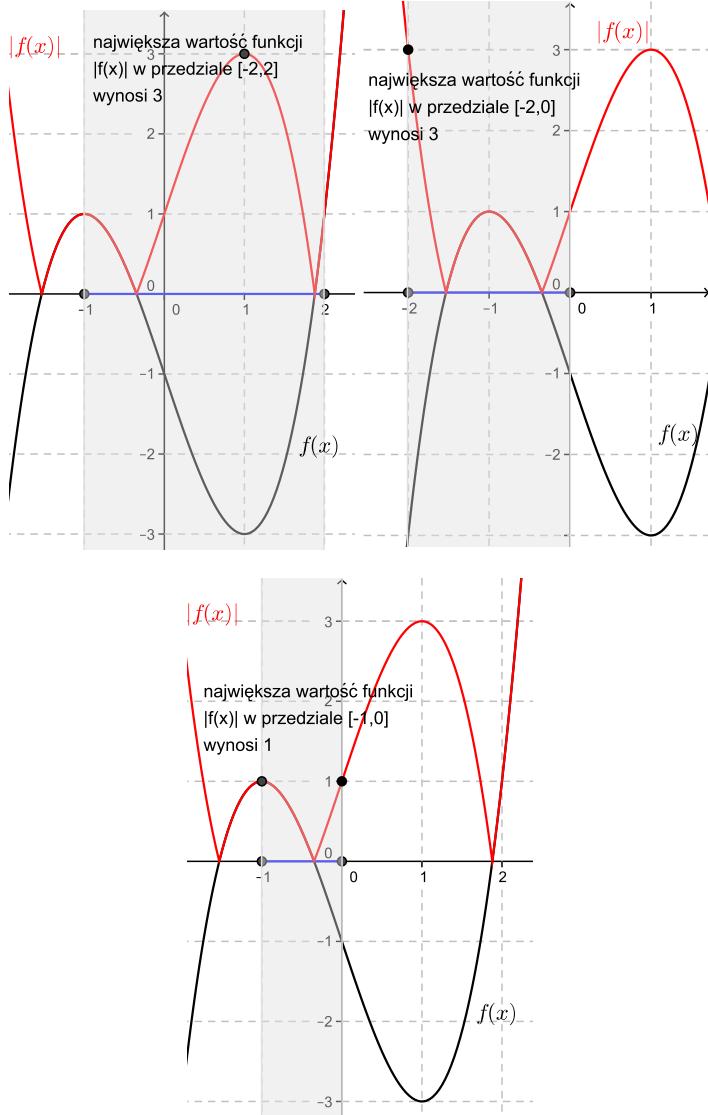
$$\|\cdot\|_{[a,b]} : C[a, b] \rightarrow \mathbb{R},$$

$$\|f\|_{[a,b]} = \sup_{x \in [a, b]} |f(x)|.$$

Norma ta przyporządkowuje każdej funkcji f jej największą wartość modułu na przedziale $[a, b]$. Wartość normy dla tej samej funkcji może zmieniać się, gdy zmieniamy przedział.

Zauważmy, że każdy wielomian jest funkcją ciągłą, zatem dla dowolnego przedziału $[a, b]$ należy on do przestrzeni $C[a, b]$. Można więc dla niego obliczać normę supremum.

Przykład Rozpatrzmy funkcję $f(x) = x^3 - 3x - 1$. Na rysunkach poniżej przedstawiono wykres tej funkcji oraz jej modułu. Dla różnych przedziałów $[a, b]$ zaznaczono największe wartości funkcji $|f(x)|$.



Z rysunków możemy odczytać, że

$$\begin{aligned}\|f\|_{[-2,2]} &= 3, \\ \|f\|_{[-2,0]} &= 3, \\ \|f\|_{[-1,0]} &= 1.\end{aligned}$$

3.5 Miary skośności baz

Każda z przedstawionych baz wielomianowych ma inne właściwości numeryczne. Jeśli będziemy badać uwarunkowanie zadania obliczania wartości wielomianu we wszystkich punktach z pewnego przedziału $[a, b]$, to okaże się, że dla różnych baz jest ono bardzo odmienne. W tym zadaniu danymi są współczynniki wielomianu, przedział $[a, b]$ jest ustalony.

Aby zbadać uwarunkowanie tego zadania postąpimy nieco inaczej, niż było to omówione na poprzednim wykładzie. Będziemy bowiem analizować wielkość

$$\frac{\|w - \tilde{w}\|_{[a,b]}}{\|w\|_{[a,b]}},$$

gdzie $w \in \Pi_n$ jest wielomianem zapisanym w bazie $\{u_i\}_{i=0}^n$, czyli

$$w(x) = \sum_{i=0}^n a_i u_i(x),$$

\tilde{w} jest wielomianem o nieco zaburzonych współczynnikach

$$\tilde{w}(x) = \sum_{i=0}^n (1 + \varepsilon_i) a_i u_i(x), \quad \text{dla } |\varepsilon_i| \leq \nu.$$

Ponieważ

$$\begin{aligned}|\tilde{w}(x) - w(x)| &= \left| \sum_{i=0}^n ((1 + \varepsilon_i) a_i - a_i) u_i(x) \right| \\ &= \left| \sum_{i=0}^n \varepsilon_i a_i u_i(x) \right| \\ &\leq \sum_{i=0}^n |\varepsilon_i| \cdot |a_i| \cdot |u_i(x)| \\ &\leq \nu \sum_{i=0}^n |a_i| \cdot |u_i(x)|,\end{aligned}$$

to

$$\frac{\|w - \tilde{w}\|_{[a,b]}}{\|w\|_{[a,b]}} \leq \nu \frac{\|\sum_{i=0}^n |a_i| \cdot |u_i(x)|\|_{[a,b]}}{\|\sum_{i=0}^n a_i u_i(x)\|_{[a,b]}}.$$

Ułamek

$$\frac{\|\sum_{i=0}^n |a_i| \cdot |u_i(x)|\|_{[a,b]}}{\|\sum_{i=0}^n a_i u_i(x)\|_{[a,b]}}$$

przyjmuje się za wskaźnik uwarunkowania zadania obliczania wartości wielomianu w przedziale $[a, b]$ zapisanego w bazie $\{u_i\}_{i=0}^n$. Jeśli chcemy porównać dwie bazy, to możemy rozważyć przy ustalonej bazie wszystkie wielomiany i wybrać najlepszy wskaźnik uwarunkowania. Ta wielkość nazywa się *miarą skośności bazy* i wyraża się wzorem

$$\sup \left\{ \frac{\|\sum_{i=0}^n |a_i| \cdot |u_i(x)|\|_{[a,b]}}{\|\sum_{i=0}^n a_i u_i(x)\|_{[a,b]}} : a_i \in \mathbb{R}, i = 0, 1, \dots, n \right\}.$$

Będę ją oznaczać przez $C(\{u_i\}, [a, b], n)$. Poniżej podaję wartości miary skośności dla przykładowych baz przestrzeni Π_n .

baza $\{u_i\}$	przedział $[a, b]$	miara skośności $C(\{u_i\}, [a, b], n)$
potęgowa („centralna”)	$[-a, a]$ ($a > 0$)	$D_n(1 + \sqrt{2})^n$
potęgowa („lokalna”)	$[0, a]$ ($a > 0$)	$E_n((1 + \sqrt{2})^2)^n$
potęgowa	$[a, 1]$ ($a \geq 0$, $a < 1$)	$F_n \left(\frac{(\sqrt{1+a}+\sqrt{2})^2}{1-a} \right)^n$
Czebyszewa	$[-1, 1]$	$\leq \sqrt{2n+1}$
Lagrange'a wzły równoodległe wzły Czebyszewa	$[a, b]$ ($a < b$)	$\sim c2^n$ $\sim c_1 + c_2 \ln(n)$

Stale c, c_1, c_2 są niewielkie. Liczby D_n, E_n i F_n spełniają nierówności

$$0.5 \leq D_n \leq 1.1,$$

$$\begin{aligned} 0.5 &\leq E_n \leq 1.02, \\ 0.5 &\leq F_n \leq 1.02 \end{aligned}$$

oraz $1 + \sqrt{2} \approx 2.4142$, a $(1 + \sqrt{2})^2 \approx 5.8284$. Skośność bazy potęgowej nie jest zadowalająca dla nawet niezbyt dużych n — miara skośności rośnie tu w tempie wykładniczym. Gdybyśmy chcieli spełnić warunek $C(\{x^i\}, [a, b], n) \leq 100$, to dla baz „centralnych” możemy posługiwać się tylko wielomianami stopnia co najwyżej 6 ($n \leq 6$), a „lokalnych” jedynie stopnia nie przekraczającego 3 ($n \leq 3$). Stosowanie baz potęgowych ze względu na możliwość sporej wrażliwości na zaburzenia współczynników powinno być ograniczone jedynie dla wielomianów niskich stopni. Należy jednak pamiętać, że miara skośności bada przypadek najgorszego wielomianu, więc nie zawsze musi być tak źle.

Baza Czebyszewa jest wyjątkowo dobra. Jeśli chcemy, by $C(\{T_i\}, [a, b], n) \leq 100$, to mamy to zagwarantowane dla wszystkich $n \leq 5000$. Posługując się bazą Czebyszewa na dowolnym przedziale $[a, b]$ należy przeskalać wielomiany Czebyszewa — posłużyć się funkcjami

$$\tilde{T}_{n+1}(x) = T_{n+1}\left(\frac{x - \frac{a+b}{2}}{\frac{b-a}{2}}\right)$$

i możliwe jest uzyskanie podobnej skośności jak w przypadku przedziału $[-1, 1]$.

Baza Lagrange'a może być bardzo podatna na zaburzenia współczynników — jeśli wybierzemy węzły równoodległe. Ale dla węzłów Czebyszewa sytuacja się zmienia ogromnie — tu miara skośności jest jedna z najlepszych, jaki dotąd zostały wyliczone.

Skośność bazy Newtona zależy od uporządkowania węzłów. Jeśli są w porządku rosnącym lub malejącym, to miara skośności jest duża i zawsze gorsza niż dla bazy Lagrange'a. Dużo lepiej się dzieje, gdy wybiera się specjalne węzły w odpowiednim porządku (np. węzły Czebyszewa w pewnym, na pozór chaotycznym ustaleniu).

4 Algorytmy obliczania wartości wielomianu

Niech $w \in \Pi_n$. Rozpatrzmy zadanie wyznaczenia $w(\alpha)$ na podstawie danych współczynników wielomianu w w pewnej bazie oraz punktu $\alpha \in \mathbb{R}$.

4.1 Algorytm Hornera dla bazy naturalnej

Dane: $a_0, a_1, \dots, a_n, \alpha$
 Wynik: $w(\alpha)$

Wielomian w ma postać

$$w(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1} + a_n x^n.$$

Korzystając bezpośrednio z powyższego wzoru można zapisać algorytm obliczający $w(\alpha)$. Jednak jego koszt jest wyższy niż algorytmu poniżej.

Zauważmy, że $w(\alpha)$ możemy zapisać następująco

$$\begin{aligned} w(\alpha) &= a_0 + \alpha(a_1 + a_2 \alpha + \dots + a_{n-1} \alpha^{n-2} + a_n \alpha^{n-1}) \\ &= a_0 + \alpha(a_1 + \alpha(a_2 + \dots + a_{n-1} \alpha^{n-3} + a_n \alpha^{n-2})) \\ &\quad \vdots \\ &= a_0 + \alpha(a_1 + \alpha(\underbrace{a_2 + \dots + \alpha(\underbrace{a_{n-1} + \alpha(a_n)}_{w_{n-1}})}_{w_2} \dots)). \end{aligned}$$

(W tym miejscu warto przerwać lekturę i spróbować samodzielnie zapisać algorytm.)

Wprowadźmy oznaczenia

$$\begin{aligned} w_n &= a_n, \\ w_{n-1} &= a_{n-1} + \alpha a_n = a_{n-1} + \alpha w_n, \\ w_{n-2} &= a_{n-2} + \alpha(a_{n-1} + \alpha(a_n)) = a_{n-2} + \alpha w_{n-1}, \\ &\quad \vdots \\ w_1 &= a_1 + \alpha w_2, \\ w_0 &= a_0 + \alpha w_1. \end{aligned}$$

Wielkość w_0 jest równa $w(\alpha)$.

Powyzsze zależności można zapisać zwięźlej

$$\begin{aligned} w_n &= a_n, \\ w_i &= a_i + \alpha w_{i+1} \text{ dla } i = n-1, n-2, \dots, 0 \end{aligned}$$

i prowadzą one do algorytmu nazywanego *schematem Hornera*.

```
w[n]=a[n];
for(i=n-1;i>=0;i--)
    w[i]=a[i]+alfa*w[i+1];
```

W tablicy a zapisane są współczynniki wielomianu w w bazie naturalnej, zmienna alfa zawiera punkt α . Wielkości w_i przechowywane są w tablicy w , element $w[0]$ zawiera wartość $w(\alpha)$.

Nietrudno policzyć, że koszt czasowy wynosi n dodawań i n mnożeń. W dalszym ciągu wykładu będę używała oznaczenia ops dla zapisania kosztu działania dodawania lub odejmowania, opm dla mnożenia lub dzielenia i opp dla pierwiastkowania. Zatem koszt algorytmu Hornera to $n \text{ ops} + n \text{ opm}$.

Zauważmy, że zamiast tablicy w wystarczy użyć pojedynczej zmiennej

```
w=a[n];
for(i=n-1;i>=0;i--)
    w=a[i]+alfa*w;
```

Niekiedy użycie tablicy w jest jednak wskazane, bo liczby w_i mają swoją interpretację. Są one współczynnikami w bazie naturalnej wielomianu powstałego przez podzielenie w przez jednomian $x - \alpha$, co można zapisać

$$w(x) = w_0 + (x - \alpha)[w_1 + w_2 x + \dots + w_n x^{n-1}].$$

Udowodniono, że jest to najtańszy algorytm służący do wyznaczania $w(\alpha)$ dla wielomianu danego w bazie naturalnej.

Na koniec omówię numeryczne własności tego algorytmu.

Twierdzenie 1. Niech $(2n+1)\nu \leq 0.1$. Algorytm Hornera jest numerycznie poprawny z największą stałą kumulacji równą w przybliżeniu $2n+1$. Dokładniej jeśli oznaczymy zadanie przez

$$\begin{aligned}\phi : \mathbb{R}^{n+2} &\rightarrow \mathbb{R} \\ \phi(\alpha, a_0, a_1, \dots, a_n) &= w(\alpha),\end{aligned}$$

a przez

$$\mathcal{A}(\alpha, a_0, a_1, \dots, a_n)$$

wynik zadania wyznaczony algorytmem Hornera, to zachodzi

$$\text{fl}(\mathcal{A}(\alpha, a_0, \dots, a_n)) = \phi(\alpha, a_0(1+\sigma_0), \dots, a_n(1+\sigma_n)),$$

gdzie $|\sigma_i| \leq K_i\nu$ i $K_i \approx 2i+1$.

Stałe kumulacji można uznać za niewielkie, bo zwykle n nie jest duże. Algorytm jest więc bardzo dobrej jakości.

4.2 Uogólniony algorytm Hornera

Dane: $b_0, b_1, \dots, b_n, \alpha$
 x_1, x_2, \dots, x_n
Wynik: $w(\alpha)$

Obecnie wielomian w zapisany jest w bazie Newtona

$$w(x) = b_0 + b_1(x-x_1) + b_2(x-x_1)(x-x_2) + \dots + b_n \prod_{i=1}^n (x-x_i).$$

Postępując podobnie jak dla bazy naturalnej zapisujemy $w(\alpha)$ następująco

$$b_0 + (\alpha - x_1)[b_1 + (\alpha - x_2)[b_2 + \dots + (\alpha - x_{n-1})[b_{n-1} + (\alpha - x_n)b_n]]]$$

Analogicznie też zapisujemy algorytm, oznaczając uprzednio nawiasy kwadratowe przez w_i i wyprowadzając zależność między w_i a w_{i+1} .

```
w=b[n];  
for(i=n-1;i>=0;i--)  
    w=b[i]+(alfa-x[i])*w;
```

Zwróćmy uwagę, że $x[i]$ przechowuje węzeł x_{i+1} , bo tablice w C są numerowane od 0. Koszt wynosi $2n$ ops + n opm. Algorytm jest numerycznie poprawny z największą stałą kumulacji równą w przybliżeniu $3n+1$.

4.3 Algorytmy dla bazy Czebyszewa

Dane: $c_0, c_1, \dots, c_n, \alpha$
Wynik: $w(\alpha)$

Będziemy obecnie obliczać $w(\alpha)$, gdzie w jest zapisany w bazie Czebyszewa, zatem

$$w(\alpha) = c_0T_0(\alpha) + c_1T_1(\alpha) + \dots + c_{n-1}T_{n-1}(\alpha) + c_nT_n(\alpha).$$

Algorytm I Korzystając z zależności rekurencyjnej otrzymujemy, że

$$\begin{aligned}T_0(\alpha) &= 1, \\ T_1(\alpha) &= \alpha, \\ T_k(\alpha) &= 2\alpha T_{k-1}(\alpha) - T_{k-2}(\alpha), \quad \text{dla } k = 2, 3, \dots\end{aligned}$$

Algorytm na podstawie powyższych zależności kolejno oblicza $T_k(\alpha)$ i sumy częściowe

$$w_k(\alpha) = c_0T_0(\alpha) + c_1T_1(\alpha) + \dots + c_kT_k(\alpha)$$

dla $k = 0, 1, 2, \dots, n$. Do zapamiętania sum częściowych wystarczy jedna zmienna w . Naliczanie $T_k(\alpha)$ wymaga trzech zmiennych: aby obliczyć $T_k(\alpha)$ trzeba mieć zapamiętane $T_{k-1}(\alpha)$ i $T_{k-2}(\alpha)$. Niech będą to T , $T1$ i $T2$. Cały algorytm ma postać

```
if (n==0) return c[0];  
T2=1.0;  
T1=alfa;  
dwa_alfa=2*alfa;  
w=c[0]+c[1]*alfa;  
for(i=2;i<=n;i++){  
    T=dwa_alfa*T1-T2;  
    w=w+c[i]*T;  
    T2=T1;  
    T1=T;  
}  
return w;
```

Koszt wynosi $(2n-1)$ ops + $2n$ opm.

Jakość algorytmu opisuje poniższe twierdzenie.

Twierdzenie 2. Niech $\mathcal{A}_1(\alpha, w)$ oznacza wynik obliczony powyższym algorytmem dla wielomianu w i punktu α . Wtedy

$$|\text{fl}(\mathcal{A}_1(\alpha, w)) - w(\alpha)| \leq \nu C n^2 \|w\|_{[-1,1]}$$

dla pewnej (niewielkiej) stałej C .

Jakość jest zatem dobra, ale w szczególnym sensie — błąd bezwzględny nie przekracza $\nu C n^2 \|w\|_{[-1,1]}$. O ile $\|w\|_{[-1,1]}$ nie jest olbrzymie, to błąd jest niewielki. Nie można w przypadku obliczania wartości wielomianu używać błędu względnego

$$\frac{|\text{fl}(\mathcal{A}_1(\alpha, w)) - w(\alpha)|}{|w(\alpha)|},$$

jako uniwersalnego miernika, bo dla miejsc zerowych mianownik byłby zerem, a w ich okolicy błąd ten musi być duży. Niemniej, gdy $w(\alpha)$ jest rzędu $\|w\|_{[-1,1]}$, to mamy zagwarantowany dla takiego α niewielki błąd względny nawet, gdy norma $\|w\|_{[-1,1]}$ jest duża.

Algorytm II Okazuje się, że można skonstruować algorytm tańszy od poprzedniego. We wzorze

$$w(\alpha) = c_0T_0(\alpha) + \dots + c_{n-2}T_{n-2}(\alpha) + c_{n-1}T_{n-1}(\alpha) + c_nT_n(\alpha)$$

zastąpmy $T_n(\alpha)$ wyrażeniem $2\alpha T_{n-1}(\alpha) - T_{n-2}(\alpha)$:

$$\begin{aligned} w(\alpha) &= c_0 T_0(\alpha) + c_1 T_1(\alpha) + \dots + c_{n-1} T_{n-1}(\alpha) \\ &\quad + c_n [2\alpha T_{n-1}(\alpha) - T_{n-2}(\alpha)] \end{aligned}$$

i pogrupujmy wyrazy względem T_k . Otrzymamy

$$\begin{aligned} w(\alpha) &= c_0 T_0(\alpha) + \dots + c_{n-3} T_{n-3}(\alpha) \\ &\quad + \widetilde{c_{n-2}} T_{n-2}(\alpha) + \widetilde{\widetilde{c_{n-1}}} T_{n-1}(\alpha), \end{aligned}$$

gdzie $\widetilde{c_{n-1}} = c_{n-1} + 2\alpha c_n$ i $\widetilde{c_{n-2}} = c_{n-2} - c_n$.

Uzyskana suma jest tej samej postaci, co wyjściowa, ale ma o jeden mniej składników. Postępując analogicznie możemy kolejno skracać sumę, aż będzie się składała z dwóch składników zawierających $T_0(\alpha)$ i $T_1(\alpha)$ (mnożone przez pewne współczynniki).

Schemat działania algorytmu można zatem przedstawić następująco

k	T_0	T_1	T_2	\dots	T_{n-3}	T_{n-2}	T_{n-1}	T_n
$n-2$	c_0	c_1	c_2	\dots	c_{n-3}	c_{n-2}	$\widetilde{c_{n-1}}$	c_n
$n-3$	c_0	c_1	c_2	\dots	c_{n-3}	$\widetilde{c_{n-2}}$	$\widetilde{\widetilde{c_{n-1}}}$	
\vdots	\vdots	\vdots	\vdots	\vdots				
1		\tilde{c}_0	\tilde{c}_1	\tilde{c}_2				
0		\tilde{c}_0	\tilde{c}_1					

Nowe współczynniki są dane zależnościami

$$\begin{aligned} \widetilde{c_{n-1}} &= c_{n-1} + 2\alpha c_n, \\ \widetilde{c_{n-2}} &= c_{n-2} - c_n, \\ \widetilde{c_{n-2}} &= \widetilde{c_{n-2}} + 2\alpha \widetilde{c_{n-1}}, \\ \widetilde{c_{n-3}} &= c_{n-3} - \widetilde{c_{n-1}}, \\ \widetilde{c_{n-3}} &= \widetilde{c_{n-3}} + 2\alpha \widetilde{c_{n-2}}, \\ \widetilde{c_{n-4}} &= c_{n-4} - \widetilde{c_{n-2}}, \\ &\vdots \\ \widetilde{\tilde{c}_2} &= \tilde{c}_2 + 2\alpha \widetilde{\tilde{c}_3}, \\ \tilde{c}_1 &= c_1 - \widetilde{\tilde{c}_3}, \\ \widetilde{\tilde{c}_1} &= \widetilde{\tilde{c}_1} + 2\alpha \widetilde{\tilde{c}_2}, \\ \tilde{c}_0 &= c_0 - \widetilde{\tilde{c}_2}. \end{aligned}$$

Wartość $w(\alpha)$ wynosi zatem $w(\alpha) = \tilde{c}_0 + \widetilde{\tilde{c}_1} \alpha$. Możemy obecnie zapisać algorytm. Współczynniki wielomianu w bazie Czebyszewa umieszczone są w tablicy c . Zmienne $c_{_ff}$, $c_{_f}$ zawierają współczynniki stojące przy dwóch wielomianach Czebyszewa które w danym kroku mają najwyższe stopnie. Zmienna w zawiera obliczoną wartość $w(\alpha)$.

```
dwa_alfa=2*alfa;
c_ff=c[n];
c_f=c[n-1];
for(k=n-2;k>=0;k--) {
    pom=c[k]-c_ff;
    c_ff=c_f+dwa_alfa*c_ff;
    c_f=pom;
}
w=c_f+c_ff*alfa;
```

```
c_ff=c_f+dwa_alfa*c_ff;
c_f=pom;
}
w=c_f+c_ff*alfa;
```

Koszt wynosi $(2n-1)$ ops + $(n+1)$ opm. Jakość jest podobna do jakości algorytmu poprzedniego — zachodzi analogiczne twierdzenie z nieco inną stałą.

Przedstawiony algorytm nosi miano algorytmu Clenshaw'a. W literaturze spotyka się też inne sposoby jego zapisu. Ponadto ma on uogólnienie dla baz złożonych z tzw. wielomianów ortogonalnych, co na razie pominę.

4.4 Algorytm dla bazy Lagrange'a

Dane:	$d_0, d_1, \dots, d_n, \alpha$
	x_0, x_1, \dots, x_n
Wynik:	$w(\alpha)$

Obecnie wielomian w zapisany jest w bazie Lagrange'a,

$$w(\alpha) = \sum_{i=0}^n d_i l_i(\alpha).$$

Zajmijmy się najpierw przypadkiem, gdy $\alpha = x_k$ dla pewnego $k = 0, 1, \dots, n$. Ponieważ

$$l_i(x_k) = \begin{cases} 1 & \text{gdy } i = k, \\ 0 & \text{w p.p.,} \end{cases}$$

to

$$w(x_k) = \sum_{i=0}^n d_i l_i(x_k) = d_k,$$

zatem dla $\alpha = x_k$ wynik od razu jest znany i nie trzeba go obliczać.

Niech teraz α nie będzie węzłem. Wtedy

$$\begin{aligned} w(\alpha) &= \sum_{i=0}^n d_i \prod_{\substack{j=0 \\ j \neq i}}^n \frac{\alpha - x_j}{x_i - x_j} \\ &= \sum_{i=0}^n d_i \frac{\prod_{\substack{j=0 \\ j \neq i}}^n (\alpha - x_j)}{\prod_{\substack{j=0 \\ j \neq i}}^n (x_i - x_j)} \\ &= \sum_{i=0}^n d_i \frac{\prod_{\substack{j=0 \\ j \neq i}}^n (\alpha - x_j)}{(\alpha - x_i) \prod_{\substack{j=0 \\ j \neq i}}^n (x_i - x_j)}. \end{aligned}$$

W ostatniej równości wykorzystujemy przyjęte założenie o α . Przed znakiem sumy możemy obecnie wyciągnąć czynnik $\pi(\alpha) = \prod_{j=0}^n (\alpha - x_j)$. Wtedy

$$w(\alpha) = \pi(\alpha) \cdot \sum_{i=0}^n \frac{d_i}{\prod_{\substack{j=0 \\ j \neq i}}^n (x_i - x_j)} \cdot \frac{1}{\alpha - x_i}.$$

Jeśli wprowadzimy oznaczenie

$$z_i = \frac{d_i}{\prod_{\substack{j=0 \\ j \neq i}}^n (x_i - x_j)},$$

to

$$w(\alpha) = \pi(\alpha) \cdot \sum_{i=0}^n \frac{z_i}{\alpha - x_i} = \pi(\alpha) \cdot \sigma(\alpha),$$

gdzie $\sigma(\alpha) = \sum_{i=0}^n z_i / (\alpha - x_i)$.

Powyższa postać sugeruje algorytm. Wielkości z_i , które nie zależą od punktu α , należy policzyć najpierw i umieścić w tablicy \mathbf{z} .

```
for(i=0;i<=n;i++)
{
    z[i]=d[i];
    for(j=0;j<=n;j++)
        if(j!=i)
            z[i]/=x[i]-x[j];
}
```

Następnie w jednej pętli obliczamy równocześnie wielkości $\pi(\alpha)$ i $\sigma(\alpha)$. Tablica \mathbf{x} zawiera węzły, zmienna **alfa** wartość α . Zmienne **pi** oraz **sigma** służą do obliczenia $\pi(\alpha)$ oraz $\sigma(\alpha)$. W zmiennej **w** zapisany jest wynik.

```
sigma=0.0;
pi=1.0;
for(i=0;i<=n;i++)
{
    h=alfa-x[i];
    pi=pi*h;
    sigma=sigma+z[i]/h;
}
w=pi*sigma;
```

Należy dodać w odpowiednim miejscu warunki, które pozwolą zwrócić wynik w przypadku, gdy α jest węzłem. Mimo, że algorytm wygląda elegancko jednak programując go należy uważać, by nie wystąpił ani nadmiar, ani niedomiar. Występują tu bowiem iloczyny i ilorazy wielu czynników. Problem ten można rozwiązać, gdy zastosujemy specjalną reprezentację wyniku takiego iloczynu (bądź ilorazu) postaci $d \cdot 2^e$, gdzie $d \in [\frac{1}{2}, 1]$, a e jest liczbą całkowitą⁴.

Policzmy koszt przedstawionej tu wersji.⁵ Obliczenie wartości wszystkich z_i wymaga $(n^2 + n)$ ops + $(n^2 + n)$ opm. Wyznaczenie $\pi(\alpha)$ oraz $\sigma(\alpha)$ to $(2n + 2)$ ops + $(2n + 3)$ opm. Drogo, ale zauważmy, że najdroższa część, czyli obliczenie z_i nie zależy od α , wyznaczając wartości wielomianu w różnych punktach ten koszt ponosimy tylko raz. Pozostała część jest tania.

⁴Przymajemy założenie, że posługujemy się arytmetyką binarną. W przypadku dziesiętnej należałoby użyć postaci $d \cdot 10^e$, gdzie $d \in [0.1, 1]$.

⁵Wersja unikająca nadmiaru i niedomiaru jest nieco droższa. Mogą tam bowiem występować wielokrotnie działania mnożenia i dzielenia przez 2. Są one jednak tańsze niż mnożenie czy dzielenie przez liczby, które nie są potęgami dwójki — polegają jedynie na modyfikacji wykładnika i są wykonywane dokładnie.

Ponadto, gdybyśmy operowali kilkoma wielomianami optymi na tych samych węzłach, to warto zmodyfikować algorytm, tak by osobno policzyć mianowniki liczb z_i wspólne dla wszystkich wielomianów.

Jakość algorytmu jest bardzo dobra.

Twierdzenie 3. Niech $3n\nu \leq 0.1$. Przedstawiony algorytm \mathcal{A} jest numerycznie poprawny z największą stałą kumulacji równą w przybliżeniu $3n$. Dokładniej jeśli oznaczymy zadanie przez

$$\phi : \mathbb{R}^{2n+3} \rightarrow \mathbb{R}$$

$$\phi(\alpha, x_0, x_1, \dots, x_n, d_0, d_1, \dots, d_n) = w(\alpha),$$

a przez

$$\mathcal{A}(\alpha, x_0, x_1, \dots, x_n, d_0, d_1, \dots, d_n)$$

wynik zadania wyznaczony tym algorytmem, to zachodzi

$$\text{fl}(\mathcal{A}(\alpha, x_0, x_1, \dots, x_n, d_0, d_1, \dots, d_n))$$

$$= (1 + \sigma)\phi(\alpha, x_0, \dots, x_n, d_0(1 + \delta_0), \dots, d_n(1 + \delta_n)),$$

gdzie $|\delta_i| \leq K_i\nu$ i $K_i \approx 2n$ oraz $|\sigma| \leq K\nu$ i $K \approx 3n$.

Numeryczna poprawność nie oznacza tu, że na pewno użyjemy mały błąd względny wyniku. Wszystko zależy od skośności bazy. Przy bazach o dużej mierze skośności nawet niewielkie zaburzenie współczynników może znacznie zaburzyć wynik. Stąd bardzo istotny jest wybór węzłów — Czebyszewa są bardzo dobre, a równoodległe dopuszczalne tylko przy niewielkich n .

5 Algorytmy zamiany bazy

Przypomnijmy koszty posługiwania się omówionymi bazami (czyli obliczenia $w(\alpha)$ przy danych α , współczynnikach, węzłach).

Baza	Koszt	
	ops	opm
potegowa — alg. Hornera	n	n
Newtona — alg. Hornera	$2n$	n
Czebyszewa — alg. bezpośredni — alg. Clenshaw'a	$2n - 1$ $2n - 1$	$2n$ $n + 1$
Lagrange'a — rozkład $\pi(\alpha)\sigma(\alpha)$	niezależny od α : $n^2 + n \mid n^2 + n$ zależny od α : $2n + 2 \mid 2n + 3$	

Czasami (tzn. gdy wielokrotnie trzeba obliczać wartość wielomianu) sensowne wydaje się, by mając wielomian zapisany np. w bazie Lagrange'a, zamiast używać stosunkowo drogiego algorytmu, przedstawić wielomian w bazie naturalnej, czy Newtona i posłużyć się tańszym. W innych sytuacjach — np. gdy

przede wszystkim liczy się jakość obliczeń, a wielomian jest wysokiego stopnia — to ze względu na dużą skośność bazy naturalnej warto rozważyć przejście np. do bazy Czebyszewa. Podobne argumenty są przyczyną opracowania algorytmów zamiany bazy. Na wykładzie i ćwiczeniach zostaną omówione niektóre z nich.

5.1 Jeszcze raz o skośności bazy Newtona

W punkcie 3.5 wspomniałam, że skośność bazy Newtona zależy od uporządkowania węzłów. Jest ona duża, gdy np. węzły są ustawione rosnąco lub malejąco. Pamiętajmy jednak, że miara skośności zależy od n — w praktyce dla $n \leq 10$ każde uporządkowanie będzie dopuszczalne.

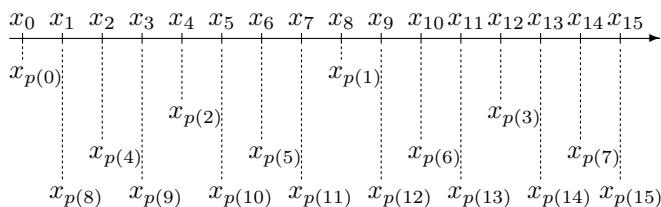
Gdy zachodzi potrzeba posługiwania się wielomianami wyższych stopni niż 10, to trzeba węzły odpowiednio uporządkować. Przedstawię Państwu dobre uporządkowanie przy założeniu, że n jest potęgą dwójki.

Dane więc mamy węzły $x_0 < x_1 < \dots < x_{n-1}$, gdzie $n = 2^k$ dla pewnego $k \geq 1$. Opiszę permutację

$$p : \{0, 1, \dots, n-1\} \rightarrow \{0, 1, \dots, n-1\},$$

dla której baza Newtona o węzłach $x_{p(0)}, x_{p(1)}, \dots, x_{p(n-1)}$ ma dobrą miarę skośności.

Idea jest taka, by kolejne przepermutowane węzły zapełniały duże luki. Dla $n = 16$ permutacja jest przedstawiona na poniższym rysunku.



Można ją opisać wzorami

$$\begin{aligned} p(0) &= 0, \\ p(i) &= p(i-1) + 8, \quad \text{dla } i = 1, \\ p(i) &= p(i-2) + 4, \quad \text{dla } i = 2, 3, \\ p(i) &= p(i-4) + 2, \quad \text{dla } i = 4, 5, 6, 7, \\ p(i) &= p(i-8) + 1, \quad \text{dla } i = 8, 9, \dots, 15. \end{aligned}$$

W ogólności dla $n = 2^k$ permutacja p wyraża się zależnościami

$$\begin{aligned} p(0) &= 0, \\ p(i) &= p(i - 2^j) + 2^{k-j-1} \quad \text{dla } j = 0, 1, \dots, k-1, \\ &\quad i = 2^j, 2^j + 1, \dots, 2^{j+1} - 1. \end{aligned}$$

Ciąg $p(0), p(1), \dots, p(n-1)$ nosi miano *ciągu van der Corput'a*. Jeśli n nie jest potęgą dwójki, to też istnieją dobre permutacje węzłów, tzn. takie, dla których miara skośności bazy Newtona jest niezbyt szybko rosnącą funkcją n — nie będę jednak o tym pisać.

Pamiętajmy, że w praktyce występują głównie wielomiany niskich stopni i uporządkowanie węzłów nie ma wtedy znaczenia.

5.2 Baza Lagrange'a → baza Newtona

Dane:	x_0, x_1, \dots, x_n — węzły,
	d_0, d_1, \dots, d_n — współczynniki w bazie Lagrange'a o węzłach x_0, x_1, \dots, x_n ,
Wyniki:	b_0, b_1, \dots, b_n — współczynniki w bazie Newtona o węzłach x_0, x_1, \dots, x_n

Algorytmy realizujące zadanie zamiany bazy Lagrange'a na Newtona zdają się najważniejsze z prezentowanych na tym wykładzie. Są używane przy interpolacji Lagrange'a, o której będzie mowa później.

Mamy zatem dane $n+1$ różnych węzłów x_0, x_1, \dots, x_n oraz współczynniki d_0, d_1, \dots, d_n wielomianu w w bazie Lagrange'a, więc

$$w(x) = \sum_{i=0}^n d_i l_i(x).$$

Ten sam wielomian chcemy przedstawić w bazie Newtona z węzłami x_0, x_1, \dots, x_{n-1} ,⁶ czyli opisać wzorem postaci

$$w(x) = b_0 + b_1(x-x_0) + b_2(x-x_0)(x-x_1) + \dots + b_n \prod_{i=0}^{n-1} (x-x_i).$$

Dla wygody zapisu oznaczmy k -ty element bazy Newtona zdefiniowanej dla węzłów x_0, x_1, \dots, x_{n-1} przez N_k , czyli

$$N_k(x) = \prod_{j=0}^{k-1} (x-x_j) \quad \text{dla } k = 0, 1, \dots, n.$$

Wtedy

$$w(x) = b_0 N_0(x) + b_1 N_1(x) + \dots + b_n N_n(x).$$

Zauważmy, ponadto że zachodzi własność

$$\begin{aligned} N_0(x) &= 1, \\ N_k(x) &= (x-x_{k-1}) N_{k-1}(x) \quad \text{dla } k = 1, 2, \dots, n. \end{aligned}$$

5.2.1 Algorytm pierwszy — dobrej jakości

Udowodnię najpierw twierdzenie pomocnicze, które pozwoli wyprowadzić algorytm.

Twierdzenie 4. Niech $l_i^{(k)}$ dla $k = 0, 1, \dots, n$ oraz $i = 0, 1, \dots, k$ oznaczają i -ty wielomian bazowy Lagrange'a dla węzłów x_0, x_1, \dots, x_k oraz niech

$$w_k(x) = \sum_{i=0}^k d_i l_i^{(k)}(x) \quad \text{dla } k = 0, 1, \dots, n$$

oraz pewnych współczynników d_0, d_1, \dots, d_n . Wtedy dla dowolnego $1 \leq k \leq n$ zachodzi

$$w_k(x) = w_{k-1}(x) + b_k N_k(x) \quad \text{gdzie } b_k = \sum_{i=0}^k \frac{d_i}{\prod_{j=0, j \neq i}^{k-1} (x_i - x_j)}.$$

⁶Proszę zwrócić uwagę, że tu jest o jeden mniej węzle niż w bazie Lagrange'a.

Dowód Ustalmy $1 \leq k \leq n$. Zauważmy, że dla $i = 0, 1, \dots, k-1$ mamy

$$\begin{aligned} l_i^{(k)}(x) - l_i^{(k-1)}(x) &= \prod_{\substack{j=0 \\ j \neq i}}^k \frac{x - x_j}{x_i - x_j} - \prod_{\substack{j=0 \\ j \neq i}}^{k-1} \frac{x - x_j}{x_i - x_j} \\ &= \left[\frac{x - x_k}{x_i - x_k} - 1 \right] \prod_{\substack{j=0 \\ j \neq i}}^{k-1} \frac{x - x_j}{x_i - x_j} \\ &= \frac{x - x_i}{x_i - x_k} \prod_{\substack{j=0 \\ j \neq i}}^{k-1} \frac{x - x_j}{x_i - x_j} \\ &= \frac{\prod_{j=0}^{k-1} (x - x_j)}{\prod_{\substack{j=0 \\ j \neq i}}^k (x_i - x_j)} = \frac{N_k(x)}{\prod_{\substack{j=0 \\ j \neq i}}^k (x_i - x_j)}. \end{aligned}$$

Ponadto

$$l_k^{(k)}(x) = \prod_{\substack{j=0 \\ j \neq k}}^k \frac{x - x_j}{x_k - x_j} = \frac{\prod_{j=0}^{k-1} (x - x_j)}{\prod_{j=0}^{k-1} (x_k - x_j)} = \frac{N_k(x)}{\prod_{j=0}^{k-1} (x_k - x_j)}.$$

Zatem

$$\begin{aligned} w_k(x) &= \sum_{i=0}^k d_i l_i^{(k)}(x) \\ &= \sum_{i=0}^{k-1} d_i \left(l_i^{(k-1)}(x) + \frac{N_k(x)}{\prod_{\substack{j=0 \\ j \neq i}}^k (x_i - x_j)} \right) + d_k l_k^{(k)}(x) \\ &= w_{k-1}(x) + \sum_{i=0}^{k-1} \frac{d_i N_k(x)}{\prod_{\substack{j=0 \\ j \neq i}}^k (x_i - x_j)} + \frac{d_k N_k(x)}{\prod_{j=0}^{k-1} (x_k - x_j)} \\ &= w_{k-1}(x) + \sum_{i=0}^k \frac{d_i N_k(x)}{\prod_{\substack{j=0 \\ j \neq i}}^k (x_i - x_j)} \\ &= w_{k-1}(x) + b_k N_k(x). \end{aligned}$$

Udowodnione twierdzenie jest podstawą budowy algorytmu. Stosując je bowiem wielokrotnie otrzymujemy

$$\begin{aligned} w(x) &= w_n(x) \\ &= w_{n-1}(x) + b_n N_n(x) \\ &= w_{n-2}(x) + b_{n-1} N_{n-1}(x) + b_n N_n(x) \\ &= w_{n-3}(x) + b_{n-2} N_{n-2}(x) + b_{n-1} N_{n-1}(x) + b_n N_n(x) \\ &\vdots \\ &= w_0(x) + b_1 N_1(x) + b_2 N_2(x) + \dots + b_n N_n(x) \\ &= b_0 N_0(x) + b_1 N_1(x) + b_2 N_2(x) + \dots + b_n N_n(x), \end{aligned}$$

$$b_k = \sum_{i=0}^k \frac{d_i}{\prod_{\substack{j=0 \\ j \neq i}}^k (x_i - x_j)} \quad \text{dla } k = 0, 1, \dots, n$$

są szukanymi liczbami. Będziemy je obliczać kolejno dla $k = 0, 1, 2, \dots, n$. Zauważmy, że $b_0 = d_0$. Jeśli wprowadzimy oznaczenie

$$z_i^{(k)} = \frac{d_i}{\prod_{\substack{j=0 \\ j \neq i}}^k (x_i - x_j)} \quad \text{dla } k = 0, 1, \dots, n \text{ oraz } i = 0, 1, \dots, k,$$

to otrzymamy

$$b_k = \sum_{i=0}^k z_i^{(k)} \quad \text{dla } k = 0, 1, \dots, n.$$

Zauważmy, że

$$\begin{aligned} z_0^{(0)} &= d_0, \\ z_0^{(1)} &= \frac{z_0^{(0)}}{x_0 - x_1}, z_1^{(1)} = \frac{d_1}{x_1 - x_0}, \\ z_0^{(2)} &= \frac{z_0^{(1)}}{x_0 - x_2}, z_1^{(2)} = \frac{z_1^{(1)}}{x_1 - x_2}, z_2^{(2)} = \frac{d_2}{(x_2 - x_0)(x_2 - x_1)}, \end{aligned}$$

w ogólności dla $k = 1, 2, \dots, n$ oraz $i = 0, 1, \dots, k-1$

$$\begin{aligned} z_i^{(k)} &= \frac{d_i}{\prod_{\substack{j=0 \\ j \neq i}}^{k-1} (x_i - x_j)} = \frac{z_i^{(k-1)}}{x_i - x_k}, \\ z_k^{(k)} &= \frac{d_k}{\prod_{j=0}^{k-1} (x_k - x_j)} = \frac{d_k}{\prod_{j=0}^{k-1} (-x_j - x_k)}. \end{aligned}$$

Powiedzmy, że mamy zdefiniowane tablice

<code>float x[n+1];</code>	tu są węzły x_0, x_1, \dots, x_n ,
<code>float d[n+1];</code>	tu są współczynniki d_0, d_1, \dots, d_n ,
<code>float b[n+1];</code>	tu będą wyniki b_0, b_1, \dots, b_n ,
<code>float z[n+1];</code>	tu będą liczby pomocnicze $z_i^{(k)}$.

Algorytm składa się z n kroków. W k -tym kroku obliczany jest wynik b_k . W tym celu najpierw wyznaczamy liczby $z_i^{(k)}$ dla $i = 0, 1, \dots, k$ i zapisujemy w tablicy z . Zastępują one

umieszczone tam wcześniej liczby $z_i^{(k-1)}$.

$$\begin{array}{l} z_0^{(0)} = d_0 \\ \hline \downarrow \cdot \frac{1}{x_0 - x_1} \\ z_0^{(1)} \quad z_1^{(1)} = \frac{d_1}{x_1 - x_0} \\ \hline \downarrow \cdot \frac{1}{x_0 - x_2} \quad \downarrow \cdot \frac{1}{x_1 - x_2} \\ z_0^{(2)} \quad z_1^{(2)} \quad z_2^{(2)} = \frac{d_2}{(x_2 - x_0)(x_2 - x_1)} \\ \hline \downarrow \cdot \frac{1}{x_0 - x_3} \quad \downarrow \cdot \frac{1}{x_1 - x_3} \quad \downarrow \cdot \frac{1}{x_2 - x_3} \\ z_0^{(3)} \quad z_1^{(3)} \quad z_2^{(3)} \quad z_3^{(3)} = \frac{d_3}{(x_3 - x_0)(x_3 - x_1)(x_3 - x_2)} \end{array}$$

Podczas obliczania $z_i^{(k)}$ dla $i = 0, 1, \dots, k$ od razu można wyliczyć $z_k^{(k)}$. Zmienna h pełni rolę pomocniczą — przechowuje kolejno liczby, przez które trzeba podzielić $z_i^{(k-1)}$ by otrzymać $z_i^{(k)}$.

```
b[0]=z[0]=d[0];
for(k=1;k<=n;k++){
    z[k]=d[k];           //obliczenie liczb z[i]
    for(i=0;i<k;i++){
        h=x[i]-x[k];
        z[i]=z[i]/h;
        z[k]=-z[k]/h;
    }
    b[k]=0.0;            //obliczenie liczby b[k]
    for(i=0;i<=k;i++)
        b[k]=b[k]+z[i];
}
```

Zauważmy, że dwie pętle wewnętrzne można zastąpić jedną. Ze względu na różne zakresy zmiennej i w instrukcjach `for`, jedno z działań trzeba wykonać poza pętlą.

```
b[0]=z[0]=d[0];
for(k=1;k<=n;k++){
    z[k]=d[k];
    b[k]=0.0;
    for(i=0;i<k;i++){
        h=x[i]-x[k];
        z[i]=z[i]/h;
        z[k]=-z[k]/h;
        b[k]=b[k]+z[i];
    }
    b[k]=b[k]+z[k];
}
```

Jeśli n jest duże, to warto przy obliczaniu elementów tablicy z zastosować specjalną ich reprezentację (w jednej zmiennej przechowywać rozwinięcie $r \in [\frac{1}{2}, 1)$, a w drugiej wykładnik $E \in \mathbb{Z}$, wartością liczby jest $r \cdot 2^E$.) Pozwoli to unikać nadmiaru i niedomiaru.

Obliczmy koszt algorytmu. Pętla wewnętrzna jest wykonywana k razy, jej treść kosztuje $2 \text{ ops} + 2 \text{ opm}$, zatem cały jej

koszt to $2k \text{ ops} + 2k \text{ opm}$. W k -tym wykonaniu pętli głównej mamy $(2k+1) \text{ ops} + 2k \text{ opm}$. Zatem całkowity koszt to

$$\sum_{k=1}^n (2k+1) \text{ ops} + \sum_{k=1}^n 2k \text{ opm} = n(n+2) \text{ ops} + n(n+1) \text{ opm}.$$

Algorytm można nieco przyspieszyć, jeśli wiadomo, że węzły są równoodległe. Poniższe twierdzenie dotyczy jakości algorytmu.

Twierdzenie 5. Niech

$$\phi : \mathbb{R}^{2n+2} \rightarrow \mathbb{R}^{n+1},$$

$$\phi(x_0, x_1, \dots, x_n, d_0, d_1, \dots, d_n) = (b_0, b_1, \dots, b_n),$$

gdzie x_i, d_i, b_i mają znaczenie opisane wyżej. Wtedy przedstawiony algorytm $\mathcal{A} : \mathbb{R}^{2n+2} \rightarrow \mathbb{R}^{n+1}$ rozwiązujący zadanie ϕ jest prawie numerycznie poprawny, tzn.

$$\text{fl}(b_k) = \sum_{i=0}^k \frac{d_i(1 + \delta_i^{(k)})}{\prod_{\substack{j=0 \\ j \neq i}}^k (x_i - x_j)} \quad \text{dla } k = 0, 1, \dots, n,$$

$$\text{gdzie } |\delta_i^{(k)}| \leq 3k\nu.$$

Sformułowanie „prawie numerycznie poprawny” oznacza tu, że algorytm jest numerycznie poprawny ze względu na każdy wynik z osobna, ale nie na wszystkie razem. Gdyby dla każdego $i = 0, 1, \dots, n$ zachodziło, że $\delta_i^{(0)} = \delta_i^{(1)} = \dots = \delta_i^{(i)}$, to algorytm byłby numerycznie poprawny, ale tak nie jest. Niemniej jakość jest dobra.

5.2.2 Algorytm różnic dzielonych

Czy można omawiane zadanie rozwiązać taniej? Okazuje się że tak, niemniej z algorytmem przedstawionym poniżej wiążą się pewne problemy związane z jakością, o czym będzie później.

Rozpatrzmy następującą tabelę.

x_0	w_{x_0}	
x_1	w_{x_1}	w_{x_0, x_1}
x_2	w_{x_2}	w_{x_0, x_1, x_2}
\vdots	\vdots	w_{x_1, x_2, x_3}
\vdots	\vdots	\vdots
x_{n-2}	$w_{x_{n-2}}$	$w_{x_{n-3}, x_{n-2}}$
x_{n-1}	$w_{x_{n-1}}$	$w_{x_{n-3}, x_{n-2}, x_{n-1}}$
x_n	w_{x_n}	$w_{x_{n-2}, x_{n-1}, x_n}$

Jej elementy nazywane są *różnicami dzielonymi* i zdefiniowane są następująco

$$w_{x_k} = w(x_k) = d_k,$$

$$\begin{aligned}
 w_{x_k, x_{k+1}} &= \frac{w_{x_{k+1}} - w_{x_k}}{x_{k+1} - x_k}, \\
 w_{x_k, x_{k+1}, x_{k+2}} &= \frac{w_{x_{k+1}, x_{k+2}} - w_{x_k, x_{k+1}}}{x_{k+2} - x_k}, \\
 &\vdots \\
 w_{x_k, x_{k+1}, \dots, x_{k+l}} &= \frac{w_{x_{k+1}, \dots, x_{k+l}} - w_{x_k, \dots, x_{k+l-1}}}{x_{k+l} - x_k},
 \end{aligned}$$

przy czym wzory są prawdziwe dla takich $k \in \mathbb{N}$ oraz $l \in \mathbb{N} \setminus \{0\}$, które wyznaczają poprawne indeksy węzłów. Liczby $w_{x_k, x_{k+1}, \dots, x_{k+l}}$ nazywamy *różnicami dzielonymi l-tego rzędu*⁷

Definicję łatwo zapamiętać przy pomocy powyższej tabeli. Każda różnica dzielona (z wyjątkiem tych z pierwszej kolumny) jest wyrażona ułamkiem, w którego liczniku są odejmowane różnice dzielone stojące na lewo, a w mianowniku odpowiednie węzły.

Można wykazać (np. przez indukcję), że

$$w_{x_k, x_{k+1}, \dots, x_{k+l}} = \sum_{i=k}^{k+l} \frac{d_i}{\prod_{j=k, j \neq i}^{k+l} (x_i - x_j)}.$$

Na podstawie twierdzenia, które było dowodzone przed omówieniem algorytmu pierwszego stwierdzamy, że szukane współczynniki w bazie Newtona spełniają równość

$$b_i = w_{x_0, \dots, x_i} \text{ dla } i = 0, 1, \dots, n,$$

czyli zapisane są w pierwszym (ukośnym) wierszu tabeli.

Przykład Dane są węzły $-1, 1, 2$ oraz wielomian w o współczynnikach $-11, 9$ i 13 w bazie Lagrange'a, czyli

$$w(x) = -11l_0(x) + 9l_1(x) + 13l_2(x).$$

Szukamy jego współczynników w bazie Newtona zdefiniowanej węzłami -1 i 1 , czyli w bazie $N_0(x) = 1, N_1(x) = x + 1, N_2(x) = (x + 1)(x - 1)$. Budujemy tabelę różnic dzielonych.

-1	-11	$\frac{9 - (-11)}{1 - (-1)} = 10$
1	9	$\frac{4 - 10}{2 - (-1)} = -2$
2	13	$\frac{13 - 9}{2 - 1} = 4$

Zatem $w(x) = -11N_0(x) + 10N_1(x) - 2N_2(x)$.

W algorytmie *różnic dzielonych Newtona* kolejno obliczamy kolumny tabeli. Używamy w tym celu tablicy y , która zawiera początkowo wartości d_i , oraz tablicy x przeznaczonej do przechowywania węzłów.⁸ Ponumerujmy kolumny tabeli różnic dzielonych w kierunku od lewej do prawej zaczynając od 0 , przy czym nie liczymy kolumny z węzłami. Zauważmy, że

1. Wyznaczenie k -tej kolumny tabeli wymaga znajomości jedynie kolumny $(k-1)$ -ej oraz węzłów. Nie będzie zatem potrzebna tablica kwadratowa — będziemy nadpisywać tablicę y .

⁷Różnice dzielone oznacza się również przez $w[x_k, x_{k+1}, \dots, x_{k+l}]$.

⁸Nie zawsze tablica x jest potrzebna — np. gdy węzły są równoodległe.

2. Kolumna k -ta jest o jeden krótsza od $(k-1)$ -ej — zwalnia się jedno miejsce, które możemy przeznaczyć na zapamiętanie b_k .
3. Każdą kolumnę należy obliczać „od dołu” — wtedy nie nadpiszemy potrzebnych jeszcze danych, a obliczone współczynniki będą kolejno pojawiały się w górnej części.

0	$w_{x_0} = b_0$	b_0	b_0	\dots	b_0
1	w_{x_1}	$w_{x_0, x_1} = b_1$	b_1	\dots	b_1
2	w_{x_2}	w_{x_1, x_2}	$w_{x_0, x_1, x_2} = b_2$	\dots	b_2
\vdots	\vdots	\vdots	\vdots	\dots	\dots
$n-2$	$w_{x_{n-2}}$	$w_{x_{n-3}, x_{n-2}}$	$w_{x_{n-4}, x_{n-3}, x_{n-2}}$	\dots	b_{n-2}
$n-1$	$w_{x_{n-1}}$	$w_{x_{n-2}, x_{n-1}}$	$w_{x_{n-3}, x_{n-2}, x_{n-1}}$	\dots	b_{n-1}
n	w_{x_n}	w_{x_{n-1}, x_n}	$w_{x_{n-2}, x_{n-1}, x_n}$	\dots	b_n
$i \uparrow k \rightarrow 0$	0	1	2	\dots	n

Zerowa kolumna jest dana. Obliczenie pierwszej kolumny ($k = 1$) można przeprowadzić tak.

```
for(i=n;i>=2;i--)
    y[i]=(y[i]-y[i-1])/(x[i]-x[i-1]);
```

Cały algorytm przedstawia się następująco.

```
for(k=1;k<=n;k++)
    for(i=n;i>=k;i--)
        y[i]=(y[i]-y[i-1])/(x[i]-x[i-k]);
```

Policzmy koszt. Treść pętli wewnętrznej kosztuje $2 \text{ ops} + 1 \text{ opm}$. Jest ona wykonywana $n - k + 1$ razy. Cały koszt pętli wewnętrznej zależy od k i wynosi $2(n - k + 1) \text{ ops} + (n - k + 1) \text{ opm}$. Pętla zewnętrzna instruuje jedynie, że wewnętrzna ma być wykonana dla $k = 1, 2, \dots, n$, zatem cały koszt algorytmu to

$$\begin{aligned}
 &\sum_{k=1}^n [2(n - k + 1) \text{ ops} + (n - k + 1) \text{ opm}] \\
 &= n(n + 1) \text{ ops} + \frac{1}{2} n(n + 1) \text{ opm}.
 \end{aligned}$$

Jakość zależy od uporządkowania węzłów. Jeśli są uporządkowane rosnąco lub malejąco, to algorytm jest numerycznie poprawny ze względu na każdy iloraz różnicowy osobno.

Twierdzenie 6. Niech

$$\begin{aligned}
 \phi : \mathbb{R}^{2n+2} &\rightarrow \mathbb{R}^{n+1}, \\
 \phi(x_0, x_1, \dots, x_n, d_0, d_1, \dots, d_n) &= (b_0, b_1, \dots, b_n),
 \end{aligned}$$

gdzie x_i, d_i, b_i mają znaczenie opisane wyżej. Wtedy algorytm różnic dzielonych $\mathcal{A} : \mathbb{R}^{2n+2} \rightarrow \mathbb{R}^{n+1}$ rozwiązujący zadanie ϕ jest prawie numerycznie poprawny, tzn.

$$\mathcal{A}(b_k) = \sum_{i=0}^k \frac{d_i(1 + \delta_i^{(k)})}{\prod_{j=0, j \neq i}^k (x_i - x_j)} \quad \text{dla } k = 0, 1, \dots, n,$$

gdzie $|\delta_i^{(k)}| \leq 3k\nu$.

Tak dobrze jest jedynie dla węzłów w kolejności rosnącej lub malejącej. W innych wypadkach nie musi być nawet numerycznie stabilny.

Zauważmy, że mamy tu pewną sprzeczność — algorytm jest dobry wtedy, gdy węzły generują bazę Newtona o dużej skośności i na odwrotnie. Można go stosować dla $n \leq 10$, wtedy wybiera się węzły uporządkowane. W innych przypadkach trzeba stosować pierwszy, droższy algorytm.

5.2.3 Algorytm różnic dzielonych w przypadku węzłów równoodległych

Przypuśćmy, że obecnie węzły są równoodległe, czyli

$$x_i = a + ih,$$

dla pewnego $a \in \mathbb{R}$ i $h > 0$. Wtedy

$$\begin{aligned} w_{x_k} &= w(x_k) = d_k, \\ w_{x_k, x_{k+1}} &= \frac{w_{x_{k+1}} - w_{x_k}}{h}, \\ w_{x_k, x_{k+1}, x_{k+2}} &= \frac{w_{x_{k+1}, x_{k+2}} - w_{x_k, x_{k+1}}}{2h}, \\ &\vdots \\ w_{x_k, x_{k+1}, \dots, x_{k+l}} &= \frac{w_{x_{k+1}, \dots, x_{k+l}} - w_{x_k, \dots, x_{k+l-1}}}{lh}. \end{aligned}$$

Idea algorytmu polaga na tym, by konstruować nieco inną tabelę różnic dzielonych — bez wykonywania dzieleni.

x_0	ω_{x_0}					
x_1	ω_{x_1}	ω_{x_0, x_1}	ω_{x_0, x_1, x_2}			
x_2	ω_{x_2}	ω_{x_1, x_2}	ω_{x_1, x_2, x_3}			
\vdots	\vdots	\vdots	\vdots			
\vdots	\vdots	\vdots	\vdots	\cdots	ω_{x_0, \dots, x_n}	
x_{n-2}	$\omega_{x_{n-2}}$	$\omega_{x_{n-3}, x_{n-2}}$	$\omega_{x_{n-3}, x_{n-2}, x_{n-1}}$			
x_{n-1}	$\omega_{x_{n-1}}$	$\omega_{x_{n-2}, x_{n-1}}$	$\omega_{x_{n-2}, x_{n-1}, x_n}$			
x_n	ω_{x_n}					

Wielkości występujące powyżej są zdefiniowane przez zależności

$$\begin{aligned} \omega_{x_k} &= w(x_k) = d_k, \\ \omega_{x_k, x_{k+1}} &= \omega_{x_{k+1}} - \omega_{x_k}, \\ \omega_{x_k, x_{k+1}, x_{k+2}} &= \omega_{x_{k+1}, x_{k+2}} - \omega_{x_k, x_{k+1}}, \\ &\vdots \\ \omega_{x_k, x_{k+1}, \dots, x_{k+l}} &= \omega_{x_{k+1}, \dots, x_{k+l}} - \omega_{x_k, \dots, x_{k+l-1}}. \end{aligned}$$

Można udowodnić (przez indukcję), że

$$w_{x_k, x_{k+1}, \dots, x_{k+l}} = \frac{\omega_{x_k, x_{k+1}, \dots, x_{k+l}}}{l!h^l}.$$

Stąd szukane współczynniki b_i wyrażają się wzorem

$$b_i = \frac{\omega_{x_0, x_1, \dots, x_i}}{i!h^i} \quad \text{dla } i = 0, 1, \dots, n.$$

Zapisanie algorytmu pozostawiam jako ćwiczenie. Zwróćmy uwagę, że ta wersja algorytmu jest znacznie tańsza: $\frac{1}{2}n(n+1)$ ops + $3n$ opm. Jakość też jest lepsza niż wersji ogólnej.

Okazuje się, że dla zadań praktycznych zachodzi często, że liczby d_i oraz d_{i+1} niewiele się różnią. Wtedy nierzadko ich różnica jest wyliczana dokładnie — bez błędów zaokrągleń. Błędy powstają dopiero przy dzieleniu podczas obliczania b_i . Jeśli h jest potęgą dwójki, to i dzielenie przez h^i jest wykonane dokładnie. W naturalnych przypadkach mamy do czynienia z bardzo dobrą stabilnością tego algorytmu.

5.3 Baza naturalna → baza Czebyszewa

Dane: a_0, a_1, \dots, a_n — współczynniki w bazie naturalnej

Wyniki: c_0, c_1, \dots, c_n — współczynniki w bazie Czebyszewa

Obecnie wielomian dany jest w bazie naturalnej

$$w(x) = \sum_{i=1}^n a_i x^i,$$

szukamy takich współczynników c_i , by miał on postać

$$w(x) = \sum_{i=1}^n c_i T_i(x).$$

Przy wyprowadzaniu algorytmu będziemy korzystać z następującej własności wielomianów Czebyszewa

$$xT_k(x) = \frac{T_{k-1}(x) + T_{k+1}(x)}{2}, \quad \text{dla } k = 1, 2, \dots$$

Wynika ona bezpośrednio z definicji rekurencyjnej. W dalszej części tego punktu będziemy stosować skrót notacyjny

$$T_k = T_k(x).$$

Zacznijmy od przedstawienia w w sposób podobny do tego, który jest podstawą schematu Hornera,

$$w(x) = a_0 + x[a_1 + x[a_2 + \dots + x[a_{n-3} + x(a_{n-2} + x a_{n-1} + x^2 a_n)]]].$$

Przekształcimy obecnie wielomian w nawiasie okrągłym tak, by został on zapisany w bazie Czebyszewa:

$$\begin{aligned} a_{n-2} + x a_{n-1} + x^2 a_n \\ = a_{n-2} T_0 + a_{n-1} T_1 + a_n x T_1 \\ = a_{n-2} T_0 + a_{n-1} T_1 + a_n \frac{T_0 + T_2}{2} \\ = (a_{n-2} + \frac{a_n}{2}) T_0 + a_{n-1} T_1 + \frac{a_n}{2} T_2. \end{aligned}$$

Jest to początkowy krok algorytmu. W następnych coraz większe fragmenty wielomianu w będą zapisywane w bazie

Czebyszewa — począwszy od najbardziej zagnieźdzonych nawiasów. Dla wygody będziemy numerować kroki od $k = n - 2$ do $k = 0$. W k -tym kroku będziemy generować nowe współczynniki na podstawie tych z kroku $k + 1$ (faktycznie zmieniać się będą tylko współczynniki o wyższych indeksach).

krok	współczynniki							
	a_0	a_1	\dots	a_{n-4}	a_{n-3}	a_{n-2}	a_{n-1}	a_n
$n - 2$	\parallel	\parallel		\parallel	\parallel			
	\tilde{a}_0	\tilde{a}_1	\dots	\tilde{a}_{n-4}	\tilde{a}_{n-3}	\tilde{a}_{n-2}	\tilde{a}_{n-1}	\tilde{a}_n
$n - 3$	\parallel	\parallel		\parallel				
	$\tilde{\tilde{a}}_0$	$\tilde{\tilde{a}}_1$	\dots	$\tilde{\tilde{a}}_{n-4}$	$\tilde{\tilde{a}}_{n-3}$	$\tilde{\tilde{a}}_{n-2}$	$\tilde{\tilde{a}}_{n-1}$	$\tilde{\tilde{a}}_n$
$n - 4$	\parallel	\parallel		\parallel				
	$\tilde{\tilde{\tilde{a}}}_0$	$\tilde{\tilde{\tilde{a}}}_1$	\dots	$\tilde{\tilde{\tilde{a}}}_{n-4}$	$\tilde{\tilde{\tilde{a}}}_{n-3}$	$\tilde{\tilde{\tilde{a}}}_{n-2}$	$\tilde{\tilde{\tilde{a}}}_{n-1}$	$\tilde{\tilde{\tilde{a}}}_n$
\dots								

Zatem po wykonaniu kroku $k = n - 2$ otrzymujemy

$$w(x) = \tilde{a}_0 + x[\tilde{a}_1 + \dots + x[\tilde{a}_{n-3} + x(\tilde{a}_{n-2}T_0 + \tilde{a}_{n-1}T_1 + \tilde{a}_nT_2)]],$$

gdzie

$$\begin{aligned}\tilde{a}_i &= a_i \quad \text{dla } i = 0, 1, \dots, n - 3, \\ \tilde{a}_{n-2} &= a_{n-2} + \frac{a_n}{2}, \\ \tilde{a}_{n-1} &= a_{n-1}, \\ \tilde{a}_n &= \frac{a_n}{2}.\end{aligned}$$

W następnym kroku dokonujemy przekształcenia najbardziej zagnieźdzonego nawiasu kwadratowego:

$$\begin{aligned}&\tilde{a}_{n-3} + x(\tilde{a}_{n-2}T_0 + \tilde{a}_{n-1}T_1 + \tilde{a}_nT_2) \\ &= \tilde{a}_{n-3}T_0 + \tilde{a}_{n-2}xT_0 + \tilde{a}_{n-1}xT_1 + \tilde{a}_nxT_2 \\ &= \tilde{a}_{n-3}T_0 + \tilde{a}_{n-2}T_1 + \tilde{a}_{n-1}\frac{T_0 + T_2}{2} + \tilde{a}_n\frac{T_1 + T_3}{2} \\ &= \tilde{\tilde{a}}_{n-3}T_0 + \tilde{\tilde{a}}_{n-2}T_1 + \tilde{\tilde{a}}_{n-1}T_2 + \tilde{\tilde{a}}_nT_3,\end{aligned}$$

gdzie

$$\begin{aligned}\tilde{\tilde{a}}_{n-3} &= \tilde{a}_{n-3} + \frac{\tilde{a}_{n-1}}{2}, \\ \tilde{\tilde{a}}_{n-2} &= \tilde{a}_{n-2} + \frac{\tilde{a}_n}{2}, \\ \tilde{\tilde{a}}_{n-1} &= \frac{\tilde{a}_{n-1}}{2}, \\ \tilde{\tilde{a}}_n &= \frac{\tilde{a}_n}{2}.\end{aligned}$$

W ogólności w kroku k -tym mamy

$$\begin{aligned}&\hat{a}_k + x(\hat{a}_{k+1}T_0 + \hat{a}_{k+2}T_1 + \dots + \hat{a}_nT_{n-k-1}) \\ &= \hat{a}_kT_0 + \hat{a}_{k+1}xT_0 + \hat{a}_{k+2}xT_1 + \dots + \hat{a}_nxT_{n-k-1} \\ &= \hat{a}_kT_0 + \hat{a}_{k+1}T_1 + \hat{a}_{k+2}\frac{T_0 + T_2}{2} + \dots \\ &\quad + \hat{a}_n\frac{T_{n-k-2} + T_{n-k}}{2} \\ &= \hat{\hat{a}}_kT_0 + \hat{\hat{a}}_{k+1}T_1 + \hat{\hat{a}}_{k+2}T_2 + \dots + \hat{\hat{a}}_nT_{n-k},\end{aligned}$$

gdzie

$$\begin{aligned}\hat{a}_k &= \hat{a}_k + \frac{\hat{a}_{k+2}}{2}, \\ \hat{a}_{k+1} &= \hat{a}_{k+1} + \frac{\hat{a}_{k+3}}{2}, \\ \hat{a}_{k+2} &= \frac{\hat{a}_{k+2}}{2} + \frac{\hat{a}_{k+4}}{2}, \\ \hat{a}_{k+3} &= \frac{\hat{a}_{k+3}}{2} + \frac{\hat{a}_{k+5}}{2}, \\ &\dots \\ \hat{a}_{n-3} &= \frac{\hat{a}_{n-3}}{2} + \frac{\hat{a}_{n-1}}{2}, \\ \hat{a}_{n-2} &= \frac{\hat{a}_{n-2}}{2} + \frac{\hat{a}_n}{2}, \\ \hat{a}_{n-1} &= \frac{\hat{a}_{n-1}}{2}, \\ \hat{\hat{a}}_n &= \frac{\hat{a}_n}{2}.\end{aligned}$$

Okazuje się, że w algorytmie wystarczy użyć tylko jednej tablicy **a**. Początkowo zapisane są w niej współczynniki a_i wieleomianu w bazie naturalnej. Krok k -ty można zaprogramować następująco

1. najpierw wszystkie współczynniki o indeksach

$$i = k + 2, k + 3, \dots, n$$

dzielimy przez 2 i zapamiętujemy wyniki w tej samej tablicy **a**,

2. potem popraw współczynniki o indeksach

$$i = k, k + 1, \dots, n - 2,$$

czyli do współczynnika **a[i]** dodaj **a[i+2]**.

Oto cały algorytm

```
for(k=n-2;k>=0;k--){
    for(i=k+2;i<=n;i++)
        a[i]=a[i]/2;
    for(i=k;i<=n-2;i++)
        a[i]=a[i]+a[i+2];
}
```

Można też go zapisać nieco inaczej, łącząc pętle wewnętrzne w jedną.

```
for(k=n-2;k>=0;k--)
    for(i=k+2;i<=n;i++){
        a[i]=a[i]/2;
        a[i-2]=a[i-2]+a[i];
    }
```

Koszt wynosi $\frac{1}{2}n(n - 1)$ ops + $\frac{1}{2}n(n - 1)$ opm. Twierdzeń dotyczących jakości algorytmu nie zamieszczam. Ze względu na dużą skośność bazy naturalnej konieczne jest trzymanie danych współczynników a_i w arytmetyce o dużej precyzji.

5.4 Baza Czebyszewa → baza naturalna

Dane:	c_0, c_1, \dots, c_n — współczynniki w bazie Czebyszewa
Wyniki:	a_0, a_1, \dots, a_n — współczynniki w bazie naturalnej

Należy obecnie rozwiązać zadanie odwrotne do tego z ostatniego punktu, czyli wielomian dany jest w bazie Czebyszewa

$$w(x) = \sum_{i=1}^n c_i T_i(x)$$

i szukamy takich współczynników a_i , by miał on postać

$$w(x) = \sum_{i=1}^n a_i x^i.$$

Można ten algorytm wyprowadzić od podstaw, ale łatwiej za- uważyć, że wystarczy poprzedni wykonać „od końca”.

```
for(k=0;k<=n-2;k++)
    for(i=n;i>=k+2;i--) {
        a[i-2]=a[i-2]-a[i];
        a[i]=a[i]*2;
    }
```

Wtedy jeśli tablica **a** zawiera początkowo współczynniki c_i w bazie Czebyszewa, to po zakończeniu algorytmu będzie przechowywać szukane współczynniki a_i w bazie naturalnej. Koszt jest identyczny jak poprzednio: $\frac{1}{2}n(n-1)$ ops + $\frac{1}{2}n(n-1)$ opm.

5.5 Baza naturalna → baza Lagrange'a

Dane:	x_0, x_1, \dots, x_n — węzły,
	a_0, a_1, \dots, a_n — współczynniki w bazie naturalnej
Wyniki:	d_0, d_1, \dots, d_n — współczynniki w bazie Lagrange'a o węzłach x_0, x_1, \dots, x_n

Odpowiednio i wielokrotnie zastosowany schemat Hornera.

5.6 Baza naturalna → baza Newtona

Dane:	x_1, x_2, \dots, x_n — węzły,
	a_0, a_1, \dots, a_n — współczynniki w bazie naturalnej
Wyniki:	b_0, b_1, \dots, b_n — współczynniki w bazie Newtona'a o węzłach x_1, x_2, \dots, x_n

Odpowiednio i wielokrotnie zastosowany schemat Hornera.

5.7 Baza Newtona → baza naturalna

Dane:	x_1, x_2, \dots, x_n — węzły,
	b_0, b_1, \dots, b_n — współczynniki w bazie Newtona'a o węzłach x_1, x_2, \dots, x_n
Wyniki:	a_0, a_1, \dots, a_n — współczynniki w bazie naturalnej

Odpowiednio i wielokrotnie zastosowany schemat Hornera.

6 Obliczanie pochodnych wielomianów

6.1 Bezpośrednie obliczenie pochodnej

Zastosowanie m.in. metod przybliżonego wyznaczania pierwiastków wymaga niejednokrotnie obliczenia kilku pierwszych pochodnych. W przypadku wielomianu nie jest trudno podać wzór na np. pierwszą pochodną. Jeśli

$$w(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n,$$

to

$$\begin{aligned} w'(x) &= a_1 + 2a_2 x + \dots + n a_n x^{n-1} \\ &= b_0 + b_1 x + \dots + b_{n-1} x^{n-1}. \end{aligned}$$

Nietrudno podać obecnie algorytm obliczenia pochodnej w w punkcie α , czyli wielkości $w'(\alpha)$. Najpierw wyznaczamy współczynniki b_i wielomianu w' :

```
for(i=1;i<=n;i++)
    b[i-1]=i*a[i];
```

Potem algorytmem Hornera obliczamy $w'(\alpha)$:

```
w=b[n-1];
for(i=n-2;i>=0;i--)
    w=w*alfa+b[i];
```

Całkowity koszt wynosi zatem

$$n \text{ opm} + (n - 1)(\text{ops} + \text{opm}) = (n - 1) \text{ ops} + (2n - 1) \text{ opm}.$$

W celu obliczenia wyższych pochodnych można postępować podobnie. Zatem, by obliczyć m pierwszych pochodnych w punkcie α oraz wartość wielomianu w punkcie α , czyli wielkości $w^{(j)}(\alpha)$ dla $j = 0, 1, \dots, m$ potrzeba $m + 1$ razy wykonać algorytm Hornera oraz m razy obliczyć współczynniki wielomianu. Za każdym razem obniżamy stopień wielomianu, zatem koszt wynosi:

$$\frac{(2n - m - 1)m}{2} \text{ ops} + (2n - m)m \text{ opm}.$$

Okazuje się, że można taniej.

6.2 Obliczanie pochodnych wielomianu algorytmem Hornera

Obliczając $w(\alpha)$ algorytmem Hornera prócz szukanej wielkości wyznaczamy współczynniki wielomianu $v(x)$, powstałego poprzez dzielenie $w(x)$ przez jednomian $(x - \alpha)$, o ile oczywiście zastosujemy wariant nie z pojedynczą zmienną, lecz z tablicą:

```
w[n]=a[n];
for(i=n-1;i>=0;i--)
    w[i]=w[i+1]*alfa+a[i];
```

Wtedy obliczone współczynniki w_i spełniają zależność:

$$\begin{aligned} w(x) &= a_0 + a_1 x + \dots + a_n x^n \\ &= w_0 + (x - \alpha)[w_1 + w_2 x + \dots + w_n x^{n-1}] \\ &= w_0 + (x - \alpha)v(x). \end{aligned}$$

Obliczmy pochodną wielomianu w :

$$w'(x) = (x - \alpha)v'(x) + v(x).$$

Stąd pochodna wielomianu w w punkcie α wynosi

$$w'(\alpha) = v(\alpha).$$

Zatem, by obliczyć $w'(\alpha)$ należy przeprowadzić algorytm Hornera dla wielomianu v , którego współczynniki zapisane są w tablicy w na pozycjach $1, \dots, n$. Jeśli użyjemy tej samej tablicy w , to nowo obliczone współczynniki w_i będą spełniały:

$$\begin{aligned} v(x) &= w_1 + (x - \alpha)[w_2 + w_3 x + \dots + w_n x^{n-2}] \\ &= w_1 + (x - \alpha)u(x), \end{aligned}$$

czyli

$$\begin{aligned} w(x) &= w_0 + (x - \alpha)[w_1 + (x - \alpha)u(x)] \\ &= w_0 + (x - \alpha)w_1 + (x - \alpha)^2 u(x). \end{aligned}$$

Stąd

$$\begin{aligned} w'(x) &= w_1 + 2(x - \alpha)u(x) + (x - \alpha)^2 u'(\alpha), \\ w''(x) &= 2u(x) + 2(x - \alpha)u'(x) + 2(x - \alpha)u'(x) \\ &\quad + (x - \alpha)^2 u''(\alpha). \end{aligned}$$

Otrzymaliśmy, że

$$\begin{aligned} w(\alpha) &= w_0, \\ w'(\alpha) &= w_1, \\ w''(\alpha) &= 2!u(\alpha). \end{aligned}$$

Znów więc obliczenie drugiej pochodnej sprowadziło się do wykonania algorytmu Hornera, tym razem dla wielomianu stopnia $n - 2$.

Postępując dalej w sposób analogiczny dochodzimy do algorytmu:

```
for(i=0;i<=n;i++)
    w[i]=a[i];
for(j=0;j<=m;j++)
    for(i=n-1;i>=j;i--)
        w[i]=w[i+1]*alfa+w[i];
```

Po jego zakończeniu współczynniki w_i zawierają m znormalizowanych pochodnych w punkcie α :

$$w_i = \frac{w^{(i)}(\alpha)}{i!}, \quad i = 0, 1, \dots, m.$$

Koszt wynosi tu

$$\begin{aligned} &(n + (n - 1) + \dots + (n - m))(\text{ops} + \text{opm}) \\ &= \frac{(2n - m)(m + 1)}{2}(\text{ops} + \text{opm}). \end{aligned}$$

6.3 Algorytm Shaw-Trauba

Zakładamy tutaj, że $\alpha \neq 0$. Algorytm polega na wyznaczeniu wielkości T_n^j dla $j = 0, \dots, m$, które mają spełniać zależność

$$\frac{w^{(j)}(\alpha)}{j!} = \frac{T_n^j}{\alpha^j}.$$

Jeśli T_n^j zdefiniujemy następująco:

$$\begin{aligned} T_i^{-1} &= a_{n-i-1}\alpha^{n-i-1}, & i &= 0, \dots, n-1, \\ T_j^j &= a_n\alpha^n, & j &= 0, \dots, m, \\ T_i^j &= T_{i-1}^{j-1} + T_{i-1}^j, & j &= 0, \dots, m, \quad i = j+1, \dots, n \end{aligned}$$

to można udowodnić, że powyższa równość będzie spełniona. Rekurencyjna definicja T_n^j prowadzi do sformułowania algorytmu.

Aby obliczyć szukane wielkości trzeba wyznaczyć elementy następującej tablicy:

T_0^{-1}	T_1^{-1}	T_2^{-1}	\dots	T_m^{-1}	T_{m+1}^{-1}	\dots	T_{n-1}^{-1}	
A	T_1^0	T_2^0	\dots	T_m^0	T_{m+1}^0	\dots	T_{n-1}^0	T_n^0
A	T_2^1	\dots	T_m^1	T_{m+1}^1	\dots	T_{n-1}^1	T_n^1	
A	\dots	T_m^2	T_{m+1}^2	\dots	T_{n-1}^2	T_n^2		
\ddots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
A	T_{m+1}^m	\dots	T_{n-1}^m	T_n^m				

Wielkość A wynosi $A = a_n\alpha^n$. Liczby z górnego wiersza można obliczyć na początku, wynoszą one z definicji $T_i^{-1} = a_{n-i-1}\alpha^{n-i-1}$. Pozostałe wielkości T_i^j należy obliczać dodając element tabelki stojący z lewej strony w tym samym wierszu oraz element stojący wierszu wyżej sąsiadującego z lewej strony kolumny:

$$\downarrow \rightarrow T_i^j$$

Obliczenia można przeprowadzić używając jednej tablicy w rozmiaru $m + 1$. Początkowo umieszczone są w niej liczby A , na końcu T_m^j dla $j = 0, 1, \dots, m$. Indeks j będzie używany do numerowania pozycji w tablicy w . Algorytm będzie działał w n krokach, numerowanych indeksem i . W i -tym kroku wyznaczamy kolumnę powyższej tabelki zawierającą liczby T_i^j dla $j = 0, \dots, \min i, m$. Obliczanie elementów kolumny należy przeprowadzić od dołu. Na końcu elementy tablicy w należy podzielić przez odpowiednie potęgi α . Ponieważ są one używane kilkakrotnie, będziemy je trzymać w tablicy $palpha$. Współczynniki wielomianu umieszczone są w tablicy a .

```
/*obliczenie poteg alfa*/
palpha[0]=1;
palpha[1]=alfa;
for(i=2;i<=n;i++)
    palpha[i]=palpha[i-1]*alfa;
```

```

/*inicjalizacja tablicy w*/
A=a[n]*potalpha[n];
for(j=0;i<=m;j++)
    w[j]=A;
        /*obliczanie wielkości T_i^j*/
for(i=1;i<=n;i++)
{
    for(j=(i>m)?m:(i-1);j>=1;j--)
        w[j]+=w[j-1];
    w[0]+=a[n-i]*potalpha[n-i];
}
        /*obliczenie znormalizowanych pochodnych*/
for(j=1;i<=m;j++)
    w[j]/=potalpha[j];

```

Policzmy kosz tego algorytmu. Wykonujemy m dzielen, $2n$ mnożeń oraz

$$\begin{aligned}
& \sum_{i=1}^m \left(1 + \sum_{j=1}^{i-1} 1\right) + \sum_{i=m+1}^n \left(1 + \sum_{j=1}^m 1\right) \\
&= \sum_{i=1}^m i + \sum_{i=m+1}^n i = m = 1^n(1+m) \\
& \frac{(1+m)m}{2} + (m+1)(n-m) = \frac{1}{2}(m+1)(2n-m)
\end{aligned}$$

dodawań. Łącznie

$$\frac{1}{2}(m+1)(2n-m) \text{ ops} + (m+2n) \text{ opm.}$$

Można udowodnić, że algorytm Shaw-Trauba jest numerycznie poprawny — obliczone wyniki są dokładnymi wynikami wielomianu o nieco zaburzonych współczynnikach.

Zauważmy, że założenie $\alpha \neq 0$ jest za słabe. W trakcie działania algorytmu obliczamy bowiem wielkość α^n . Algorytm działa jedynie dla wielkości, dla których nie dojdzie w tym wypadku do nadmiaru, ani niedomiaru.

Uwaga. Jeśli chcemy znaleźć minimum lub maksimum lokalne, to nie zawsze warto obliczać pochodną i szukać jej miejsca zerowego. Są metody unikające obliczania pochodnej.

Interpolacja

Dorota Dąbrowska, UKSW

2017/18

Spis treści

7 Różne sformułowania zadania interpolacji	43	10.6 Przechowywanie splajnów	60
7.1 Klasyczne zadanie interpolacji	43	10.6.1 Przechowywanie splajnów kubicznych w lokalnych bazach potęgowych	60
7.2 Uogólnione klasyczne zadanie interpolacji	44	10.6.2 Reprezentacja splajnów w bazie B-splajnów	61
7.3 Zadanie interpolacji przedziałowej	44	10.7 Obliczanie wartości splajnu w punkcie	62
7.4 Istnienie i jednoznaczność, jakość rozwiązań .	45	10.7.1 Przypadek reprezentacji splajnu kubicznego w lokalnych bazach potęgowych	62
7.5 Problem zbieżności metod interpolacji klasycznej	45	10.7.2 Przypadek reprezentacji splajnu w bazie B-splajnów	62
8 Interpolacja Lagrange'a	46	11 Interpolacja splajnami kubicznymi	63
8.1 Sformułowanie zadania interpolacji Lagrange'a	46	11.1 Sformułowanie zadania interpolacji	63
8.2 Istnienie i jednoznaczność rozwiązań	46	11.2 Wyznaczenie współczynników w lokalnych bazach potęgowych splajnu naturalnego interpolującego funkcję	64
8.3 Numeryczne rozwiązywanie zadania interpolacji Lagrange'a	46	11.2.1 Układ równań	64
8.4 Reszta i błąd interpolacji	47	11.2.2 Rozwiązywanie układu równań	66
8.5 Oszacowania górne reszty i błędu dla funkcji mającej $n+1$ ciągły pochodnych	47	11.2.3 Algorytm wyznaczania współczynników splajnu	67
8.6 Zjawisko Rungego	49	11.2.4 Algorytm rozwiązywania układów równań z macierzą trójdziagonalną o dominującej przekątnej	67
8.7 Problem zbieżności interpolacji Lagrange'a . .	50	11.3 Wyznaczenie współczynników kubicznego splajnu interpolacyjnego reprezentowanego w bazie B-splajnów	68
8.8 Własności funkcji Lebesgue'a	52	11.4 Reszta interpolacyjna	69
8.9 Błędy interpolacji dla funkcji niekoniecznie posiadających $n+1$ ciągły pochodny	53	11.5 Zastosowania B-splajnów w grafice komputerowej	69
8.10 Jeszcze raz o mierze skośności bazy Lagrange'a	53		
9 Interpolacja Hermite'a	54	12 Wielomiany trygonometryczne	70
9.1 Sformułowanie zadania	54	12.1 Zespolone wielomiany trygonometryczne	70
9.2 Numeryczne rozwiązywanie zadania interpolacji Hermite'a	55	12.2 Reprezentacja zespolonych wielomianów trygonometrycznych	70
9.3 Oszacowanie reszty interpolacyjnej	56	12.3 Obliczanie wartości wielomianu zespolonego	70
10 Splajny, algorytmy obliczania wartości splajnów w punkcie	57	12.4 Rzeczywiste wielomiany trygonometryczne	70
10.1 Definicja splajnu kubicznego	57	12.5 Reprezentacja rzeczywistych wielomianów trygonometrycznych	70
10.2 Definicja splajnu k -tego rzędu	57	12.6 Obliczanie wartości wielomianu rzeczywistego	70
10.3 Splajny kubiczne prawo- i lewostronne swobodne oraz okresowe	58		
10.4 Przestrzeń liniowa splajnów	59		
10.5 B-splajny	59		

12.6.1 Algorytm Goertzel'a	71
13 Interpolacja wielomianami trygonometrycznymi	71
13.1 Sformułowanie zadania	71
13.2 Wyznaczanie współczynników wielomianu interpolacyjnego	72

7 Różne sformułowania zadania interpolacji

Wiele zjawisk fizycznych jest opisywanych przez funkcje, których dokładnie nie znamy, lecz potrafimy mierzyć wartości tych funkcji lub ich pochodnych dla określonych wartości argumentu. Przykładem może być położenie, prędkość i przyspieszenie poruszającego się obiektu. Zdarza się też, że funkcja jest precyzyjnie opisana, lecz wyznaczanie jej wartości jest dla danego zastosowania zbyt czasochłonne. W obu przypadkach chcielibyśmy na podstawie wartości funkcji w pewnych punktach (zwanych *węzłami*) znaleźć inną, „prostą” funkcję dobrze przybliżającą oryginalną. Ta tematyka jest podstawą zadań interpolacji i aproksymacji.

Należy zdać sobie sprawę, że nie ma uniwersalnej i zawsze dobrej metody przybliżania funkcji. Zawsze niezbędna jest dodatkowa wiedza dotycząca opisywanego zjawiska (np. zachowanie się pochodnych). Bez niej wybór metody jest niemożliwy. Poza tym przybliżenia są zwykle dobre jedynie dla punktów niezbyt odległych od punktów pomiarowych, dalekosądne prognozy niekoniecznie będą prawdziwe.

Przykład W jednej ze swoich książek¹ Wiatr powołuje się na prognozę przedstawioną przez GUS² dotyczące przyrostu naturalnego i bezkrytyczne wyciąga z niej wnioski. Prognoza opiera się o dane dotyczące liczby ludności pochodzące z różnych lat, w książce wymienione są następujące.

Rok	Liczba ludności
1650	545 mln
1750	728 mln
1850	1171 mln
1950	2517 mln
1970	3632 mln

Prognoza dotyczyła roku 2000, w którym wg niej miało być 6 129 734 mln ludności, w rzeczywistości było ponad 1000 razy mniej.

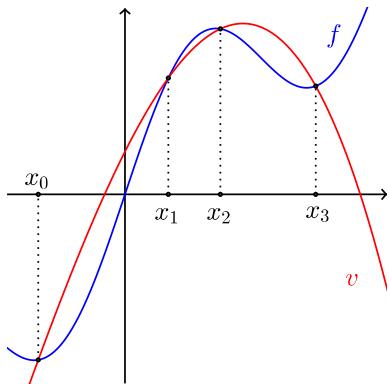
7.1 Klasyczne zadanie interpolacji

Dana jest przestrzeń F funkcji rzeczywistych lub zespolonych („trudnych”), które chcemy przybliżać oraz przestrzeń V funkcji odpowiednio rzeczywistych lub zespolonych („łatwych”), którymi będziemy przybliżali. Elementami przestrzeni V powinny być funkcje, dla których tanio można obliczać wartości.

Wybierzmy $f \in F$. Niech x_0, x_1, \dots, x_n oznaczają różne punkty z dziedziny funkcji f , będziemy nazywać je *węzłami*. Szukamy funkcji $v \in V$ spełniającej zależności $f(x_i) = v(x_i)$ dla $i = 0, 1, \dots, n$, nazywane *równaniami interpolacyjnymi*. Funkcję v nazywamy *funkcją interpolującą* f .

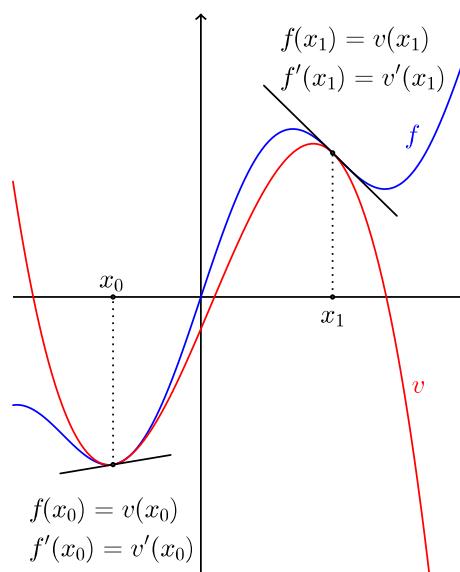
¹ Wiatr J.J., *Społeczeństwo. Wstęp do socjologii systematycznej*. Warszawa 1970.

² Rocznik demograficzny 1945-1966



Dane: x_0, x_1, \dots, x_n
 $f(x_0), f(x_1), \dots, f(x_n)$

Wynik: $v \in V$ taka, że $f(x_i) = v(x_i)$ dla $i = 0, 1, \dots, n$



$f(x_0) = v(x_0)$
 $f'(x_0) = v'(x_0)$

Dane: x_0, x_1, \dots, x_m
 k_0, k_1, \dots, k_m
 $f(x_0), f(x_1), \dots, f(x_n)$
 $f^{(1)}(x_0), f^{(1)}(x_1), \dots, f^{(1)}(x_n)$
 \vdots
 $f^{(k_1-1)}(x_0), f^{(k_2-1)}(x_1), \dots, f^{(k_m-1)}(x_n)$

Wynik: $v \in V$ taka, że $f^{(k)}(x_i) = v^{(k)}(x_i)$,
dla $i = 0, 1, \dots, m$ oraz $k = 0, 1, \dots, k_i - 1$.

Tradycyjnie dane zapisuje się w tzw. *tabelce interpolacyjnej*, której górny wiersz zawiera węzły, a dolny wartości funkcji:

x_0	x_1	\dots	x_n
$f(x_0)$	$f(x_1)$	\dots	$f(x_n)$

Od wyboru przestrzeni V zależy rodzaj i jakość interpolacji dla konkretnej funkcji $f \in F$, lub wszystkich funkcji z przestrzeni F . Jeśli V jest przestrzenią wielomianową, to mówimy o interpolacji wielomianowej Lagrange'a. W przypadku V będącego przestrzenią splajnów — o interpolacji splajnami, wielomianów trygonometrycznych — interpolacji trygonometrycznej, funkcji wymiernych — interpolacji wymiernej.

Przestrzeń F musi być odpowiednio „mała”. Dopuszczenie, by zawierała dowolne funkcje ciągłe powoduje, że zadanie interpolacji traci sens: dla dowolnej tabelki interpolacyjnej można dobrać funkcję ciągłą ją spełniającą i w punktach znajdujących się pomiędzy węzłami przyjmującą wartości dowolnie wielkie.

7.2 Uogólnione klasyczne zadanie interpolacji

Niektóre zastosowania wymagają, by funkcja interpolująca nie tylko dobrze przybliżała samą funkcję interpolowaną, ale też jej pochodne. Prowadzi to do uogólnienia zadania interpolacji.

Tak jak poprzednio dane są przestrzenie F i V , funkcja f oraz węzły: x_0, x_1, \dots, x_m . Z każdym węzłem związana jest jego *krotność*, czyli liczba naturalna informująca ile kolejnych wartości pochodnych musi zachowywać szukana funkcja v . Przyjmujemy tu umowę, że pochodne numerujemy od zera i zerowa pochodna oznacza wartość funkcji. Krotnościami będziemy oznaczać przez k_0, k_1, \dots, k_m , a ich sumę przez $n + 1$. Funkcja v , która interpoluje f musi spełniać równania interpolacyjne:

$$f^{(k)}(x_i) = v^{(k)}(x_i), \text{ dla } i = 0, 1, \dots, m, k = 0, 1, \dots, k_i - 1.$$

Postać tabelki interpolacyjnej jest następująca. W wierszu górnym każdy węzeł wypisujemy tyle razy, ile wynosi jego krotność. W dolny wierszu pod x_i znajdują się kolejno wartości pochodnych (od 0 do $k_i - 1$):

x_0	x_0	\dots	x_0	x_1	\dots
$f^{(0)}(x_0)$	$f^{(1)}(x_0)$	\dots	$f^{(k_0-1)}(x_0)$	$f^{(0)}(x_1)$	\dots

Zauważmy, że jeśli wszystkie krotności wynoszą 1, to zadanie sprowadza się do klasycznego zadania interpolacji. Zadanie uogólnionej interpolacji wielomianowej nosi miano interpolacji Hermite'a.

7.3 Zadanie interpolacji przedziałowej

Wartości funkcji, które mierzamy lub obliczamy są zwykle obarczone błędami. Nie można też zawsze zakładać, że dane są dokładne węzły — one też są wyznaczane przez urządzenia pomiarowe (np. zegary) lub są skutkiem obliczeń. Nowe podejście do interpolacji uwzględnia te fakty.

Jak wcześniej, dane są $F, V, f \in F$. Węzły będą reprezentowane przez przedziały $[x_i, \bar{x}_i]$ ($i = 1, 2, \dots, n$), które opisują wszystkie dopuszczalne wartości, jakie mogą się pojawić po uwzględnieniu błędów pomiarów lub obliczeń. Zakłada się przy tym, że przedziały są rozłączne.

Podobnie reprezentowane są wartości funkcji: przez przedziały $[y_i, \bar{y}_i]$, niekoniecznie rozłączne. Do przedziału $[y_i, \bar{y}_i]$ należą wszystkie wartości funkcji f dla punktów przedziału $[x_i, \bar{x}_i]$ obarczone dopuszczalnymi błędami.

Rozpatrzmy obecnie wszystkie funkcje v ze zbioru V spełniające warunki

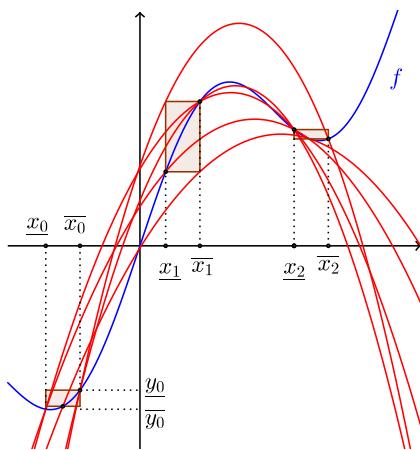
$$\forall i=1,2,\dots,n \exists x_i \in [\underline{x}_i, \bar{x}_i] \exists y_i \in [\underline{y}_i, \bar{y}_i] \quad v(x_i) = y_i.$$

Oznaczmy zbiór takich funkcji przez W .

Zadanie interpolacji rozumiane jest jako wyznaczenie dla dowolnego przedziału $[\underline{x}, \bar{x}]$ (reprezentującego pewien punkt z dziedziny f) zbioru

$$I([\underline{x}, \bar{x}]) = \{v(x) : x \in [\underline{x}, \bar{x}], v \in W\}$$

Zwykle zadowalamy się przedziałem, który jest dobrym przybliżeniem tego zbioru.



Dane: $\begin{bmatrix} [\underline{x}_0, \bar{x}_0], & [\underline{x}_1, \bar{x}_1] & \dots & [\underline{x}_n, \bar{x}_n] \\ [\underline{y}_0, \bar{y}_0], & [\underline{y}_1, \bar{y}_1] & \dots & [\underline{y}_n, \bar{y}_n] \end{bmatrix}$

Wynik: $I([\underline{x}, \bar{x}])$

7.4 Istnienie i jednoznaczność, jakość rozwiązania

Analiza zadań interpolacji wymaga odpowiedzi na następujące pytania.

1. Czy istnieje rozwiązanie zadania interpolacji?
2. Kiedy zadanie ma jednoznaczne rozwiązanie?
3. Jaka jest jakość interpolacji dla konkretnej funkcji f ?
4. Czy dla wszystkich funkcji z przestrzeni F jakość rozwiązania zadania interpolacji jest zadowalająca?
5. Jak opisać przestrzeń F , dla której jakość rozwiązania zadania interpolacji jest zadowalająca?

Ponieważ V jest przestrzenią liniową, to równania interpolacyjne zadania klasycznego sprowadzają się do układu równań liniowych, w którym niewiadomymi są współczynniki v w pewnej bazie przestrzeni V . W uogólnionym zadaniu układ również będzie liniowy, o ile pochodne funkcji z przestrzeni V również należą do V . Wtedy odpowiedź na dwa pierwsze pytania sprowadza się do analizy zbioru rozwiązań odpowiedniego układu równań liniowych. Zauważmy, że warunkiem koniecznym jednoznaczności rozwiązania jest, by liczba równań interpolacyjnych była nie mniejsza niż wymiar przestrzeni V .

Jakość interpolacji klasycznej (zwykłej lub uogólnionej) dla konkretnej funkcji f mierzymy przy użyciu pojęć reszty interpolacyjnej i błędu interpolacji. *Resztą interpolacyjną* nazываемy wielkość

$$r(x) = f(x) - v(x),$$

gdzie v interpoluje f , a x należy do dziedziny f . Wyraża ona punktową odległość między f i v . *Błędem interpolacji na zbiorze B* nazywamy

$$R = \sup_{x \in B} |r(x)|.$$

Zakładamy przy tym, że B jest domkniętym i ograniczonym podzbiorem dziedziny funkcji f . Zwykle zbiór B jest przedziałem domkniętym. Błąd interpolacji wskazuje najgorsze zachowanie reszty interpolacyjnej na zbiorze B .

W dalszej części będziemy stosować poniższe oznaczenie. Ustalmy funkcję f i zbiór B zawarty w jej dziedzinie. Przez $\|f\|_B$ będziemy rozumieć wielkość

$$\|f\|_B = \sup_{x \in B} |f(x)|$$

nazywaną *normą supremum*.³ Zauważmy, że $R = \|r\|_B$.

W prosty sposób pojęcia reszty i błędu interpolacji można uogólnić dla interpolacji przedziałowej. Można przyjąć na przykład

$$r([\underline{x}, \bar{x}]) = \sup_{x \in [\underline{x}, \bar{x}]} \sup_{y \in I([\underline{x}, \bar{x}])} (f(x) - y),$$

$$R_\varepsilon = \sup_{\substack{[\underline{x}, \bar{x}] \subset B \\ |\bar{x} - \underline{x}| < \varepsilon}} |r([\underline{x}, \bar{x}])|.$$

Parametr ε mierzy „dokładność”, z jaką opisywany jest punkt dziedziny.

Ocena jakości interpolacji dla całej przestrzeni F lub wskazanie takiej F , że jakość jest zadowalająca jest zadaniem najtrudniejszym i wymaga odrębnych analiz dla każdego rodzaju interpolacji.

7.5 Problem zbieżności metod interpolacji klasycznej

Rozpatrzmy obecnie sytuację, gdy przy ustalonym sposobie interpolacji będziemy zagęszczać siatkę węzłów. Dokładniej, dany jest ciąg układów węzłów z przedziału $[a, b]$ takich, że

- $n - ty$ element ciągu ma składa się z $n + 1$ węzłów,
- maksymalna odległość między sąsiednimi węzłami dla k tego układu jest większa niż dla $k + 1$ -go układu.

Metoda interpolacji jest wyznaczona przez wybór F , V , $f \in F$, oraz ciągu układu węzłów i polega na wyznaczeniu ciągu funkcji $v_n \in V$ interpolujących f w kolejnych układach węzłów.

Intuicja podpowiada, że im gęstsza siatka, tym jakość interpolacji powinna być lepsza. Niemniej nie zawsze tak jest. W opisany przypadku jakość mierzmy pojęciem zbieżności.

³Można udowodnić, że jest to rzeczywiście norma w przestrzeni funkcji ciągłych na B .

Definicja 1. Mówimy, że metoda interpolacji jest *zbieżna* jeśli

$$\forall_{x \in [a,b]} \lim_{n \rightarrow \infty} v_n(x) = f(x).$$

Mówimy, że metoda interpolacji jest *zbieżna jednostajnie* jeśli ciąg v_n zbiega jednostajnie do f na $[a,b]$.

Najbardziej „wartościowe” są metody zbieżne jednostajnie. Gwarantują one uzyskanie dobrej jakości interpolacji dla dośćcznie dużych n .

8 Interpolacja Lagrange'a

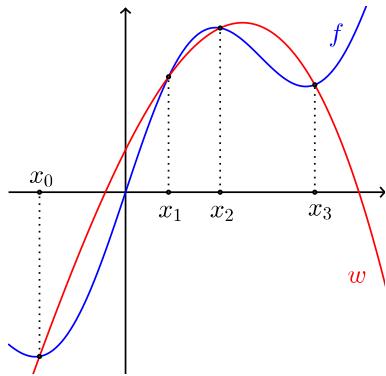
8.1 Sformułowanie zadania interpolacji Lagrange'a

Interpolacja Lagrange'a jest zadaniem klasycznym, które zostało sformułowane i rozwiążane około 200 lat temu. Dana jest przestrzeń F funkcji rzeczywistych („trudnych”), które chcemy przybliżać oraz przestrzeń $V = \Pi_n$ wielomianów stopnia co najwyżej n (przestrzeń funkcji „prostych”), którymi będziemy przybliżali elementy $f \in F$.

Wybierzmy $f \in F$. Niech x_0, x_1, \dots, x_n oznaczają *różne* punkty z dziedziny funkcji f , będącymi nazywali je *węzłami*. Szukamy wielomianu $w \in \Pi_n$ spełniającego zależności

$$f(x_i) = w(x_i) \text{ dla } i = 0, 1, \dots, n,$$

które nazywane są *równaniami interpolacyjnymi*. Funkcję w nazywamy *wielomianem Lagrange'a interpolującym* f w węzłach x_0, x_1, \dots, x_n .



Podsumujmy sformułowanie zadania w tabeli.

Dane:	x_0, x_1, \dots, x_n
	$f(x_0), f(x_1), \dots, f(x_n)$
Wynik:	$w \in \Pi_n$ taki, że $f(x_i) = w(x_i)$ dla $i = 0, 1, \dots, n$

Tradycyjnie dane zapisuje się w tzw. *tabelce interpolacyjnej*, której górnny wiersz zawiera węzły, a dolny wartości funkcji:

x_0	x_1	\dots	x_n
$f(x_0)$	$f(x_1)$	\dots	$f(x_n)$

8.2 Istnienie i jednoznaczność rozwiązań

Czy dla dowolnej funkcji $f \in F$ istnieje wielomian Lagrange'a $w \in \Pi_n$ interpolujący f w węzłach x_0, x_1, \dots, x_n ? Tak, bo

$$w(x) = \sum_{i=0}^n f(x_i) l_i(x),$$

gdzie l_i są funkcjami Lagrange'a, spełnia warunki interpolacyjne $w(x_i) = f(x_i)$ dla $i = 0, 1, \dots, n$.

Czy taki wielomian jest wyznaczony jednoznacznie? Tak, bo gdyby istniały dwa różne wielomiany $w, v \in \Pi_n$ spełniające równania interpolacyjne

$$w(x_i) = f(x_i) = v(x_i), \quad i = 0, 1, \dots, n.$$

to wielomian $h = w - v$ byłby również elementem przestrzeni Π_n i nie byłby tożsamociwie równy zero. Zauważmy, że $h(x_i) = 0$ dla $i = 0, 1, \dots, n$, zatem h ma $n+1$ różnych miejsc zerowych, co daje sprzeczność, bo $h \in \Pi_n$.

8.3 Numeryczne rozwiązanie zadania interpolacji Lagrange'a

Numerycznie rozwiązuje się to zadanie na dwa sposoby.⁴

1. Należy zapisać wielomian interpolacyjny Lagrange'a w bazie Lagrange'a i w niej wyznaczać wartości tego wielomianu.
2. Można przejść do bazy Newtona przy pomocy jednego z omówionych algorytmów, następnie posługiwać się ogólnym schematem Hornera.

Przykład Dana jest funkcja $f(x) = \sin(x)$. Chcemy ją interpolować wielomianem w punktach $x_0 = -\frac{\pi}{2}$, $x_1 = 0$, $x_2 = \frac{\pi}{2}$. Zatem szukany wielomian $w \in \Pi_2$ musi spełniać zależności

$$w\left(-\frac{\pi}{2}\right) = \sin\left(-\frac{\pi}{2}\right), \quad w(0) = \sin(0), \quad w\left(\frac{\pi}{2}\right) = \sin\left(\frac{\pi}{2}\right).$$

Metoda 1. Rozwiązaniem jest

$$w(x) = -1 \cdot l_0(x) + 0 \cdot l_1(x) + 1 \cdot l_2(x)$$

gdzie

$$\begin{aligned} l_0(x) &= \frac{2}{\pi^2} x \left(x - \frac{\pi}{2}\right), \\ l_1(x) &= -\frac{4}{\pi^2} \left(x + \frac{\pi}{2}\right) \left(x - \frac{\pi}{2}\right), \\ l_2(x) &= \frac{2}{\pi^2} x \left(x + \frac{\pi}{2}\right). \end{aligned}$$

⁴Teoretycznie można takich sposobów generować więcej. Wystarczy równania interpolacyjne zapisać w postaci układu równań liniowych, w którym niewiadomymi są współczynniki wielomianu w jakiejś bazie. Następnie zastosować znany (np. Gaussa) algorytm rozwiązywania układów równań. Takie podejście ma dwie poważne wady — jest kosztowne ($\Theta(n^3)$ operacji, gdy układ jest pełny) i może być zlej jakości, gdy miara skośności bazy jest duża lub zadanie rozwiązywania układu równań źle uwarunkowane.

Metoda 2. Zastosujmy algorytm różnic dzielonych dla węzłów równoodległych.

$$\begin{array}{c|ccc} -\frac{\pi}{2} & -1 & & \\ \hline 0 & 0 & 1 & \\ \hline \frac{\pi}{2} & 1 & & \end{array}$$

Zapiszmy w w bazie Newtona o węzłach $-\frac{\pi}{2}$ i 0 . Współczynniki w tej bazie wynoszą

$$b_0 = 1, \quad b_1 = \frac{1}{1!h} = \frac{2}{\pi}, \quad b_2 = \frac{0}{2!h^2} = 0,$$

zatem

$$w(x) = -1 \cdot N_0(x) + \frac{2}{\pi} N_1(x) + 0 \cdot N_2(x),$$

gdzie

$$N_0(x) = 1, \quad N_1(x) = x + \frac{\pi}{2}, \quad N_2(x) = x \left(x + \frac{\pi}{2} \right).$$

8.4 Reszta i błąd interpolacji

Jakość interpolacji dla konkretnej funkcji f mierzymy przy pomocy pojęć reszty interpolacyjnej i błędu interpolacji.

Definicja 2. Resztą interpolacyjną nazywamy funkcję

$$r(x) = f(x) - w(x),$$

gdzie w interpoluje f w węzłach x_0, x_1, \dots, x_n i x należy do dziedziny f .

Reszta interpolacyjna wyraża punktową odległość między f i w .

Definicja 3. Błądem interpolacji na przedziale $[a, b]$ nazywamy liczbę

$$R = \sup_{x \in [a, b]} |r(x)|,$$

zakładamy, że $[a, b]$ jest podzbiorem dziedziny funkcji f .

Błąd interpolacji wskazuje najgorsze zachowanie reszty interpolacyjnej na przedziale $[a, b]$.

W dalszej części będziemy często używać normy supremum.⁵

$$\|f\|_{[a, b]} = \sup_{x \in [a, b]} |f(x)|.$$

⁵Przypomnijmy, jak rozwiążujemy zadanie znalezienia maksimum globalnego funkcji f na pewnym zbiorze. Jeśli funkcja f jest różniczkowalna, to „kandydatami” na maksimum globalne są:

1. maksima lokalne (szukamy ich wśród punktów, gdzie f' się zeruje),
2. punkty z brzegu zbioru, czyli w przypadku przedziału $[a, b]$ punkty a i b .

Dla każdego z „kandydatów” obliczamy wartość funkcji f i wybieramy ten, dla którego jest ona największa. Jeśli szukamy maksimum globalnego funkcji $|f|$, to nawet, gdy f jest różniczkowalna, to $|f|$ już nie musi. Niemniej zadanie nadal łatwo rozwiązać — szukamy zarówno maksimum jak i minimum globalnego funkcji f , a następnie wybieramy ten z dwóch punktów, dla którego moduł wartości f jest większy. Szukanie minimum odbywa się analogicznie do maksimum i można te dwa obliczenia połączyć.

Czasami będziemy pisać $\|f\|$ zamiast $\|f\|_{[a, b]}$, o ile z kontekstu wiadomo, o jaki przedział chodzi. Zauważmy, że

$$R = \|r\|_{[a, b]}.$$

8.5 Oszacowania górne reszty i błędu dla funkcji mającej $n+1$ ciągły pochodnych

Podstawowe własność reszty interpolacyjnej w przypadku funkcji dostatecznie gładkiej⁶ podają poniższe twierdzenia. Pierwsze jest podstawą dowodów innych twierdzeń, samo w sobie nie ma praktycznego znaczenia.

Twierdzenie 1. Dane są węzły interpolacyjne x_0, x_1, \dots, x_n oraz funkcja f . Niech w będzie wielomianem Lagrange'a interpolującym f . Wtedy

$$r(x) = w_{x_0, x_1, \dots, x_n, x} \prod_{i=0}^n (x - x_i).$$

Warto zauważyć, że różnica dzielona $w_{x_0, x_1, \dots, x_n, x}$ zależy jedynie od wartości funkcji w punktach x, x_0, x_1, \dots, x_n i żeby ją obliczyć wcale nie trzeba znać wielomianu interpolacyjnego w — wystarczy skonstruować odpowiednią tabelkę różnic dzielonych. Stąd stosuje się często oznaczenie⁷ $f_{x_0, x_1, \dots, x_n, x}$ zamiast $w_{x_0, x_1, \dots, x_n, x}$.

Poniższe twierdzenie jest wnioskiem z poprzedniego (ale dowód jest dość długi — ok. strony). Charakteryzuje ono zachowanie reszty w terminach $(n+1)$ pochodnej funkcji. Ma ono zatem zastosowanie tylko dla funkcji, które tę pochodną posiadają.

Twierdzenie 2. Dany jest przedział $[a, b]$ zawierający wszystkie węzły x_0, x_1, \dots, x_n oraz punkt x . Niech rzeczywista funkcja f ma $n+1$ ciągły pochodnych na $[a, b]$. Wtedy istnieje punkt $\xi(x) \in [a, b]$ zależny od x taki, że

$$r(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \prod_{i=0}^n (x - x_i).$$

Punktu $\xi(x)$ nie jesteśmy w stanie określić dokładnie, znamy jedynie przedział, do którego należy. Ponadto dla każdego $x \in [a, b]$ może być inny. W praktyce bezpośrednie korzystanie z tego twierdzenia nie jest częste, zwykle posługujemy się wnioskami z niego płynącymi. Łatwo je wyprowadzić, co przedstawiam niżej.

Zastosujmy oznaczenie

$$N_{n+1}(x) = \prod_{i=0}^n (x - x_i).$$

⁶W matematyce gładkość funkcji oznacza posiadanie ciągłych pochodnych.

⁷Powszechnie przyjętymi oznaczeniami są również $f[x_0, x_1, \dots, x_n, x]$ i $w[x_0, x_1, \dots, x_n, x]$.

Zauważmy, że dla dowolnego $x \in [a, b]$ zachodzi

$$\begin{aligned} |r(x)| &= \frac{\|f^{(n+1)}(\xi(x))\|}{(n+1)!} |N_{n+1}(x)| \\ &\leq \frac{\|f^{(n+1)}\|}{(n+1)!} |N_{n+1}(x)|. \end{aligned}$$

Użyta norma $\|\cdot\|$ oznacza zawsze w tym punkcie normę supremum na przedziale $[a, b]$.

Wniosek 1. *Dany jest przedział $[a, b]$ zawierający wszystkie węzły x_0, x_1, \dots, x_n oraz punkt x . Niech rzeczywista funkcja f ma $n+1$ ciągły pochodnych na przedziale $[a, b]$. Wtedy*

$$|r(x)| \leq \frac{\|f^{(n+1)}\|}{(n+1)!} |N_{n+1}(x)|.$$

Zauważmy dalej, że dla dowolnego $x \in [a, b]$ mamy

$$|r(x)| \leq \frac{\|f^{(n+1)}\|}{(n+1)!} \|N_{n+1}\|,$$

zatem

$$R \leq \frac{\|f^{(n+1)}\|}{(n+1)!} \|N_{n+1}\|.$$

Wniosek 2. *Dany jest przedział $[a, b]$ zawierający wszystkie węzły x_0, x_1, \dots, x_n . Niech rzeczywista funkcja f ma $n+1$ ciągły pochodnych na przedziale $[a, b]$. Wtedy*

$$R \leq \frac{\|f^{(n+1)}\|}{(n+1)!} \|N_{n+1}\|.$$

Powyższa nierówność jest najczęściej używana w praktyce. Można zastanawiać się, czy oszacowanie nie jest zbyt zgrubne, czyli czy rzeczywiście prawa strona oddaje zachowanie błędu interpolacji. Okazuje się, że istnieją funkcje (nietrywialne), dla których to oszacowanie jest dokładne, o czym mowa później.

Twierdzenie 3. *Dany jest przedział $[a, b]$ zawierający wszystkie węzły x_0, x_1, \dots, x_n . Dla dowolnej liczby $M > 0$ istnieje funkcja f posiadająca $n+1$ ciągły pochodnych taka, że*

$$\|f^{(n+1)}\| = M$$

oraz

$$R = \frac{\|f^{(n+1)}\|}{(n+1)!} \|N_{n+1}\|.$$

Funkcję, o której mowa w twierdzeniu można wskazać — jest nią np.

$$f(x) = M \frac{x^{n+1}}{(n+1)!}$$

dla dowolnego $M > 0$. Wyznaczenie błędu interpolacji tej funkcji pozostawiam jako ćwiczenie.

Następne twierdzenie pozwala oszacować wielkość $\|N_{n+1}\|$. Występujące w nim założenie o uporządkowaniu węzłów jest istotne (funkcja N_{n+1} nie zmieni się, gdy przestawimy

węzły) — pozwala jedynie w zgrabny sposób wyrazić użytkę tam wielkość h . Oznacza ona długość najdłuższego przedziału wyznaczonego przez sąsiednie węzły — sąsiednie na osi liczbowej. Istotne jest natomiast założenie, że przedział $[a, b]$ jest wyznaczony przez skrajne węzły.

Twierdzenie 4. *Niech $a = x_0 < x_1 < \dots < x_n = b$ będą dowolnymi węzłami. Wtedy*

$$\|N_{n+1}\| \leq \frac{n!h^{n+1}}{4},$$

gdzie $h = \max_{i=0,1,\dots,n-1} (x_{i+1} - x_i)$.

Pozostawiam wykonanie dowodu jako ćwiczenie. Wykorzystajmy powyższe twierdzenie do kolejnego przedstawienia oszacowania na błąd interpolacji. Jest to bezpośredni wniosek z Twierdzenia 4 i Wniosku 2.

Wniosek 3. *Niech $a \leq x_0 < x_1 < \dots < x_n \leq b$ oraz rzeczywista funkcja f ma $n+1$ ciągły pochodnych na przedziale $[a, b]$. Wtedy*

$$R \leq \frac{\|f^{(n+1)}\|}{4(n+1)} h^{n+1},$$

gdzie

$$h = \max_{i=0,1,\dots,n-1} (x_{i+1} - x_i).$$

Dotychczas węzły traktowaliśmy jako z góry dane. Zdarza się, że sposób ich wyboru zależy od osoby implementującej algorytm. Wtedy powstaje pytanie, które są najlepsze. Pełna odpowiedź zależy zarówno od funkcji f , jak i przedziału $[a, b]$ i nie da jej się wyrazić w sposób prosty analitycznie. Ponadto f nie musi być znana.

Zastanówmy się więc, przy jakim wyborze węzłów oszacowanie z Wniosku 2 jest najlepsze. Po prawej jego stronie jedynie czynnik $\|N_{n+1}\|$ zależy od węzłów, zatem szukamy węzłów, dla których jest on najmniejszy. Okazuje się, że rozwiązaniem są węzły Czebyszewa.

Twierdzenie 5. *Dla dowolnych różnych węzłów x_0, x_1, \dots, x_n oraz przedziału $[a, b]$ zawierającego wszystkie węzły wyrażenie*

$$\|N_{n+1}\| = \sup_{x \in [a,b]} \left| \prod_{i=0}^n (x - x_i) \right|$$

jest najmniejsze dla węzłów Czebyszewa w $[a, b]$ i jego wartością jest wtedy

$$\frac{1}{2^n} \left(\frac{b-a}{2} \right)^{n+1}.$$

Dowód ponownie zostawiam jako ćwiczenie. Podkreślimy jeszcze raz, że wcale nie minimalizujemy błędu interpolacji na $[a, b]$, lecz jedynie jego oszacowanie. Przypomnijmy, że wybór węzłów Czebyszewa daje również małą miarę skośności bazy Lagrange'a.

Następny wniosek łączy Twierdzenie 5 z Wnioskiem 2.

Wniosek 4. Niech rzeczywista funkcja f ma $n+1$ ciągły pochodnych na pewnym przedziale $[a, b]$. Dokonujemy interpolacji f w węzłach Czebyszewa z $[a, b]$. Wtedy

$$R \leq \frac{\|f^{(n+1)}\|}{2^n(n+1)!} \left(\frac{b-a}{2}\right)^{n+1}.$$

Wszystkie (za wyjątkiem pierwszego) przedstawione w tym punkcie twierdzenia i wnioski dotyczą funkcji, o której zakładamy, że ma dostatecznie dużo ciągłych pochodnych. Jeśli ten warunek nie jest spełniony, to oszacowań jest dokonać trudniej i są one mniej dokładne. Pewna metoda zostanie przedstawiona w późniejszej części wykładu.

Przykład Weźmy, jak w poprzednim przykładzie funkcję $f(x) = \sin(x)$ oraz węzły $x_0 = -\frac{\pi}{2}$, $x_1 = 0$, $x_2 = \frac{\pi}{2}$. Oszacujmy wartość reszty interpolacyjnej w przedziale $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Zauważmy, że $\|f^{(3)}\| = 1$. Zatem z Wniosku 1

$$|r(x)| \leq \frac{1}{6} \left| x(x^2 - \frac{\pi^2}{4}) \right|.$$

Zajmijmy się obecnie błędem interpolacji. Z Wniosku 2 $R \leq \frac{1}{6} \|N_3\|$, gdzie $N_3 = x(x^2 - \frac{\pi^2}{4})$. Obliczmy $\|N_3\|$, czyli znajdziemy maksimum globalne funkcji $|N_3|$ na zbiorze $[a, b]$. W tym celu wyznaczmy ekstrema lokalne w przedziale $[a, b]$ funkcji N_3 . Ponieważ

$$N'_3(x) = 3x^2 - \frac{\pi^2}{4} = 3 \left(x - \frac{\pi\sqrt{3}}{6} \right) \left(x + \frac{\pi\sqrt{3}}{6} \right)$$

to ekstremami lokalnymi są $\frac{\pi\sqrt{3}}{6}$ i $-\frac{\pi\sqrt{3}}{6}$. Wartość funkcji $|N_3|$ jest w obu punktach równa i wynosi $\frac{\pi^3}{36}\sqrt{3}$. Wartości na brzegach przedziału wynoszą 0, zatem $\|N_3\| = \frac{\pi^3}{36}\sqrt{3}$ i

$$R \leq \frac{\pi^3}{216}\sqrt{3} \approx 0.25.$$

Posługując się Wnioskiem 3 otrzymalibyśmy

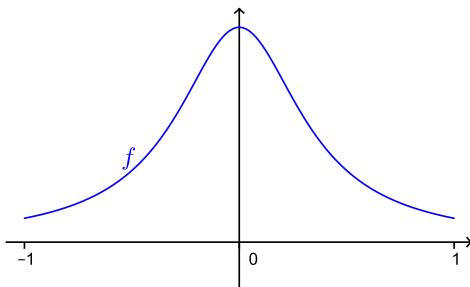
$$R \leq \frac{\pi^3}{12 \cdot 8} \approx 0.32.$$

Gdybyśmy wybrali węzły Czebyszewa $x_0 = -\frac{\pi\sqrt{3}}{4}$, $x_1 = 0$, $x_2 = \frac{\pi\sqrt{3}}{4}$, to z Wniosku 4 otrzymalibyśmy oszacowanie

$$R \leq \frac{1}{24} \left(\frac{\pi}{2}\right)^3 \approx 0.16.$$

8.6 Zjawisko Rungego

Rozważmy funkcję $f(x) = \frac{1}{1+8x^2}$ o wykresie

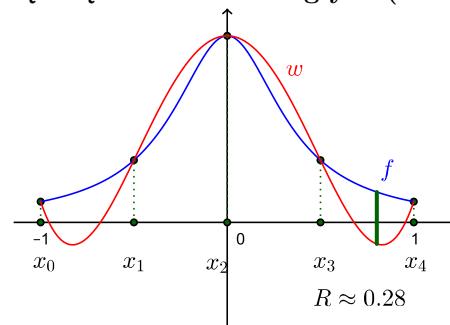


Będziemy ją interpolować przy użyciu x_0, x_1, \dots, x_n , które

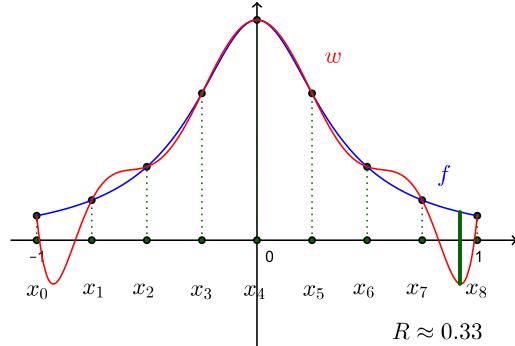
1. są węzłami równoodległymi dla $n = 4, 8, 16$,
2. są węzłami Czebyszewa dla $n = 4, 8, 16$.

Za każdym razem wyznaczany jest błąd interpolacji w przedziale $[-1, 1]$, czyli wielkość $R = \sup_{x \in [-1, 1]} |r(x)|$.

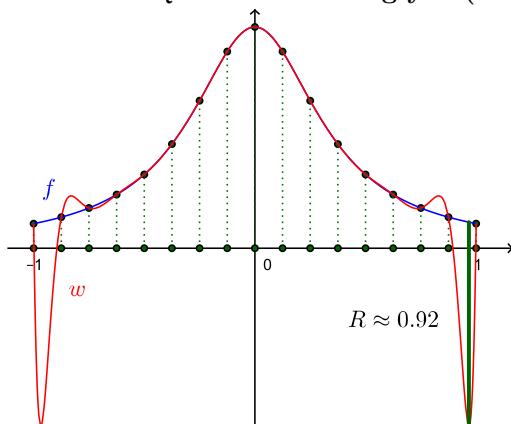
Pięć węzłów równoodległych ($n = 4$)



Dziewięć węzłów równoodległych ($n = 8$)



Siedemnaście węzłów równoodległych ($n = 16$)



Wbrew intuicji zwiększanie liczby węzłów powoduje pogorszenie jakości interpolacji. Duża reszta interpolacyjna nie dotyczy na szczeble całego przedziału, lecz tylko jego krańców. W tabeli zamieszczam przybliżone wartości błędu liczonego na przedziale $[-1, 1]$ oraz $[-0.75, 0.75]$.

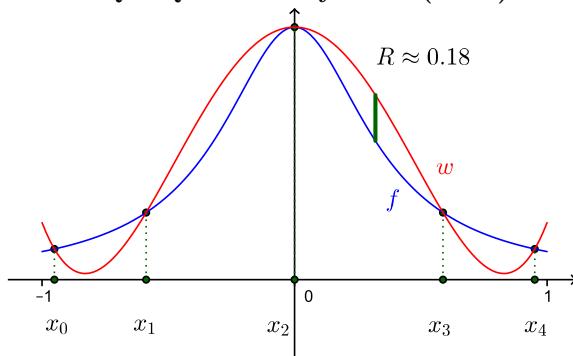
n	$\sup_{x \in [-1,1]} r(x) $	$\sup_{x \in [-0.75,0.75]} r(x) $
4	$2.8 \cdot 10^{-1}$	$2.6 \cdot 10^{-1}$
8	$3.3 \cdot 10^{-1}$	$7.1 \cdot 10^{-2}$
16	$9.2 \cdot 10^{-1}$	$1.4 \cdot 10^{-2}$
32	$1.3 \cdot 10^1$	$8.0 \cdot 10^{-3}$
64	$4.6 \cdot 10^3$	$2.6 \cdot 10^{-6}$

n	$\sup_{x \in [-1,1]} r(x) $	$\sup_{x \in [-0.75,0.75]} r(x) $
4	$1.8 \cdot 10^{-1}$	$1.8 \cdot 10^{-1}$
8	$4.2 \cdot 10^{-2}$	$4.2 \cdot 10^{-2}$
16	$2.7 \cdot 10^{-3}$	$2.7 \cdot 10^{-3}$
32	$1.1 \cdot 10^{-5}$	$1.1 \cdot 10^{-5}$
64	$1.6 \cdot 10^{-10}$	$1.6 \cdot 10^{-10}$

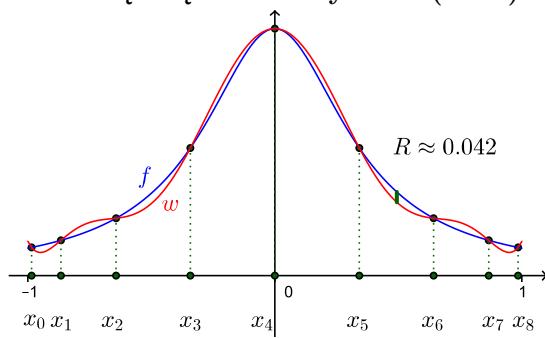
Zjawisko pogarszania się jakości interpolacji, które obserwujemy przy krańcach przedziału interpolacji, przy wzroście liczby węzłów (zagęszczających się) nosi miano *zjawiska Rungego*. Zauważmy, że węzły Czebyszewa pozwalają znacznie zredukować błąd.

Dobra jakość blisko krańców przedziału jest zasługą tego, że węzły Czebyszewa są tam gęściej rozmieszczone niż w okolicy 0. Proszę zwrócić uwagę, że nawet w przedziale $[-0.75, 0.75]$ błąd jest istotnie mniejszy (dla $n \geq 16$) niż w przypadku węzłów równoodległych.

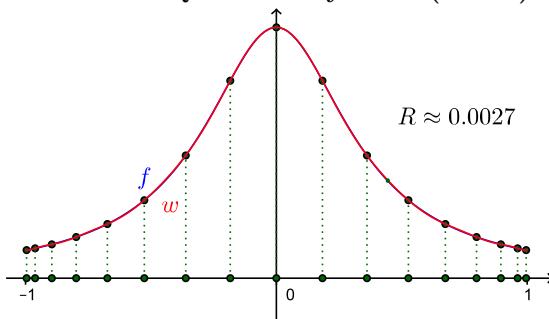
Pięć węzłów Czebyszewa ($n = 4$)



Dziewięć węzłów Czebyszewa ($n = 8$)



Siedemnaście węzłów Czebyszewa ($n = 16$)



Na ostatnim wykresie z racji na zbyt małą skalę wydaje się, że f i wielomian interpolacyjny są równe — tak oczywiście nie jest. Oto tabela z przybliżonymi wartościami błędów dla węzłów Czebyszewa.

8.7 Problem zbieżności interpolacji Lagrange'a

Powstają obecnie pytania:

1. Czy dla danej funkcji f zwiększenie liczby węzłów (z równoczesnym ich zagęszczeniem) poprawia jakość interpolacji?
2. Czy dla danej funkcji f można tak dobrą sposób zwiększać liczbę węzłów, by jakość interpolacji się poprawiała?
3. Czy istnieje „uniwersalny” (czyli niezależny od funkcji) sposób zwiększania liczby węzłów, by jakość interpolacji się poprawiała?

Zacznijmy od sprecyzowania, co mamy na myśli mówiąc o zagęszczaniu węzłów. Dany jest więc ciąg \mathcal{X}_n układów węzłów z przedziału $[a, b]$ takich, że

- n -ty element ciągu \mathcal{X}_n składa się z $n+1$ węzłów,
- maksymalna odległość między sąsiednimi na osi liczbowej węzłami dla n tego układu jest większa niż dla $(n+1)$ -go układu.

Dla danej funkcji f oraz ciągu \mathcal{X}_n konstruujemy ciąg wielomianów $w_n \in \Pi_n$ interpolujących f w kolejnych układach węzłów.

Definicja 4. Mówimy, że metoda interpolacji Lagrange'a wyznaczona przez ciąg \mathcal{X}_n jest *zbieżna* dla funkcji f jeśli ciąg w_n zbiega punktowo do f w przedziale $[a, b]$, czyli

$$\forall_{x \in [a,b]} \lim_{n \rightarrow \infty} w_n(x) = f(x).$$

Definicja 5. Mówimy, że metoda interpolacji Lagrange'a wyznaczona przez ciąg \mathcal{X}_n jest *zbieżna jednostajnie* dla funkcji f jeśli ciąg w_n zbiega jednostajnie do f na $[a, b]$, czyli

$$\lim_{n \rightarrow \infty} \|f - w_n\| = 0,$$

gdzie $\|\cdot\|$ oznacza normę supremum na przedziale $[a, b]$.

Druga definicja jest równoważna stwierdzeniu, że błąd interpolacji na $[a, b]$ dąży do zera wraz ze wzrostem n . Metody zbieżne jednostajnie gwarantują uzyskanie dobrej jakości interpolacji dla wszystkich punktów z $[a, b]$, o ile n jest dostatecznie duże. Przypomnijmy, że ze zbieżności jednostajnej wynika zbieżność punktowa, ale nie na odwrót.

Poniższe twierdzenie daje odpowiedź negatywną na pytanie 3. Zwróćmy uwagę, że dotyczy ono zbieżności jednostajnej.

Twierdzenie 6. *Dla dowolnego ciągu \mathcal{X}_n układów węzłów istnieje funkcja ciągła, dla której metoda interpolacji Lagrange'a wyznaczona przez ten ciąg nie jest zbieżna jednostajnie.*

Niemniej, jeśli zawężymy klasę funkcji do tak zwanych funkcji całkowitych,⁸ to odpowiedź się zmienia. Klasa ta zawiera wiele użytecznych w praktyce funkcji.

Twierdzenie 7. *Dla dowolnej funkcji całkowitej i dowolnego ciągu układów węzłów metoda interpolacji Lagrange'a wyznaczona przez ten ciąg jest zbieżna jednostajnie.*

Twierdzenie powyższe jest również spełnione dla klasy szerszej, do której należą funkcje rozwijalne w otoczeniu punktu $\frac{a+b}{2}$ w nieskończony szereg Taylora o promieniu zbieżności większym niż $\frac{e(b-a)}{2}$.

Kolejne twierdzenie dotyczy pytania 2.

Twierdzenie 8. *Dla dowolnej funkcji ciągłej istnieje ciąg układów węzłów, dla którego metoda interpolacji Lagrange'a jest zbieżna jednostajnie.*

Mimo odpowiedzi pozytywnej twierdzenie to ma niewielkie znaczenie praktyczne. Nie istnieje łatwa metoda wyznaczania ciągu układów, o którym mowa w twierdzeniu. Ponadto dla zastosowań ważne jest, by nie tylko uzyskać zbieżność, ale by zbieżność była dostatecznie szybka.

Pozostaje odpowiedź na pytanie pierwsze. Tym razem mamy daną zarówno funkcję f jak i ciąg układów węzłów \mathcal{X}_n . Odpowiedź nie jest jednoznaczna — dla pewnych funkcji (np. tych z Twierdzenia 7) można dla żądanej jakości zawsze znaleźć n , by ją zagwarantować. Są też ciągi układów i funkcje „nie-dobrane” do siebie i wtedy jakość może nawet się pogarszać wraz ze wzrostem n — tak jak przy zjawisku Rungego.

Ostatnie twierdzenie w tym punkcie jeszcze raz uwypukli zalety węzłów Czebyszewa. Przedstawione zostanie ono w postaci ogólnej — dla dowolnych węzłów, a wcześniej omówię używane tam pojęcie wielomianu optymalnego.

Na pierwszym roku analizy powinni Państwo spotkać się z twierdzeniem Weierstrassa:

Twierdzenie 9. (Weierstrassa) *Dla dowolnej funkcji ciągłej $f : [a, b] \rightarrow \mathbb{R}$ istnieje ciąg wielomianów takich, że zbiega on jednostajnie do f .*

⁸Funkcje całkowite prawdopodobnie nie są omawiane na kursie analizy matematycznej dla informatyków. Są to funkcje, które są nieskończonie wiele razy różniczkowalne, ale w sensie silniejszym, bo zespolonym.

Daje ono gwarancje, że wielomiany mogą służyć do dobrego przybliżania dowolnej funkcji ciągłej. Nie jest jednak podstawą do wyprowadzenia praktycznych metod znalezienia tych wielomianów. Poza tym zastosowania narzucają ograniczenia na dopuszczalny stopień wielomianów.

Można podejść do zagadnienia przybliżania funkcji przez wielomian w inny sposób. Ustalmy teraz funkcję f i przedział $[a, b]$. Wśród wszystkich wielomianów z przestrzeni Π_n będziemy szukać wielomianu, który najlepiej przybliża f w sensie normy supremum na przedziale $[a, b]$. Można udowodnić istnienie takiego wielomianu i są nawet praktyczne metody jego wyznaczenia (lub przynajmniej jego przybliżenia).⁹ Wielomian ten nazywamy *n-tym wielomianem optymalnym dla f* . Oznaczmy go przez w_n^* , zatem spełnia on

$$\|f - w_n^*\| = \inf_{v \in \Pi_n} \|f - v\|.$$

Ustalmy teraz węzły x_0, x_1, \dots, x_n i rozpatrzmy wielomian w_n z przestrzeni Π_n , który interpoluje f w tych węzłach. Występuje on oczywiście wśród wielomianów, po których jest liczone infimum, zatem

$$\|f - w_n\| \geq \|f - w_n^*\|.$$

Powstaje pytanie: jak dużo razy wielkość $\|f - w_n\|$ może być większa od $\|f - w_n^*\|$?

Zauważmy, że

$$f(x) - w_n(x) = (f(x) - w_n^*(x)) + (w_n^*(x) - w_n(x)).$$

Ponieważ w_n interpoluje f w ustalonych węzłach, a w_n^* interpoluje sam siebie, to

$$\begin{aligned} w_n(x) &= \sum_{i=0}^n f(x_i) l_i(x), \\ w_n^*(x) &= \sum_{i=0}^n w_n^*(x_i) l_i(x), \end{aligned}$$

stąd

$$\begin{aligned} |w_n^*(x) - w_n(x)| &= \left| \sum_{i=0}^n w_n^*(x_i) l_i(x) - \sum_{i=0}^n f(x_i) l_i(x) \right| \\ &= \left| \sum_{i=0}^n (w_n^*(x_i) - f(x_i)) l_i(x) \right| \\ &\leq \sum_{i=0}^n |w_n^*(x_i) - f(x_i)| \cdot |l_i(x)| \\ &\leq \sum_{i=0}^n \|w_n^* - f\| \cdot |l_i(x)| \\ &= \|w_n^* - f\| \sum_{i=0}^n |l_i(x)| \\ &= \|w_n^* - f\| \Lambda_n(x), \end{aligned}$$

⁹O tych metodach będzie mowa później, przy omawianiu metod aproksymacji, są one jednak kosztowne.

gdzie Λ_n jest tak zwaną *funkcją Lebesgue'a*

$$\Lambda_n(x) = \sum_{i=0}^n |l_i(x)|.$$

Otrzymujemy więc, że

$$\|w_n^* - w_n\| \leq \|w_n^* - f\| \cdot \|\Lambda_n\|$$

oraz

$$\begin{aligned} \|f - w_n\| &= \|(f - w_n^*) + (w_n^* - w_n)\| \\ &\leq \|f - w_n^*\| + \|w_n^* - w_n\| \\ &\leq \|f - w_n^*\| + \|w_n^* - f\| \cdot \|\Lambda_n\| \\ &= \|f - w_n^*\|(1 + \|\Lambda_n\|). \end{aligned}$$

Uzyskany wynik jest treścią ostatniego twierdzenia.

Twierdzenie 10. Niech $f : [a, b] \rightarrow \mathbb{R}$ będzie dowolną funkcją ciągłą, w_n wielomianem interpolującym f w węzłach x_0, x_1, \dots, x_n , a w_n^* n -tym wielomianem optymalnym dla f . Wówczas

$$\|f - w_n\| \leq \|f - w_n^*\|(1 + \|\Lambda_n\|),$$

gdzie Λ_n jest funkcją Lebesgue'a.

Wiemy zatem, że norma $\|f - w_n\|$ jest co najwyżej $(1 + \|\Lambda_n\|)$ razy większa niż $\|f - w_n^*\|$. Bez znajomości własności funkcji Λ_n trudno powiedzieć, czy to dużo, czy mało. Dokładniej funkcje Lebesgue'a zostaną przedstawione w następnym punkcie — między innymi nierówność zachodząca dla węzłów Czebyszewa:

$$\|\Lambda_n\| \leq \frac{2}{\pi} \ln n + 4.$$

Wynika z niej, że jeśli w_n interpoluje f w węzłach Czebyszewa, to

$$\|f - w_n\| \leq \|f - w_n^*\| \left(5 + \frac{2}{\pi} \ln n \right).$$

Zatem dla $n \leq 10$ wielomian interpolujący f w węzłach Czebyszewa jest co najwyżej 5.5 razy gorszy niż wielomian optymalny, dla $n \leq 100$ co najwyżej 8 razy gorszy, a dla $n \leq 1000$ co najwyżej 9.4 razy.¹⁰ Można powiedzieć, że jest prawie optymalny. Nie musi to wcale oznaczać zbieżności $\|f - w_n\|$ do zera, ale zobaczymy później, że założenie o posiadaniu przez funkcję jednej ciągłej pochodnej wystarczy, by zagwarantować zbieżność, choć niezbyt szybką.

8.8 Własności funkcji Lebesgue'a

Zacznijmy od narysowania przykładowych wykresów funkcji Lebesgue'a

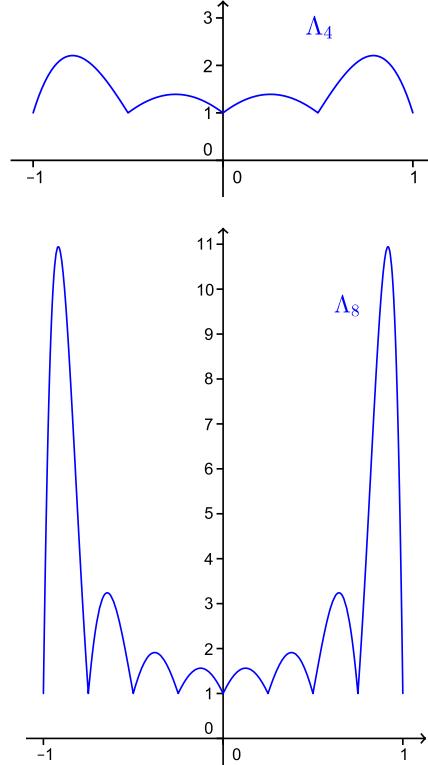
$$\Lambda_n(x) = \sum_{i=0}^n |l_i(x)|$$

¹⁰Autorzy wykładów z Metod Numerycznych na mediawiki.ilab.pl zamieszczają oszacowanie nieco lepsze:

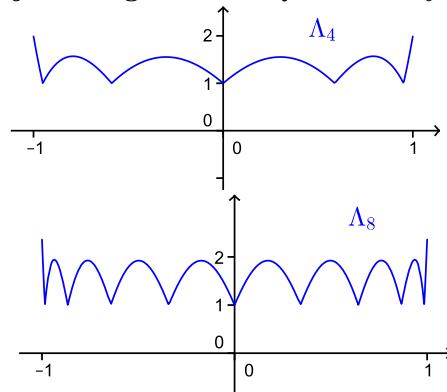
$$\|f - w_n\| \leq \|f - w_n^*\| \left(2 + \frac{2}{\pi} \ln(n+1) \right).$$

dla wybranych węzłów x_0, x_1, \dots, x_n z przedziału $[-1, 1]$.

Funkcje Lebesgue'a dla węzłów równoodległych



Funkcje Lebesgue'a dla węzłów Czebyszewa



Okazuje się, że przy węzłach równoodległych wartości funkcji szybko wzrastają wraz ze wzrostem n — dotyczy to głównie krańców przedziałów. Węzły Czebyszewa powodują dużo wolniejszy i znacznie równomierniejszy wzrost wartości. Poniższe twierdzenie potwierdza złe własności węzłów równoodległych i dobre Czebyszewa.

Twierdzenie 11. Dany jest przedział $[a, b]$ oraz węzły x_0, x_1, \dots, x_n należące do niego.

1. Jeśli węzły są równoodległe, to

$$\|\Lambda_n\| \geq Ce^{\frac{n}{2}},$$

gdzie C jest pewną stałą.

2. Dla węzłów Czebyszewa

$$\|\Lambda_n\| \leq \frac{2}{\pi} \ln n + 4.$$

8.9 Błędy interpolacji dla funkcji niekoniecznie posiadających $n+1$ ciągłych pochodnych

Niech $s \in \mathbb{N} \setminus \{0\}$. Oznaczmy przez $C[a, b]$ przestrzeń funkcji ciągłych, a przez $C^s[a, b]$ przestrzeń funkcji posiadających s ciągły pochodny na $[a, b]$. Wiemy, że

$$C[a, b] \subset C^1[a, b] \subset C^2[a, b] \subset C^3[a, b] \dots$$

Poniższe twierdzenie pozwala oszacować normę $\|f - w_n^*\|$ dla funkcji f , jeśli wiemy, że należy ona do przestrzeni $C^s[a, b]$. To z kolei umożliwia podanie pewnych ograniczeń górnych błędu interpolacji.

Twierdzenie 12. (Jacksona) Niech $s \in \{2, 3, \dots, n\}$ oraz w_n^* będzie n -tym wielomianem optymalnym dla f . Wtedy

$$\|f - w_n^*\| \leq \begin{cases} 6\omega(f, \frac{b-a}{2n}) & \text{dla } f \in C[a, b], \\ 3\frac{b-a}{n}\|f'\| & \text{dla } f \in C^1[a, b], \\ 6s\frac{(s-1)^{s-1}}{(s-1)!} \left(\frac{b-a}{n}\right)^s \|f^{(s)}\| & \text{dla } f \in C^s[a, b], \end{cases}$$

gdzie $\|\cdot\|$ oznacza normę supremum na $[a, b]$, a $\omega(f, h)$ tzw. moduł ciągłości funkcji określony wzorem

$$\omega(f, h) = \sup_{\substack{t \in (0, h) \\ [x, x+t] \subset [a, b]}} |f(x+t) - f(x)|.$$

Wykorzystamy obecnie Twierdzenie 10, by w przypadku funkcji niekoniecznie posiadających $n+1$ ciągły pochodny móc podać oszacowanie błędu interpolacji przy węzłach x_0, x_1, \dots, x_n .

Twierdzenie 13. Niech $f : [a, b] \rightarrow \mathbb{R}$ będzie dowolną funkcją ciągłą, a w_n wielomianem interpolującym f w węzłach x_0, x_1, \dots, x_n . Wówczas

1. jeśli $f \in C[a, b]$, to

$$R \leq 6\omega(f, \frac{b-a}{2n})(1 + \|\Lambda_n\|),$$

2. jeśli $f \in C^1[a, b]$, to

$$R \leq 3\frac{b-a}{n}\|f'\|(1 + \|\Lambda_n\|),$$

3. jeśli $s \in \{2, 3, \dots, n\}$ i $f \in C^s[a, b]$, to

$$R \leq 6s\frac{(s-1)^{s-1}}{(s-1)!} \left(\frac{b-a}{n}\right)^s \|f^{(s)}\|(1 + \|\Lambda_n\|).$$

Zauważmy, że gdy wiemy, że funkcja posiada przynajmniej jedną ciągłą pochodną oraz dokonamy interpolacji w $n+1$ węzłach Czebyszewa, to z punktu 2. mamy

$$R \leq 3(b-a)\|f'\| \frac{5 + \frac{2}{\pi} \ln n}{n}.$$

Ponieważ prawa strona nierówności dąży do zera przy $n \rightarrow \infty$, to metoda interpolacji Lagrange'a przy wyborze węzłów Czebyszewa jest zbieżna jednostajnie dla funkcji z $C^1[a, b]$.

8.10 Jeszcze raz o mierze skośności bazy Lagrange'a

Posiadając wiedzę o interpolacji Lagrange'a łatwo jest opisać miarę skośności dla bazy Lagrange'a w terminach funkcji Lebesguea. Zupełnie możliwe jest więc napisanie programu, który dla zadanych węzłów wielkość tę wyznaczy.

Przypomnijmy, że miara skośności bazy Lagrange'a wyraża się wzorem

$$C(\{l_i\}, [a, b], n) = \sup_{\substack{a_i \in \mathbb{R} \\ i=0, 1, \dots, n}} \frac{\|\sum_{i=0}^n |a_i| \cdot |l_i(x)|\|}{\|\sum_{i=0}^n a_i l_i(x)\|},$$

gdzie $\|\cdot\|$ oznacza normę supremum na odcinku $[a, b]$. Ponieważ liczymy supremum po wszystkich możliwych zestawach współczynników a_i , to wiemy, że będzie ono większe lub równe niż ułamek stojący po prawej stronie obliczony dla konkretnych współczynników, powiedzmy wszystkich równych 1:

$$C(\{l_i\}, [a, b], n) \geq \frac{\|\sum_{i=0}^n |l_i(x)|\|}{\|\sum_{i=0}^n l_i(x)\|}.$$

Można udowodnić, że dla dowolnego x zachodzi $\sum_{i=0}^n l_i(x) = 1$, zatem $\|\sum_{i=0}^n l_i(x)\| = 1$. Zauważmy, że $\sum_{i=0}^n |l_i(x)|$ to funkcja Lebesgue'a, którą oznaczamy przez Λ_n . Zatem

$$C(\{l_i\}, [a, b], n) \geq \|\Lambda_n\|.$$

Z drugiej strony dla dowolnych współczynników a_i wielomian $\sum_{i=0}^n a_i l_i(x)$ jest wielomianem interpolującym tabelkę

$$\begin{array}{c|c|c|c} x_0 & x_1 & \dots & x_n \\ \hline a_0 & a_1 & \dots & a_n \end{array},$$

więc w węźle x_i przyjmuje wartość a_i , stąd

$$\left\| \sum_{i=0}^n a_i l_i(x) \right\| \geq \sup_{i=0,1,\dots,n} |a_i|.$$

Ponadto dla dowolnego x mamy

$$\begin{aligned} \sum_{i=0}^n |a_i| \cdot |l_i(x)| &\leq \sum_{i=0}^n \left(\sup_{i=0,1,\dots,n} |a_i| \right) \cdot |l_i(x)| \\ &\leq \left(\sup_{i=0,1,\dots,n} |a_i| \right) \sum_{i=0}^n |l_i(x)| \\ &= \left(\sup_{i=0,1,\dots,n} |a_i| \right) \Lambda_n(x). \end{aligned}$$

Stąd

$$\left\| \sum_{i=0}^n |a_i| \cdot |l_i(x)| \right\| \leq \left(\sup_{i=0,1,\dots,n} |a_i| \right) \|\Lambda_n\|$$

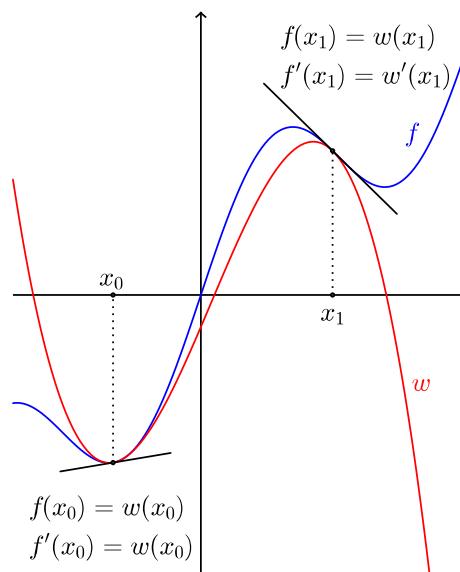
oraz

$$C(\{l_i\}, [a, b], n) \leq \|\Lambda_n\|.$$

Ostatecznie mamy, że

$$C(\{l_i\}, [a, b], n) = \|\Lambda_n\|.$$

Własności funkcji Lebesgue'a zostały przedstawione wcześniej. Algorytm, który oblicza wartości funkcji Λ_n można skonstruować na bazie algorytmu obliczającego wartości wielomianu zapisanego w bazie Lagrange'a — wystarczy w odpowiednich miejscach wstawić moduły. Obliczenie przybliżenia normy supremum może polegać obliczaniu wartości funkcji na gęstej siatce i wyborze największej z nich.



Podsumujmy sformułowanie zadania w tabeli.

Dane:	$x_0, k_0, f(x_0), f'(x_0), \dots, f^{(k_0-1)}(x_0)$	$x_1, k_1, f(x_1), f'(x_1), \dots, f^{(k_1-1)}(x_1)$	\dots	$x_m, k_m, f(x_m), f'(x_m), \dots, f^{(k_m-1)}(x_m)$
	\vdots	\vdots		\vdots
Wynik:	$w \in \Pi_n$ taki, że $n + 1 = \sum_{i=0}^n k_i$ oraz			
	$f^{(j)}(x_i) = w^{(j)}(x_i) \text{ dla } i = 0, 1, \dots, m$			
	$j = 0, \dots, k_i - 1$			

Można udowodnić, że rozwiązanie tego zadania istnieje i jest jednoznaczne.

Zauważmy, że jeśli wszystkie krotności wynoszą jeden, to zadanie sprowadza się do interpolacji Lagrange'a. W innym wypadku, wymagamy nie tylko zgodności wartości, lecz również pochodnych.

Do zapisu zadania interpolacji Hermite'a stosuje się też tabelę interpolacyjną. Zwykle ma ona postać

$$\begin{array}{c|c|c|c|c|c|c|c} x_0 & x_0 & \dots & x_0 & \dots & x_m & x_m & \dots & x_m \\ \hline f(x_0) & f'(x_0) & \dots & f^{(k_0-1)}(x_0) & \dots & f(x_m) & f'(x_m) & \dots & f^{(k_m-1)}(x_m) \end{array}$$

W pierwszym wierszu każdy węzeł występuje tyle razy, ile wynosi jego krotność. Z każdym jego wystąpieniem związana jest informacja o f w tym węźle — ma ona postać wartości kolejnej pochodnej w tym węźle (od „zerowej”, czyli zwykłej wartości funkcji, do $(k_i - 1)$ włącznie). Taka tabela pozwala łatwo napisać równania interpolacyjne — każda kolumna to jedno równanie określające równość odpowiednich pochodnych funkcji i wielomianu w odpowiednim węźle.

Jeśli krotności wszystkich węzłów są podobnej wielkości, to niektórzy zapisują tabelę interpolacyjną inaczej:

Zadanie polega na znalezieniu wielomianu w stopnia nie większego niż n takiego, że

$$\begin{aligned} w^{(j)}(x_i) &= f^{(j)}(x_i), \quad \text{dla } i = 0, \dots, m, \\ j &= 0, \dots, k_i - 1, \end{aligned}$$

przy czym wartości występujące po prawej stronie są dane. Zauważmy, że by to było możliwe f musi posiadać odpowiednio wiele pochodnych.

x_0	x_1	\dots	x_m
$f(x_0)$	$f(x_1)$	\dots	$f(x_m)$
$f'(x_0)$	$f'(x_1)$	\dots	$f'(x_m)$
\vdots	\vdots		\vdots
$f^{(k_0-2)}(x_0)$	$f^{(k_1-1)}(x_1)$	\dots	$f^{(k_m-2)}(x_m)$
$f^{(k_0-1)}(x_0)$	—	\dots	$f^{(k_m-1)}(x_m)$

Przykład Dane są dwa węzły $x_0 = 0$ i $x_2 = 2$, pierwszy dwukrotny, drugi jednokrotny. Zatem $m = 1$, $k_0 = 2$, $k_1 = 1$, $n = 2$. Ponadto znane są wartości

$$f(0) = 4, \quad f'(0) = -2, \quad f(2) = 0.$$

Zadanie to można zapisać przy pomocy jednej z tabel interpolacyjnych

0	0	2
4	-2	0

lub

0	2
4	0

—

Szukamy wielomianu $w \in \Pi_2$ spełniającego równania interpolacyjne

$$w(0) = 4, \quad w'(0) = -2, \quad w(2) = 0.$$

Niech $w(x) = ax^2 + bx + c$. Dostajemy układ równań

$$\begin{cases} c = 4, \\ b = -2, \\ 4a + 2b + c = 0, \end{cases}$$

którego rozwiązaniem jest $a = 0$, $b = -2$, $c = 4$. Zatem $w(x) = -2x + 4$.

9.2 Numeryczne rozwiązywanie zadania interpolacji Hermite'a

Sposób rozwiązania przedstawiony w poprzednim przykładzie (przedstawienie wielomianu w bazie naturalnej) nie jest używany w praktyce — jest kosztowny i ma złe własności numeryczne. Okazuje się, że można algorytm różnic dzielonych uogólnić tak, by rozwiązywał zadanie interpolacji Hermite'a.

Interesuje nas obecnie wyznaczenie współczynników wielomianu w w bazie Newtona określonej węzłami

$$\underbrace{x_0, \dots, x_0}_{k_0}, \underbrace{x_1, \dots, x_1}_{k_1}, \dots, \underbrace{x_{m-1}, \dots, x_{m-1}}_{k_{m-1}}, \underbrace{x_m, \dots, x_m}_{k_m-1}.$$

Proszę zwrócić uwagę, że węzeł x_m nie występuje k_m razy lecz tylko $k_m - 1$. Tabelę różnic dzielonych tworzymy podobnie,

jak w przypadku „zwykłym”.

x_0	$f(x_0)$	$\frac{f(x_0)-f(x_0)}{x_0-x_0} \quad \frac{f'(x_0)}{1!}$	$\frac{f''(x_0)}{2!}$
x_0	$f(x_0)$	$\frac{f(x_0)-f(x_0)}{x_0-x_0} \quad \frac{f'(x_0)}{1!}$	\vdots
x_0	$f(x_0)$	$\frac{f(x_1)-f(x_0)}{x_1-x_0}$	$\frac{f'(x_1)-f(x_0)}{1!} - \frac{f'(x_1)-f(x_0)}{x_1-x_0}$
x_1	$f(x_1)$	$\frac{f(x_1)-f(x_1)}{x_1-x_1} \quad \frac{f'(x_1)}{1!}$	$\frac{f''(x_1)}{2!}$
x_1	$f(x_1)$	\vdots	\vdots
x_1	$f(x_1)$	$\frac{f(x_2)-f(x_1)}{x_2-x_1}$	\vdots
\vdots	\vdots	\vdots	\vdots
x_m	$f(x_m)$	\vdots	\vdots
x_m	$f(x_m)$	$\frac{f(x_m)-f(x_m)}{x_m-x_m} \quad \frac{f'(x_m)}{1!}$	$\frac{f''(x_m)}{2!}$
x_m	$f(x_m)$	\vdots	

1. Wypisujemy kolumnę zawierającą wszystkie węzły z uwzględnieniem ich krotności.
2. W sąsiedniej kolumnie wpisujemy wartości funkcji obliczone dla stojących obok węzłów (jest to kolumna nr 0).
3. Obliczamy „zwykłe” różnice dzielone — tam gdzie się da, czyli nie wystąpi dzielenie przez 0.
4. Jeśli nie da się policzyć „zwykłej” różnicy dzielonej (występuje dzielenie przez 0), to wpisujemy wielkość $\frac{1}{k!} f^{(k)}(x_i)$, gdzie
 - (a) k jest numerem kolumny (kolumny z węzłami nie numerujemy, a dalej numerujemy od 0),
 - (b) x_i jest tym węzłem, który występowałby w mianowniku, gdybyśmy liczyli „zwykłą” różnicę dzieloną.

Współczynniki szukanego wielomianu w bazie Newtona określonej węzłami

$$\underbrace{x_0, \dots, x_0}_{k_0}, \underbrace{x_1, \dots, x_1}_{k_1}, \dots, \underbrace{x_{m-1}, \dots, x_{m-1}}_{k_{m-1}}, \underbrace{x_m, \dots, x_m}_{k_m-1}$$

znajdują się w pierwszym („ukośnym”) wierszu tabeli różnic dzielonych. O ile węzły są uporządkowane, to algorytm jest prawie numerycznie poprawny. Zaprogramowanie algorytmu w przypadkach szczególnych (np. gdy wszystkie węzły mają tę samą krotność) oraz ogólnym pozostawiamy jako ćwiczenie.

Przykład Zastosujmy algorytm różnic dzielonych do rozwiązania zadania z poprzedniego przykładu. Szukamy współ-

czynników b_0, b_1, b_2 wielomianu w w bazie $1, x, x^2$.

$$\begin{array}{c|ccc} & 4 \\ 0 & -2 \\ \hline 0 & 4 & \frac{-2+2}{2-0} \\ & \frac{0-4}{2-0} \\ \hline 2 & 0 \end{array}$$

Stąd $b_0 = 4, b_1 = -2, b_2 = 0$, czyli $w(x) = 4 - 2x$.

9.3 Oszacowanie reszty interpolacyjnej

Pojęcia reszty i błędu interpolacji definiuje się tak jak po przednio:

$$r(x) = f(x) - w(x),$$

$$R = \sup_{x \in [a,b]} |f(x) - w(x)| = \|r\|,$$

gdzie w jest wielomianem Hermite'a interpolującym f w węzłach z przedziału $[a, b]$.

Twierdzenia dotyczące reszty przy interpolacji Hermite'a są uogólnieniem tych, które dotyczą interpolacji Lagrange'a.

Twierdzenie 14. *Dany jest przedział $[a, b]$ zawierający wszystkie węzły x_0, x_1, \dots, x_m oraz punkt x . Niech rzeczywista funkcja f ma $n + 1$ ciągły pochodnych na $[a, b]$, gdzie $n = \sum_{i=0}^m k_i - 1$. Wtedy istnieje punkt $\xi(x) \in [a, b]$ zależny od x taki, że*

$$r(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \prod_{i=0}^m (x - x_i)^{k_i}.$$

Jeśli wprowadzimy oznaczenie

$$\tilde{N}_{n+1}(x) = \prod_{i=0}^m (x - x_i)^{k_i},$$

to możemy zapisać wnioski z powyższego twierdzenia, które wyglądają niemal identycznie jak w przypadku interpolacji Lagrange'a.

Wniosek 5. *Dany jest przedział $[a, b]$ zawierający wszystkie węzły x_0, x_1, \dots, x_m oraz punkt x . Niech rzeczywista funkcja f ma $n + 1$ ciągły pochodnych na przedziale $[a, b]$, gdzie $n = \sum_{i=0}^m k_i - 1$. Wtedy*

$$|r(x)| \leq \frac{\|f^{(n+1)}\|}{(n+1)!} |\tilde{N}_{n+1}(x)|.$$

Wniosek 6. *Dany jest przedział $[a, b]$ zawierający wszystkie węzły x_0, x_1, \dots, x_m . Niech rzeczywista funkcja f ma $n + 1$ ciągły pochodnych na przedziale $[a, b]$, gdzie $n = \sum_{i=0}^m k_i - 1$. Wtedy*

$$R \leq \frac{\|f^{(n+1)}\|}{(n+1)!} \|\tilde{N}_{n+1}\|.$$

Problem doboru optymalnych węzłów ze względu na powyższe oszacowanie błędu jest obecnie bardziej skomplikowany.

Można go łatwo rozwiązać, gdy np. wszystkie węzły są tej samej krotności:

$$k_0 = k_1 = \dots = k_m = k.$$

Wtedy

$$\tilde{N}_{n+1}(x) = \prod_{i=0}^m (x - x_i)^k = \left(\prod_{i=0}^m (x - x_i) \right)^k = (N_{m+1}(x))^k,$$

a zatem

$$\|\tilde{N}_{n+1}\| = \|N_{m+1}\|^k.$$

Jak było stwierdzono przy interpolacji Lagrange'a, norma po prawej stronie będzie najmniejsza, gdy x_0, x_1, \dots, x_m będą węzłami Czebyszewa w $[a, b]$. Wtedy

$$\|N_{m+1}\| = \frac{1}{2^m} \left(\frac{b-a}{2} \right)^{m+1}.$$

Przy takim wyborze węzłów błąd interpolacji Hermite'a można oszacować

$$\begin{aligned} R &\leq \frac{\|f^{(n+1)}\|}{(n+1)!} \|\tilde{N}_{n+1}\| \\ &= \frac{\|f^{(n+1)}\|}{(n+1)!} \left(\frac{1}{2^m} \left(\frac{b-a}{2} \right)^{m+1} \right)^k \\ &= \frac{\|f^{(n+1)}\|}{(n+1)!} \frac{1}{2^{mk}} \left(\frac{b-a}{2} \right)^{(m+1)k} \\ &= \frac{\|f^{(n+1)}\|}{(n+1)!} \frac{1}{2^{n+1-k}} \left(\frac{b-a}{2} \right)^{n+1} \end{aligned}$$

Powyższe spostrzeżenia pozwalają sformułować następujący wniosek.

Wniosek 7. *Dana jest rzeczywista funkcja f oraz przedział $[a, b]$. Dokonujemy interpolacji Hermite'a f w x_0, x_1, \dots, x_m , które są węzłami Czebyszewa z $[a, b]$ oraz ich krotności są równe i wynoszą k . Niech $n = (m+1)k - 1$ i założymy, że f ma $n+1$ ciągły pochodnych na $[a, b]$. Wtedy*

$$R \leq \frac{\|f^{(n+1)}\|}{2^{n+1-k}(n+1)!} \left(\frac{b-a}{2} \right)^{n+1}.$$

Przykład Dana jest funkcja $f(x) = \sin x$. Interpolujemy ją na przedziale $[-\frac{\pi}{2}, \frac{\pi}{2}]$ w dwóch podwójnych (czyli o krotności równej 2) węzłach $x_0 = -\frac{\pi}{2}, x_1 = \frac{\pi}{2}$. Oszacujmy błąd interpolacji, w tym celu posłużmy się Wnioskiem 6.

Zauważmy, że dla dowolnego n mamy tu $\|f^{(n+1)}\| = 1$. Pozostało policzyć normę supremum funkcji

$$\tilde{N}_4(x) = \left(x + \frac{\pi}{2} \right)^2 \left(x - \frac{\pi}{2} \right)^2 = \left[\left(x + \frac{\pi}{2} \right) \left(x - \frac{\pi}{2} \right) \right]^2.$$

Zauważmy, że

$$\|\tilde{N}_4\| = \|N_2\|^2,$$

gdzie

$$N_2(x) = \left(x + \frac{\pi}{2} \right) \left(x - \frac{\pi}{2} \right).$$

Ponieważ wykresem N_2 jest parabola, to jej ekstremum lokalne znajduje się w wierzchołku $x = 0$ i wynosi $-\frac{\pi^2}{4}$. W punktach brzegowych przedziału funkcja N_2 ma wartość zero, zatem

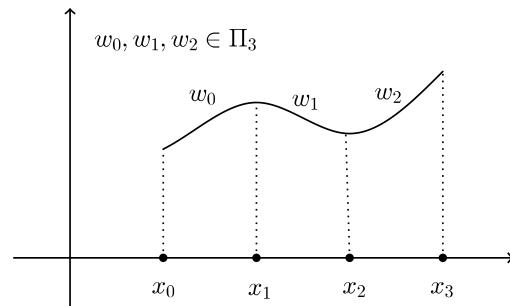
$$\|N_2\| = \frac{\pi^2}{4}.$$

Otrzymujemy

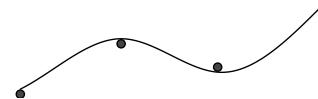
$$R \leq \frac{1}{4!} \frac{\pi^2}{4} = \frac{\pi^2}{96} \approx 0.103.$$

Gdybyśmy dokonali interpolacji w dwóch podwójnych węzłach Czebyszewa w $[-\frac{\pi}{2}, \frac{\pi}{2}]$ to z Wniosku 7 otrzymalibyśmy

$$R \leq \frac{1}{2^{24}!} \left(\frac{\pi}{2}\right)^4 = \frac{\pi^4}{1536} \approx 0.0634.$$



Okazuje się, że występują w naturze funkcje będące splajnami kubicznymi. Gdybyśmy w deskę wbili pewną liczbę gwoździ i pomiędzy je włożyły elastyczną listwę to odkształci się ona zgodnie z prawie Hooke'a tak, by zminimalizować energię potencjalną sprężystości. Przyjmie ona kształt właśnie splajnu kubicznego.



10 Splajny, algorytmy obliczania wartości splajnów w punkcie

Splajny są funkcjami będącymi „sklejeniem” wielomianów. Wymagane jest, by sklejenie zostało wykonane w sposób gładki, czyli z zachowaniem ciągłości odpowiednio wielu pochodnych. W zależności od maksymalnego stopnia wielomianu oraz liczby ciągłych pochodnych mówimy o rzędzie splajnu. Zacznę od przedstawienia splajnów trzeciego rzędu, które nazywane są zwykle splajnami kubicznymi i mają najwięcej zastosowań.

10.1 Definicja splajnu kubicznego

By móc zdefiniować splajn niezbędne jest podanie węzłów, które muszą być uporządkowane.

Definicja 6. *Splajnem kubicznym opartym na węzłach*

$$a = x_0 < x_1 < \dots < x_n = b$$

nazywamy funkcję $S : [a, b] \rightarrow \mathbb{R}$ taką, że

1. S zawężona do przedziału $[x_i, x_{i+1}]$ dla $i = 0, \dots, n-1$ jest wielomianem stopnia co najwyżej trzeciego,
2. S jest funkcją klasy $C^2[a, b]$.

Zakładamy, że $n \in \mathbb{N} \setminus \{0\}$. Na każdym przedziale wyznaczonym przez sąsiednie węzły splajn może być innym wielomianem stopnia co najwyżej trzeciego. Przypomnijmy, że druga własność jest skróconą formą zapisu, że w każdym punkcie przedziału $[a, b]$ splajn S posiada dwie ciągle pochodne. Zwróćmy uwagę, że dziedziną S jest $[a, b]$.

Nazwa splajn pochodzi od ang. *spline*, co oznacza giętką listwę używaną przez budowniczych statków do wyznaczania konturów poszycia łodzi. W języku polskim listwę tą określa się rzeczownikiem *giętka*. Można w literaturze spotkać się z nazwami *funkcje giętkie* lub *funkcje sklejane* dla określenia splajnów.

Przykład Funkcja

$$S_1(x) = \begin{cases} x^2 - 2x & \text{dla } x \geq 0, \\ x^3 + x^2 - 2x & \text{dla } x < 0 \end{cases}$$

jest splajnem kubicznym dla węzłów $x_0 = -3, x_1 = 0, x_2 = 2$. Wystarczy sprawdzić, że S_1 posiada dwie ciągle pochodne w punkcie 0.

Funkcje $S_2(x) = x, S_3(x) = x^2, S_4(x) = x^3$ są splajnami kubicznymi przy dowolnym wyborze węzłów.

Funkcja $f(x) = x^4$ nie jest splajnem kubicznym dla żadnego układu węzłów.

10.2 Definicja splajnu k -tego rzędu

Definicję splajnu kubicznego można uogólnić tak, by dopuścić wielomiany innych stopni.

Definicja 7. *Splajnem k -tego rzędu opartym na węzłach*

$$a = x_0 < x_1 < \dots < x_n = b$$

nazywamy funkcję $S : [a, b] \rightarrow \mathbb{R}$ taką, że

1. S zawężona do przedziału $[x_i, x_{i+1}]$ dla $i = 0, \dots, n-1$ jest wielomianem stopnia co najwyżej k -tego,
2. S jest funkcją klasy $C^{k-1}[a, b]$.

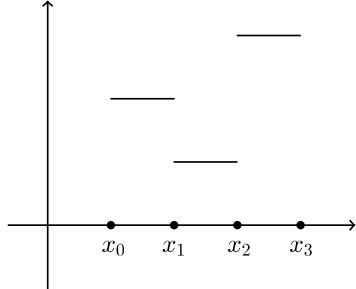
W definicji tej przyjmuje się umowę, że klasa $C^0[a, b]$ zawiera wszystkie funkcje ciągłe na $[a, b]$, klasa $C^{-1}[a, b]$ do-

wolne funkcje o dziedzinie $[a, b]$. Zauważmy, że dla $k = 3$ otrzymujemy splajny kubiczne.

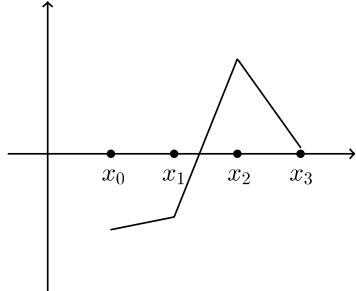
Przykład Funkcje S_2 , S_3 i S_4 z poprzedniego przykładu są splajnami trzeciego, czwartego, piątego itd. rzędu. Funkcja S_2 jest splajnem pierwszego rzędu.

Przykład Oto przykładowe wykresy splajnów zerowego, pierwszego i drugiego rzędu.

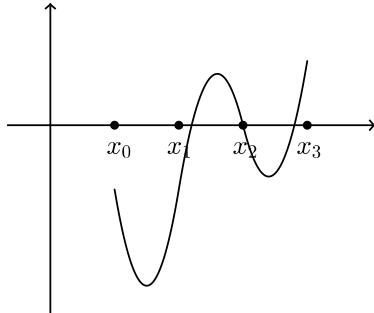
Splajn zerowego rzędu oparty na węzłach x_0, x_1, x_2, x_3



Splajn pierwszego rzędu oparty na węzłach x_0, x_1, x_2, x_3



Splajn drugiego rzędu oparty na węzłach x_0, x_1, x_2, x_3



Przykład Dane są węzły $x_0 = 1, x_1 = 2, x_2 = 3$. Funkcja $S_1(x) = x^2$ jest splajnem drugiego rzędu opartym na tych węzłach, ale nie jest splajnem pierwszego rzędu. Funkcja

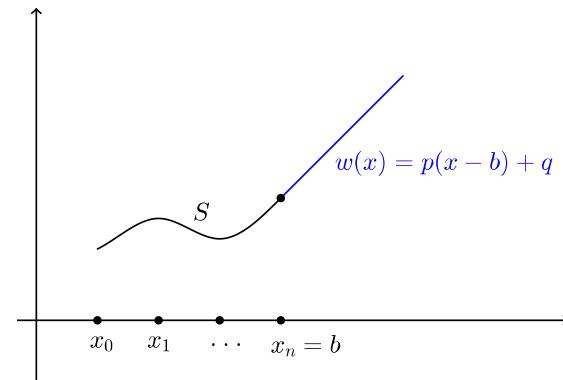
$$S_2(x) = \begin{cases} x - 3 & \text{dla } x \in [1, 2], \\ -1 & \text{dla } x \in [2, 3] \end{cases}$$

jest splajnem pierwszego rzędu opartym na tych węzłach, ale nie jest splajnem drugiego rzędu.

10.3 Splajny kubiczne prawo- i lewostronne swobodne oraz okresowe

Czasami istnieje potrzeba rozszerzenia dziedziny splajnu kubicznego poza przedział $[a, b]$ z zachowaniem ciągłości obu pochodnych. Można to zrobić na kilka sposobów.

Po pierwsze, można z prawej strony kontynuować splajn S linią prostą.



Aby to uzyskać z zachowaniem własności, że „przedłużony” splajn jest klasy $C^2[a, +\infty]$, musi być

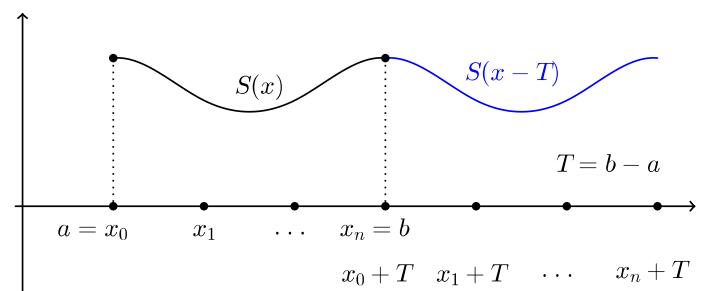
$$\begin{aligned} S(b) &= w(b) = q, \\ S'(b) &= w'(b) = p, \\ S''(b) &= w''(b) = 0. \end{aligned}$$

Warunki te są możliwe do spełnienia wtedy i tylko wtedy, gdy $S''(b) = 0$. Analogicznie można rozważyć przedłużenie splajnu z lewej strony przedziału $[a, b]$ lub z obu stron.

Definicja 8.

1. Splajn kubiczny S spełniający warunek $S''(b) = 0$ nazywamy *prawostronne swobodnym*.
2. Splajn kubiczny S spełniający warunek $S''(a) = 0$ nazywamy *lewostronne swobodnym*.
3. Splajn kubiczny, który jest równocześnie prawo- i lewostronne swobodny nazywamy *obustronne swobodnym* lub *naturalnym*.

Innym sposobem jest rozszerzenie okresowe.



Okresem T jest tu długość przedziału $[a, b]$. Aby tak utworzona funkcja posiadała dwie ciągle pochodne na całej prostej \mathbb{R} istotnym jest, by dobrze skleić splajn S i jego przesunięcie w punkcie b . Musi być zatem spełnione, że

$$\begin{aligned} S(a) &= S(b), \\ S'(a) &= S'(b), \\ S''(a) &= S''(b). \end{aligned}$$

Definicja 9.

Splajn kubiczny S nazywamy *okresowym* jeśli

$$\begin{aligned} S(a) &= S(b), \\ S'(a) &= S'(b), \\ S''(a) &= S''(b). \end{aligned}$$

Zwróćmy uwagę, że w definicjach splajnów lewo- i prawostronnych oraz naturalnych i okresowych nie jest wspomniane o zmianie dziedziny, zatem nadal jest nią przedział $[a, b]$. Gwarantowana jest jedynie możliwość rozszerzenia dziedziny, tak by powstała funkcja, która ma dwie ciągle pochodne.

10.4 Przestrzeń liniowa splajnów

Okazuje się, że wszystkie splajny k -tego rzędu oparte na węzłach $x_0 < x_1 < \dots < x_n$ tworzą przestrzeń liniową. Dowodzi się, że ma ona wymiar $n + k$. Można wybrać bazę tej przestrzeni i zrobić to na nieskończoność wiele sposobów. Jeden z nich okazał się szczególnie użyteczny w praktyce — jest to tzw. baza B -splajnów, o których mowa w następnym punkcie.

10.5 B-splajny

Rozważmy przestrzeń splajnów k -tego rzędu opartych na węzłach $x_0 < x_1 < \dots < x_n$. Węzły te będą obecnie nazywać *siatką podstawową*. W celu konstrukcji bazy rozszerzę siatkę o k węzłów w prawo i w lewo:

$$x_{-k} < x_{-k+1} \dots < x_0 < x_1 < \dots < x_n < x_{n+1} < \dots < x_{n+k},$$

będę ją nazywać *siatką rozszerzoną*.

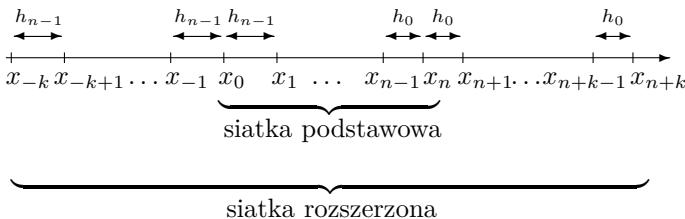
Dodanie węzłów może odbywać się dowolnie, zwykle jednak przyjmuje się, że węzły

$$x_{-k} < x_{-k+1} \dots < x_0$$

są rozmieszczone równoodlegle z krokiem $h_0 = x_1 - x_0$, a węzły

$$x_n < x_{n+1} < \dots < x_{n+k}$$

też równoodlegle, ale z krokiem $h_{n-1} = x_n - x_{n-1}$.



Węzły dodane pełnią rolę pomocniczą. Nadal przyjmujemy, że dziedziną splajnu jest przedział $[a, b] = [x_0, x_n]$.

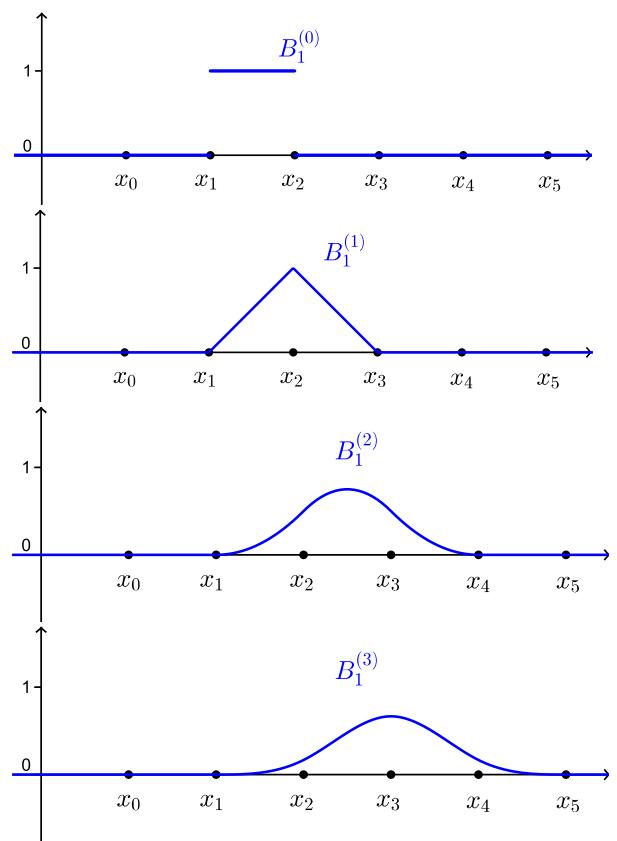
Definicja 10. B -splajnami k -tego rzędu nazywamy funkcje $B_i^{(k)}$, które są zdefiniowane rekurencyjnie (ze względu na rząd):

$$B_i^{(0)}(x) = \begin{cases} 1 & \text{dla } x \in [x_i, x_{i+1}), \\ 0 & \text{w pp.} \end{cases}$$

$$B_i^{(p)}(x) = \frac{x-x_i}{x_{i+p}-x_i} B_i^{(p-1)}(x) + \frac{x_{i+p+1}-x}{x_{i+p+1}-x_{i+1}} B_{i+1}^{(p-1)}(x),$$

dla $p = 1, 2, \dots, k$.

Oto przykładowe wykresy B-splajnów.



Funkcje $B_i^{(k)}$ posiadają szereg własności.

1. Dla dowolnego x zachodzi nierówność $0 \leq B_i^{(k)}(x) \leq 1$.
2. Nośniakiem¹¹ funkcji $B_i^{(k)}$ jest przedział $[x_i, x_{i+k+1}]$.
3. Dla dowolnego $x \in [a, b]$ zachodzi

$$\sum_{i=-k}^{n-1} B_i^{(k)}(x) = 1.$$

Własność ta zachodzi również dla $x = b$, o ile $k \geq 1$.

4. Funkcje $B_{-k}^{(k)}, B_{-k+1}^{(k)}, \dots, B_{n-2}^{(k)}, B_{n-1}^{(k)}$, zawężone do przedziału $[a, b]$ są splajnami k -tego rzędu opartymi na węzłach x_0, x_1, \dots, x_n .
5. Funkcje $B_{-k}^{(k)}, B_{-k+1}^{(k)}, \dots, B_{n-2}^{(k)}, B_{n-1}^{(k)}$, zawężone do przedziału $[a, b]$ tworzą bazę przestrzeni splajnów k -tego rzędu opartych na węzłach x_0, x_1, \dots, x_n .

Ostatnia własność daje opis szukanej bazy przestrzeni splajnów k -tego rzędu opartych na węzłach x_0, x_1, \dots, x_n .

Przykład Niech $n = 2$ i siatkę podstawową tworzą węzły

$$x_i = i, \quad i = 0, 1, 2.$$

¹¹Dana jest funkcja f . Niech A oznacza zbiór tych wszystkich x , dla których $f(x) \neq 0$. Nośniakiem funkcji f jest domknięcie zbioru A .

Wyznaczamy bazę B-splajnów przestrzeni splajnów drugiego rzędu opartych na węzłach x_0, x_1, x_2 . Przestrzeń ta ma wymiar $n + k = 4$, funkcjami bazowymi są

$$B_{-2}^{(2)}, B_{-1}^{(2)}, B_0^{(2)}, B_1^{(2)}.$$

By móc je zdefiniować tworzymy najpierw siatkę rozszerzoną

$$x_i = i, \quad i = -2, 1, 0, 1, 2, 3, 4.$$

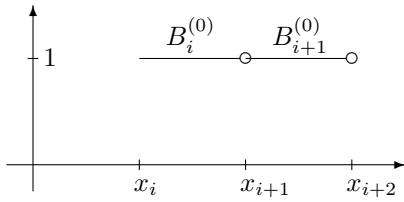
Po kolei definiujemy B-splajny zerowego, pierwszego i drugiego rzędu. Splajny zerowego rzędu to funkcje

$$B_i^{(0)}(x) = \begin{cases} 1 & \text{dla } x \in [i, i+1], \\ 0 & \text{w pp.} \end{cases}$$

Splajny pierwszego rzędu wyznaczamy z zależności rekurencyjnej

$$\begin{aligned} B_i^{(1)}(x) &= \frac{x-x_i}{x_{i+1}-x_i} B_i^{(0)}(x) + \frac{x_{i+2}-x}{x_{i+2}-x_{i+1}} B_{i+1}^{(0)}(x) \\ &= (x-i) B_i^{(0)}(x) + (i+2-x) B_{i+1}^{(0)}(x) \end{aligned}$$

Nośnikiem funkcji $B_i^{(0)}$ jest przedział $[x_i, x_{i+1}]$, zaś $B_{i+1}^{(0)}$ przedział $[x_{i+1}, x_{i+2}]$.



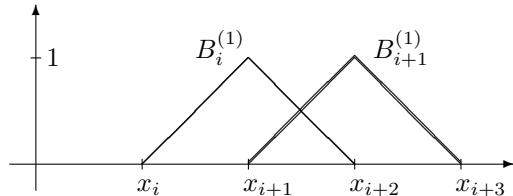
W przedziale $[x_i, x_{i+1}]$ funkcja $B_{i+1}^{(0)}$ przyjmuje wartość zero, a $B_i^{(0)}$ wartość jeden. Odwrotnie jest w przedziale $[x_{i+1}, x_{i+2}]$. Poza tymi przedziałami obie funkcje się zerują. Podsumowując

$$B_i^{(1)}(x) = \begin{cases} x-i & \text{dla } x \in [i, i+1], \\ i+2-x & \text{dla } x \in [i+1, i+2], \\ 0 & \text{w pp.} \end{cases}$$

Aby wyznaczyć splajny drugiego rzędu ponownie korzystamy z zależności rekurencyjnej

$$\begin{aligned} B_i^{(2)}(x) &= \frac{x-x_i}{x_{i+2}-x_i} B_i^{(1)}(x) + \frac{x_{i+3}-x}{x_{i+3}-x_{i+1}} B_{i+1}^{(1)}(x) \\ &= \frac{x-i}{2} B_i^{(1)}(x) + \frac{i+3-x}{2} B_{i+1}^{(1)}(x) \end{aligned}$$

Tym razem trzeba rozważyć osobno przedziały $[x_i, x_{i+1}]$, $[x_{i+1}, x_{i+2}]$, $[x_{i+2}, x_{i+3}]$ i pozostałą część osi.



Na podstawie wzoru opisującego $B_i^{(0)}$ mamy

$$B_{i+1}^{(1)}(x) = \begin{cases} x-i-1 & \text{dla } x \in [i+1, i+2], \\ i+3-x & \text{dla } x \in [i+2, i+3], \\ 0 & \text{w pp.} \end{cases}$$

Otrzymujemy zatem

$$B_i^{(2)}(x) = \begin{cases} \frac{(x-i)^2}{2} & x \in [i, i+1], \\ \frac{(x-i)(i+2-x)}{2} + \frac{(i+3-x)(x-i-1)}{2} & x \in [i+1, i+2], \\ \frac{(x-i-3)^2}{2} & x \in [i+2, i+3], \\ 0 & \text{w pp.} \end{cases}$$

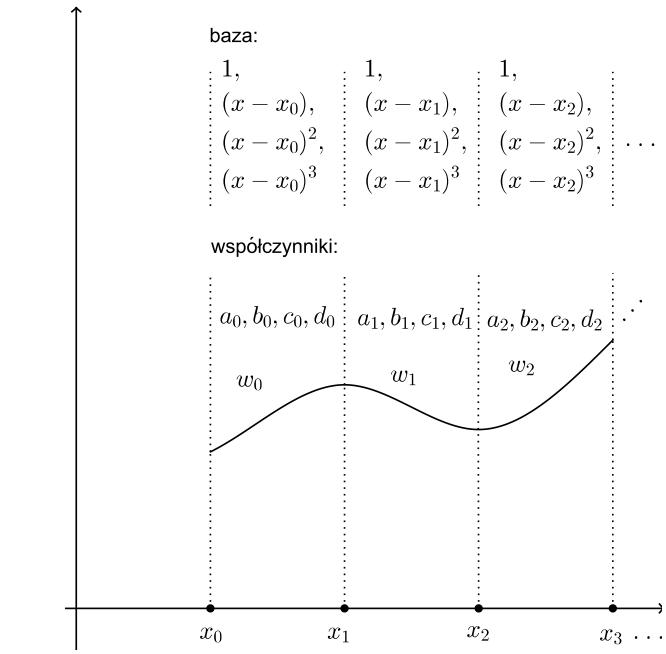
Funkcje te dla $i = -2, -1, 0, 1$ tworzą szukaną bazę, o ile zawężymy ich dziedzinę do przedziału $[a, b] = [0, 2]$.

10.6 Przechowywanie splajnów

W praktyce obliczeniowej stosuje się dwa podstawowe sposoby przechowywania splajnów. Pierwszy polega na zapamiętaniu wzorów wielomianów na każdym z podprzedziałów. Drugi wykorzystuje fakt, że B-splajny tworzą bazę — przechowuje się współczynniki splajnu w tej bazie. Pierwszy, aby uprościć zapis, przedstawię tylko dla splajnów kubicznych. Łatwo go uogólnić na dowolny rzad.

10.6.1 Przechowywanie splajnów kubicznych w lokalnych bazach potęgowych

W każdym z przedziałów $[x_i, x_{i+1}]$ splajn jest wielomianem stopnia co najwyżej trzeciego. Można więc zapamiętać cztery współczynniki dla każdego przedziału osobno, uprzednio wybrawszy bazę wielomianową. Nie musi być ona jednakowa dla każdego przedziału. Najtańsze są bazy potęgowe, lecz charakteryzuje się dużą skośnością. Na szczęście tu mamy do czynienia z wielomianami tylko trzeciego stopnia i tzw. lokalne bazy potęgowe są wystarczająco dobre. Będziemy w przedziale $[x_i, x_{i+1}]$ zapamiętywać współczynniki w bazie $1, x - x_i, (x - x_i)^2, (x - x_i)^3$.¹²



¹²Jest to faktycznie baza Newtona o węzłach x_i, x_i, x_i .

Zatem dla $x \in [x_i, x_{i+1}]$ splajn S ma postać¹³

$$S(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3.$$

Pełna informacja o splajnie składa się z $4n$ liczb

$$\begin{array}{cccc} a_0 & b_0 & c_0 & d_0 \\ a_1 & b_1 & c_1 & d_1 \\ \vdots & \vdots & \vdots & \vdots \\ a_{n-1} & b_{n-1} & c_{n-1} & d_{n-1} \end{array}$$

na przechowanie współczynników oraz $n + 1$ liczb

$$x_0, x_1, \dots, x_n$$

do zapamiętania węzłów. Jeśli węzły są równoodległe, czyli dla pewnego $h > 0$ mamy $x_i = a + ih$, $i = 0, \dots, n$, to wystarczy zapamiętać x_0 oraz h . Zatem

$$\text{koszt pamięciowy} = \begin{cases} 4n + 2 & \text{dla węzłów równoodległych,} \\ 5n + 1 & \text{w p.p..} \end{cases}$$

Reprezentacja powyższa pozwala łatwo wykonywać działania na splajnach — dodawać, mnożyć przez liczbę, obliczać wartość w punkcie.

Ma ona też wady. Przede wszystkim, jeśli przypiszemy współczynnikom a_i, b_i, c_i, d_i dowolne liczby, to tak powstała funkcja zwykle nie będzie splajnem kubicznym. Nie zostaną bowiem spełnione warunki ciągłości — te współczynniki nie są niezależne. Jeśli nawet zadbane, by dane były poprawne, to ponieważ arytmetyka zmienopozycyjna wprowadza błędy zaokrągleń, to niemal zawsze będziemy mieć do czynienia z funkcją, która nie jest splajnem, ale „prawie” splajnem — jest „prawie” ciągła i ma „prawie” ciągle dwie pochodne.

Drugą wadą — jak się okaże później — jest stosunkowo wysoki koszt pamięciowy.

Przykład Niech $x_0 = 0, x_1 = 1, x_2 = 3$ i $S(x) = (x - 1)^2$. Przedstawmy splajn S w lokalnych bazach potęgowych. Mamy do czynienia z dwoma przedziałami $[x_0, x_1] = [0, 1]$ oraz $[x_1, x_2] = [1, 3]$. W każdym z nich splajn będzie przedstawiony w innej bazie, w pierwszym będzie to baza $1, x, x^2, x^3$, w drugim $1, x - 1, (x - 1)^2, (x - 1)^3$. Zatem

$$S(x) = \begin{cases} 1 - 2x + x^2 + 0x^3, & x \in [0, 1], \\ 0 + 0(x - 1) + 1(x - 1)^2 + 0(x - 1)^3, & x \in [1, 3]. \end{cases}$$

Informacja o splajnie składa się z 11 liczb

$$\begin{aligned} \text{węzły : } & x_0 = 0, \quad x_1 = 1, \quad x_2 = 3, \\ \text{współczynniki } & \\ \text{w } [0, 1] : & a_0 = 1, \quad b_0 = -2, \quad c_0 = 1, \quad d_0 = 0, \\ \text{w } [1, 3] : & a_1 = 0, \quad b_1 = 0, \quad c_1 = 1, \quad d_1 = 0. \end{aligned}$$

10.6.2 Reprezentacja splajnów w bazie B-splajnów

Ustalmy węzły $x_0 < x_1 < \dots < x_n$. Ponieważ funkcje

$$B_i^{(k)} \text{ dla } i = -k, -k + 1, \dots, n - 1$$

¹³Ze względu na ciągłość S przedział może być obustronnie domknięty.

tworzą bazę przestrzeni splajnów rzędu k opartych na tych węzłach, to każdy splajn S z tej przestrzeni można jednoznacznie przedstawić w tej bazie

$$S(x) = \sum_{i=-k}^{n-1} b_i B_i^{(k)}(x).$$

Zatem w celu przechowania S wystarczy zapamiętać $n + k$ liczb $b_{-k}, b_{-k+1}, \dots, b_{n-1}$ oraz węzły,

$$\text{koszt pamięciowy} = \begin{cases} n + k + 2 & \text{dla węzłów równoodl.,} \\ 2n + k + 1 & \text{w p.p..} \end{cases}$$

Dla splajnów kubicznych mamy

$$\text{koszt pamięciowy} = \begin{cases} n + 5 & \text{dla węzłów równoodległych,} \\ 2n + 4 & \text{w p.p.,} \end{cases}$$

czyli niemal czterokrotnie taniej niż poprzednio w przypadku węzłów równoodległych i 2.5– krotnie przy węzłach innych (dla dużych n).

Obecnie każdy zestaw danych gwarantuje, że mamy do czynienia ze splajnem. Zmiana współczynników na skutek błędów zaokrągleń powoduje, że otrzymamy trochę inny splajn („sąsiadni”), ale nadal splajn.

Przykład Rozpatrzmy węzły $x_0 = 0, x_1 = 1, x_2 = 2$ oraz splajn $S(x) = (x - 1)^2$. Jest to splajn drugiego rzędu, zapiszmy go w bazie $B_2^{(2)}, B_1^{(2)}, B_0^{(2)}, B_1^{(2)}$. W jednym z poprzednich przykładów obliczyliśmy

$$B_i^{(2)}(x) = \begin{cases} \frac{(x-i)^2}{2} & x \in [i, i+1], \\ \frac{(x-i)(i+2-x)}{2} + \frac{(i+3-x)(x-i-1)}{2} & x \in [i+1, i+2], \\ \frac{(x-i-3)^2}{2} & x \in [i+2, i+3], \\ 0 & \text{w pp.} \end{cases}$$

gdzie $i = -2, -1, 0, 1$. Szukamy współczynników b_{-2}, b_{-1}, b_0 i b_1 takich, że

$$(x - 1)^2 = b_{-2} B_{-2}^{(2)}(x) + b_{-1} B_{-1}^{(2)}(x) + b_0 B_0^{(2)}(x) + b_1 B_1^{(2)}(x).$$

Rozbijmy zadanie na podprzedziały $[0, 1]$ i $[1, 2]$, których suma daje dziedzinę splajnu. Dla $x \in [0, 1]$ powyższa równość przyjmuje postać

$$(x - 1)^2 = b_{-2} \frac{(x-1)^2}{2} + b_{-1} \frac{(x+1)(1-x)+(2-x)x}{2} + b_0 \frac{x^2}{2},$$

która można przekształcić następująco

$$\begin{aligned} (x - 1)^2 &= b_{-2} \frac{x^2 - 2x + 1}{2} + b_{-1} \frac{-2x^2 + 2x + 1}{2} + b_0 \frac{x^2}{2}, \\ x^2 - 2x + 1 &= \frac{b_{-2} - 2b_{-1} + b_0}{2} x^2 + \frac{-2b_{-2} + 2b_{-1}}{2} x + \frac{b_{-2} + b_{-1}}{2} \end{aligned}$$

Dla $x \in [1, 2]$ mamy

$$(x - 1)^2 = b_{-1} \frac{(x-2)^2}{2} + b_0 \frac{x(2-x)+(3-x)(x-1)}{2} + b_1 \frac{(x-1)^2}{2},$$

co jest równoważne równaniom

$$\begin{aligned} (x - 1)^2 &= b_{-1} \frac{x^2 - 4x + 4}{2} + b_0 \frac{-2x^2 + 6x - 3}{2} + b_1 \frac{x^2 - 2x + 1}{2}, \\ (x - 1)^2 &= \frac{b_{-1} - 2b_0 + b_1}{2} x^2 + \frac{-4b_{-1} + 6b_0 - 2b_1}{2} x + \frac{4b_{-1} - 3b_0 + b_1}{2} \end{aligned}$$

Równości na obu przedziałach zachodzą dla dowolnych x z tych przedziałów, zatem są one spełnione wtedy i tylko wtedy,

gdy odpowiednie współczynniki przy potęgach x są równe. Otrzymujemy zatem układ

$$\left\{ \begin{array}{l} \frac{b_{-2}-2b_{-1}+b_0}{2} = 1, \\ \frac{-2b_{-2}+2b_{-1}}{2} = -2, \\ \frac{b_{-2}+b_{-1}}{2} = 1, \\ \frac{b_{-1}-2b_0+b_1}{2} = 1, \\ \frac{-4b_{-1}+6b_0-2b_1}{2} = -2, \\ \frac{4b_{-1}-3b_0+b_1}{2} = 1, \end{array} \right.$$

który jest układem równań liniowych z czterema niewiadomymi. Musi być on niesprzeczny i posiadać dokładnie jedno rozwiązanie, bo jest wynikiem analizy zadania przedstawienia splajnu w bazie, a to zadanie ma jednoznaczne rozwiązanie. Wystarczy wybrać cztery równania i je rozwiązać. Pozostałe dwa mogą służyć do sprawdzenia wyniku. Łatwo sprawdzić, że rozwiązaniem jest

$$b_{-2} = b_1 = 2, \quad b_{-1} = b_0 = 0.$$

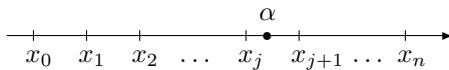
10.7 Obliczanie wartości splajnu w punkcie

W tym punkcie zajmiemy się zadaniem obliczenia wartości splajnu $S(\alpha)$ dla pewnego $\alpha \in [a, b]$. Rozważamy dwa przypadki, w zależności od tego, jak splajn jest przechowywany w pamięci komputera. Pierwszy będzie dotyczył tylko splajnów kubicznych, choć łatwo go uogólnić.

10.7.1 Przypadek reprezentacji splajnu kubicznego w lokalnych bazach potęgowych

Dane:	a_0, a_1, \dots, a_{n-1} b_0, b_1, \dots, b_{n-1} c_0, c_1, \dots, c_{n-1} d_0, d_1, \dots, d_{n-1} — współczynniki S w lokalnych bazach potęgowych x_0, x_1, \dots, x_n — węzły, $\alpha \in [x_0, x_n]$,
Wyniki:	$S(\alpha)$

Rozwiązanie dzieli się na dwie części. Po pierwsze trzeba znaleźć $j \in 0, 1, \dots, n-1$ takie, że $\alpha \in [x_j, x_{j+1}]$.



Można to zrobić metodą bisekcji kosztem rzędu $\log_2 n$. W przypadku węzłów równoodległych $x_i = a + ih$, $i = 0, 1, \dots, n$ szukane j wynosi

$$j = \left\lfloor \frac{\alpha - a}{h} \right\rfloor,$$

co można obliczyć kosztem dwóch działań zmiennopozycyjnych. Jeśli $\alpha = x_n$, to przyjmujemy $j = n - 1$.

Po zlokalizowaniu przedziału, w którym mieści się α należy sięgnąć do odpowiedniej tabelki współczynników i obliczyć $S(\alpha)$ algorytmem Hornera kosztem 7 działań.

```
pom=alpha-x[j];
S=a[j]+pom*(b[j]+pom*(c[j]+pom*d[j]));
```

Koszt obliczenia $S(\alpha)$ jest więc następujący, P oznacza porównanie.

$$\text{koszt czasowy} = \begin{cases} 5 \text{ ops} + 4 \text{ opm} & \text{dla węzłów równoodl.} \\ 4 \text{ ops} + 3 \text{ opm} + \lceil \log_2 n \rceil P & \text{w p.p..} \end{cases}$$

10.7.2 Przypadek reprezentacji splajnu w bazie B-splajnów

Tym razem rozważamy splajny k -tego rzędu dla ustalonego naturalnego k .

Dane:	$b_{-k}, b_{-k+1}, \dots, b_{n-1}$ — współczynniki S w bazie B-splajnów,
	x_0, x_1, \dots, x_n — węzły,
	$\alpha \in [x_0, x_n]$,
Wyniki:	$S(\alpha)$

Splajn S zapisany w bazie B-splajnów ma postać

$$S(x) = \sum_{i=-k}^{n-1} b_i B_i^{(k)}(x),$$

zatem

$$S(\alpha) = \sum_{i=-k}^{n-1} b_i B_i^{(k)}(\alpha),$$

Podobnie jak poprzednio pierwszą czynnością będzie zlokalizowanie przedziału, w którym punkt α się znajduje, czyli znalezienie j takiego, że $\alpha \in [x_j, x_{j+1}]$. Jak poprzednio przyjmujemy $j = n - 1$, gdy $\alpha = x_n$.

Ponieważ nośnikiem funkcji bazowej $B_i^{(k)}$ jest przedział $[x_i, x_{i+k+1}]$, to część składników sumy będzie równa零. Są to mianowicie te, dla których $B_i^{(k)}(\alpha) = 0$, czyli wyznaczone przez i spełniające

$$(x_j, x_{j+1}) \cap (x_i, x_{i+k+1}) = \emptyset,$$

co daje warunki

$$i \geq j + 1 \text{ lub } i + k + 1 \leq j$$

Stąd

$$S(\alpha) = \sum_{i=j-k}^j b_i B_i^{(k)}(\alpha).$$

Suma składa się jedynie z $k + 1$ składników.

Przedstawię poniżej algorytm de Boora służący do obliczania wartość powyższej sumy. W jego wyprowadzeniu korzystamy z definicji rekurencyjnej B-splajnów.

$$\begin{aligned} S(\alpha) &= \sum_{i=j-k}^j b_i B_i^{(k)}(\alpha) \\ &= \sum_{i=j-k}^j b_i \left[\frac{\alpha - x_i}{x_{i+k} - x_i} B_i^{(k-1)}(\alpha) + \frac{x_{i+k+1} - \alpha}{x_{i+k+1} - x_{i+1}} B_{i+1}^{(k-1)}(\alpha) \right] \\ &= \sum_{i=j-k}^j b_i \frac{\alpha - x_i}{x_{i+k} - x_i} B_i^{(k-1)}(\alpha) + \sum_{i=j-k}^j b_i \frac{x_{i+k+1} - \alpha}{x_{i+k+1} - x_{i+1}} B_{i+1}^{(k-1)}(\alpha) \\ &= \sum_{i=j-k}^j b_i \frac{\alpha - x_i}{x_{i+k} - x_i} B_i^{(k-1)}(\alpha) + \sum_{i=j-k+1}^{j+1} b_{i-1} \frac{x_{i+k} - \alpha}{x_{i+k} - x_i} B_i^{(k-1)}(\alpha) \end{aligned}$$

Skorzystajmy znowu z własności nośników funkcji bazowych. Ponieważ $B_{j+1}^{(k-1)}$ ma nośnik $[x_{j+1}, x_{j+1+k}]$, to $B_{j+1}^{(k-1)}(\alpha) = 0$. Podobnie $B_{j-k}^{(k-1)}(\alpha) = 0$. Stąd w obu powyższych sumach zakresy sumowania można przyjąć od $i = j - k + 1$ do j . Po połączeniu sum otrzymamy

$$\begin{aligned} S(\alpha) &= \sum_{i=j-k+1}^j \left(\frac{\alpha - x_i}{x_{i+k} - x_i} b_i + \frac{x_{i+k} - \alpha}{x_{i+k} - x_i} b_{i-1} \right) B_{i+1}^{(k-1)}(\alpha) \\ &= \sum_{i=j-k+1}^j (\beta_i b_i + (1 - \beta_i) b_{i-1}) B_i^{(k-1)}(\alpha), \end{aligned}$$

gdzie

$$\beta_i = \frac{\alpha - x_i}{x_{i+k} - x_i},$$

a ponieważ

$$\frac{\alpha - x_i}{x_{i+k} - x_i} + \frac{x_{i+k} - \alpha}{x_{i+k} - x_i} = 1,$$

to

$$\frac{x_{i+k} - \alpha}{x_{i+k} - x_i} = 1 - \beta_i.$$

Zauważmy, że liczby β_i zależą od α . Obecnie można już sformułować algorytm — należy tak długo korzystać z definicji rekurencyjnej aż dojdziemy do funkcji bazowych rzędu zero. W każdym kroku suma ulega skróceniu o 1.

współczynniki							s
b_{j-k}	b_{j-k+1}	b_{j-k+2}	\dots	b_{j-2}	b_{j-1}	b_j	k
↓	↓	↓	↓	↓	↓	↓	
$b_{j-(k-1)}$	b_{j-k+2}	\dots	b_{j-2}	b_{j-1}	b_j		$k-1$
↓	↓	↓	↓	↓	↓	↓	
$b_{j-(k-2)}$	\dots	b_{j-2}	b_{j-1}	b_j			$k-2$
b_{j-1}	b_j						1
	↓						
b_j							0

Algorytm przebiega kolejne wiersze z góry na dół. Każdy wiersz obliczany jest od prawej do lewej, by móc posługiwać się jedną tablicą do przechowywania kolejnych współczynników. Dla ułatwienia zapisu indeksy tablic nie są numerowane od zera. Zmienna β zawiera kolejne wartości β_i .

```
for(s=k-1;s>=0;s--){
    for(i=j;i>=j-s;i--){
        beta=(alfa-x[i])/(x[i+s+1]-x[i]);
```

```
        b[i]=b[i]*beta+b[i-1]*(1-beta);
    }
    wynik=b[j];
}
```

Wyznaczmy koszt tego algorytmu. W treści pętli wewnętrznej wykonywane są 4 dodawania i 3 mnożenia, koszt tej pętli to $(s+1)(4 \text{ ops} + 3 \text{ opm})$. Calkowity koszt algorytmu to

$$\sum_{s=0}^{k-1} (s+1)(4 \text{ ops} + 3 \text{ opm}) = \frac{k(k+1)}{2}(4 \text{ ops} + 3 \text{ opm}).$$

Należy pamiętać też o koszcie wyznaczenia j .

$$\text{koszt czasowy} = \begin{cases} \frac{k(k+1)}{2}(4 \text{ ops} + 3 \text{ opm}) + \text{ops} + \text{opm} & \text{dla węzłów równoodl.,} \\ \frac{k(k+1)}{2}(4 \text{ ops} + 3 \text{ opm}) + \lceil \log_2 n \rceil P & \text{w p.p.,} \end{cases}$$

P oznacza tu porównanie. Dla splajnów kubicznych mamy

$$\text{koszt czasowy} = \begin{cases} (25 \text{ ops} + 19 \text{ opm}) & \text{dla węzłów równoodl.,} \\ (24 \text{ ops} + 18 \text{ opm}) + \lceil \log_2 n \rceil P & \text{w p.p.,} \end{cases}$$

co przy węzłach równoodległych daje koszt niemal dziewięciokrotnie wyższy niż w przypadku lokalnych baz potęgowych.

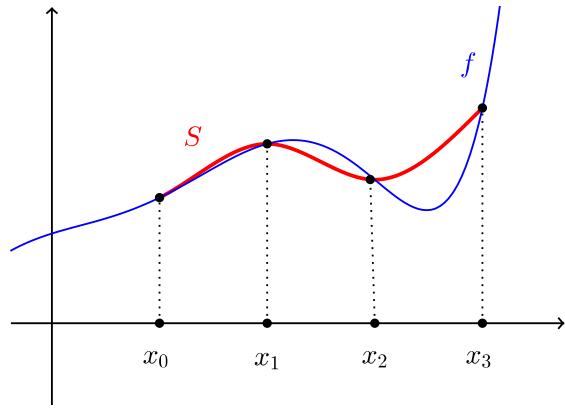
Przedstawiony algorytm de Boora jest numerycznie poprawny, wskaźniki kumulacji są tu rzędu k . Uwarunkowanie zadania jest znacznie lepsze niż w przypadku lokalnych baz potęgowych.

11 Interpolacja splajnami kubicznymi

11.1 Sformułowanie zadania interpolacji

Dany jest układ węzłów $x_0 < x_1 < \dots < x_n$, oraz wartości funkcji f w tych węzłach. Chcemy interpolować f splajnem kubicznym S opartym na zadanych węzłach. Ma on zatem spełniać równania interpolacyjne

$$S(x_i) = f(x_i), \text{ dla } i = 0, \dots, n.$$



Podobnie jak przy interpolacji Lagrange'a do zapisu danych można stosować tabelkę interpolacyjną

$$\begin{array}{c|c|c|c|c} x_0 & x_1 & \dots & x_n \\ \hline f(x_0) & f(x_1) & \dots & f(x_n) \end{array}.$$

Przed przystąpieniem do opracowania algorytmu należy odpowiedzieć na dwa pytania.

1. Czy istnieje taki splajn S ?
2. Czy S jest wyznaczony jednoznacznie?

Okazuje się, że istnieje, ale nie jedyny. Przestrzeń splajnów kubicznych jest wymiaru $n + 3$, czyli aby wyznaczyć splajn S wystarczy znać jego $n + 3$ współczynniki w dowolnej bazie. Ustalmy zatem bazę u_1, u_2, \dots, u_{n+3} . Szukamy takich $a_1, a_2, \dots, a_{n+3} \in \mathbb{R}$, że

$$S(x) = a_1 u_1(x) + a_2 u_2(x) + \dots + a_{n+3} u_{n+3}(x).$$

Równania interpolacyjne prowadzą do układu równań liniowych

$$\begin{cases} a_1 u_1(x_0) + a_2 u_2(x_0) + \dots + a_{n+3} u_{n+3}(x_0) = f(x_0), \\ a_1 u_1(x_1) + a_2 u_2(x_1) + \dots + a_{n+3} u_{n+3}(x_1) = f(x_1), \\ \vdots \\ a_1 u_1(x_n) + a_2 u_2(x_n) + \dots + a_{n+3} u_{n+3}(x_n) = f(x_n), \end{cases}$$

przy czym niewiadomych jest $n + 3$, a równań $n + 1$. Jest to układ niesprzeczny, zatem ma nieskończenie wiele rozwiązań. Aby miał on jedyne rozwiązanie warunkiem koniecznym jest dodanie dwóch równań. Okazuje się, że jeśli zrobi się to na jeden z poniższych sposobów, to S będzie wyznaczony jednoznacznie.¹⁴ Wybrane sposoby są używanymi w zadaniach praktycznych.

1. $S''(x_0) = 0, S''(x_n) = 0$

Rozwiązaniem zadania interpolacji będzie wtedy splajn naturalny.

2. $S''(x_0) = 0, S'(x_0) = f'(x_0)$

Uzyskamy splajn lewostronnie swobodny z narzuconym warunkiem na pochodną w lewym krańcu (czyli współczynnikiem kierunkowym prostej, która może przedłużyć splajn w kierunku $-\infty$ z zachowaniem dwóch ciągłych pochodnych).

3. $S''(x_n) = 0, S'(x_n) = f'(x_n)$

Splajn prawostronnie swobodny z warunkiem na pochodną w prawym krańcu.

4. $S'(x_0) = f'(x_0), S'(x_n) = f'(x_n)$

Uzyskamy tzw. *splajn hermitowski*. Węzły wewnętrzne są jednokrotne, zaś brzegowe dwukrotne.

5. $S(x_0) = S(x_n), S'(x_0) = S'(x_n), S''(x_0) = S''(x_n)$

Uzyskamy splajn okresowy. Choć widoczne są trzy dodatkowe równania, to faktycznie są dwa. Pierwsze stanowi jedynie opis, kiedy można taki sposób dodania rów-

nań zastosować. Mianowicie wśród równań interpolacyjnych są

$$\begin{aligned} S(x_0) &= f(x_0), \\ S(x_n) &= f(x_n). \end{aligned}$$

Z nich i równania $S(x_0) = S(x_n)$ wynika, że $f(x_0) = f(x_n)$. Jeśli tak nie będzie, to interpolacja splajnem określonym nie będzie możliwa.

6. $S(x_{-1}) = f(x_{-1}), S(x_{n+1}) = f(x_{n+1})$

Ten sposób stosuje się jedynie przy reprezentacji w bazie B-splajnów. Węzły x_{-1} oraz x_{n+1} pochodzą z siatki rozszerzonej.

11.2 Wyznaczenie współczynników w lokalnych bazach potęgowych splajnu naturalnego interpolującego funkcję

Dane: x_0, x_1, \dots, x_n — węzły,
 $f(x_0), f(x_1), \dots, f(x_n)$ — wartości,

Wyniki: $a_0, a_1, \dots, a_{n-1},$
 $b_0, b_1, \dots, b_{n-1},$
 $c_0, c_1, \dots, c_{n-1},$
 d_0, d_1, \dots, d_{n-1} — współczynniki w lokalnych bazach potęgowych splajnu naturalnego S interpolującego f

11.2.1 Układ równań

Przypomnijmy, że splajn S reprezentowany w lokalnych bazach potęgowych wyraża się dla $x \in [x_i, x_{i+1}]$ wzorem

$$S(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3.$$

Korzystając z tego przedstawienia i z równań interpolacyjnych

$$S(x_i) = f(x_i), \quad i = 0, 1, \dots, n-1,$$

otrzymujemy

$$a_i = f(x_i), \quad i = 0, 1, \dots, n-1.$$

W ostatnim przedziale $[x_{n-1}, x_n]$ splajn S ma postać

$$\begin{aligned} S(x) = & a_{n-1} + b_{n-1}(x - x_{n-1}) + c_{n-1}(x - x_{n-1})^2 + \\ & + d_{n-1}(x - x_{n-1})^3, \end{aligned}$$

z której należy skorzystać by napisać ostatnie równanie interpolacyjne $S(x_n) = f(x_n)$. Aby skrócić zapis stosuję oznaczenie $h_{n-1} = x_n - x_{n-1}$

$$a_{n-1} + b_{n-1}h_{n-1} + c_{n-1}h_{n-1}^2 + d_{n-1}h_{n-1}^3 = f(x_n).$$

Wiemy, że S jest naturalny, czyli $S''(x_0) = S''(x_n) = 0$. Obliczmy pierwszą i drugą pochodną splajnu S w $[x_i, x_{i+1}]$:

$$\begin{aligned} S'(x) &= b_i + 2c_i(x - x_i) + 3d_i(x - x_i)^2, \\ S''(x) &= 2c_i + 6d_i(x - x_i). \end{aligned}$$

¹⁴Za dowody można uznać wyprowadzenia algorytmów w następnych punktach. Obejmą one dwa przypadki, ale reszta przebiega podobnie.

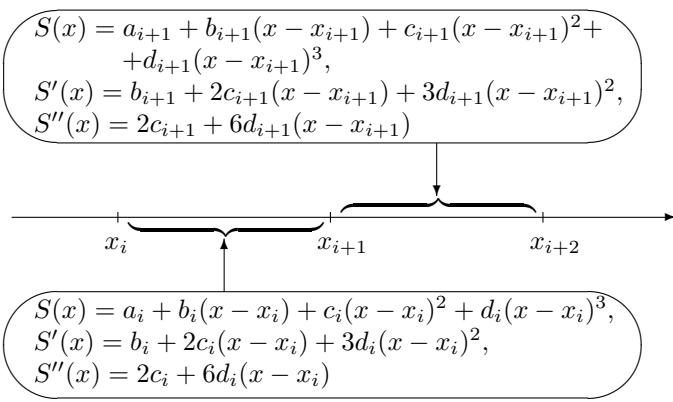
Dwa następne równania wyrażają zerowanie się drugich pochodnych w krańcowych węzłach

$$\begin{aligned} 2c_0 &= 0, \\ 2c_{n-1} + 6d_{n-1}(x_n - x_{n-1}) &= 0. \end{aligned}$$

Po uproszczeniu i podstawieniu $h_{n-1} = x_n - x_{n-1}$ otrzymujemy

$$\boxed{\begin{aligned} c_0 &= 0, \\ c_{n-1} + 3d_{n-1}h_{n-1} &= 0. \end{aligned}}$$

Jak dotąd napisaliśmy $n+3$ równania (te w ramkach), a niewiadomych jest $4n$. Brakujące równania opiszą tzw. warunki dobrego sklejenia. Przypomnijmy, że wzięcie dowolnych współczynników a_i, b_i, c_i, d_i nie gwarantuje uzyskania funkcji, która jest splajnem, bo może ona nie mieć dwóch ciągłych pochodnych. Ciągłość jest zagrożona w węzłach wewnętrznych x_1, x_2, \dots, x_{n-1} , gdyż z lewej i prawej strony każdego takiego węzła mamy do czynienia z różnymi wielomianami.



Wzory obowiązujące z lewej i prawej strony węzła wewnętrznego x_{i+1} muszą dać te same wartości dla $x = x_{i+1}$. Stąd z ciągłości S

$$a_i + b_i(x_{i+1} - x_i) + c_i(x_{i+1} - x_i)^2 + d_i(x_{i+1} - x_i)^3 = a_{i+1},$$

z ciągłości pierwszej pochodnej

$$b_i + 2c_i(x_{i+1} - x_i) + 3d_i(x_{i+1} - x_i)^2 = b_{i+1},$$

z ciągłości drugiej pochodnej

$$2c_i + 6d_i(x_{i+1} - x_i) = 2c_{i+1},$$

gdzie $i = 0, 1, \dots, n-2$. Podstawiając $h_i = x_{i+1} - x_i$ i porządkując te równania otrzymujemy

$$\boxed{\begin{aligned} b_i h_i + c_i h_i^2 + d_i h_i^3 &= a_{i+1} - a_i, \\ 2c_i h_i + 3d_i h_i^2 &= b_{i+1} - b_i, \\ 3d_i h_i &= c_{i+1} - c_i, \end{aligned} \quad i = 0, 1, \dots, n-2}$$

Wszystkie równania w ramkach dają układ $4n$ równań z $4n$ niewiadomymi. Podstawiając $a_i = f(x_i)$ dla $i = 0, 1, \dots, n-1$ otrzymamy

$$\boxed{\begin{aligned} b_i h_i + c_i h_i^2 + d_i h_i^3 &= f(x_{i+1}) - f(x_i), \quad i = 0, 1, \dots, n-1, \\ 2c_i h_i + 3d_i h_i^2 &= b_{i+1} - b_i, \quad i = 0, 1, \dots, n-2, \\ 3d_i h_i &= c_{i+1} - c_i, \quad i = 0, 1, \dots, n-2 \\ 3d_{n-1} h_{n-1} &= 0 - c_{n-1}, \\ c_0 &= 0, \\ a_i &= f(x_i), \end{aligned} \quad i = 0, 1, \dots, n-1}$$

Zauważmy, że równania z trzeciej i czwartej linii są podobnej postaci. Układ da się zapisać bardziej elegancko, jeśli wprowadzimy pomocnicze oznaczenie $c_n = 0$ i połączymy te dwie linie. Przy okazji przekształćmy pierwszą grupę równań. Podzielmy je przez h_i (odpowiednio) i zauważmy, że

$$\frac{f(x_{i+1}) - f(x_i)}{h_i} = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} = f_{x_i, x_{i+1}}$$

jest różnicą dzieloną pierwszego rzędu. Zatem mamy układ

$$\begin{cases} b_i + c_i h_i + d_i h_i^2 = f_{x_i, x_{i+1}}, & i = 0, 1, \dots, n-1, \\ 2c_i h_i + 3d_i h_i^2 = b_{i+1} - b_i, & i = 0, 1, \dots, n-2, \\ 3d_i h_i = c_{i+1} - c_i, & i = 0, 1, \dots, n-1 \\ c_0 = 0, \\ c_n = 0, \\ a_i = f(x_i), & i = 0, 1, \dots, n-1 \end{cases}$$

Przykład Napiszemy równania pozwalające wyznaczyć współczynniki splajnu kubicznego obustronnie swobodnego interpolującego funkcję $f(x) = \ln x$ w węzłach $1, e, e^2$.

Mamy trzy węzły definiujące splajn, zatem dwa przedziały, w lokalnych bazach potęgowych S ma postać

$$S(x) = \begin{cases} S_1(x) & \text{dla } x \in [1, e], \\ S_2(x) & \text{dla } x \in [e, e^2], \end{cases}$$

gdzie

$$\begin{aligned} S_1(x) &= A + B(x-1) + C(x-1)^2 + D(x-1)^3, \\ S_2(x) &= E + F(x-e) + G(x-e)^2 + H(x-e)^3. \end{aligned}$$

Szukamy współczynników A, B, C, D, E, F, G, H . W dalszej części przydadzą się wzory na pochodne

$$\begin{aligned} S'_1(x) &= B + 2C(x-1) + 3D(x-1)^2, \\ S'_2(x) &= F + 2G(x-e) + 3H(x-e)^2, \\ S''_1(x) &= 2C + 6D(x-1), \\ S''_2(x) &= 2G + 6H(x-e). \end{aligned}$$

Splajn ma interpolować funkcję f , zatem

$$\begin{aligned} S(1) &= f(1), \\ S(e) &= f(e), \\ S(e^2) &= f(e^2). \end{aligned}$$

Aby powyższe wzory opisywały splajn trzeba zagwarantować, by S był klasy C^2 , czyli

$$\begin{aligned} S_1(e) &= S_2(e), \\ S'_1(e) &= S'_2(e), \\ S''_1(e) &= S''_2(e). \end{aligned}$$

Ponadto splajn ma być obustronnie swobodny, czyli

$$\begin{aligned} S''(1) &= 0, \\ S''(e^2) &= 0. \end{aligned}$$

Możemy zatem napisać osiem równań z ośmioma niewiadomymi:

$$\begin{cases} A = 0, \\ E = 1, \\ E + F(e^2 - e) + G(e^2 - e)^2 + H(e^2 - e)^3 = 2, \\ A + B(e - 1) + C(e - 1)^2 + D(e - 1)^3 = E, \\ B + 2C(e - 1) + 3D(e - 1)^2 = F, \\ 2C + 6D(e - 1) = 2G, \\ 2C = 0, \\ 2G + 6H(e^2 - e) = 0. \end{cases}$$

11.2.2 Rozwiązywanie układu równań

Naszym obecnym zadaniem jest rozwiązywanie układu

$$\begin{cases} b_i + c_i h_i + d_i h_i^2 = f_{x_i, x_{i+1}}, & i = 0, 1, \dots, n-1, \\ 2c_i h_i + 3d_i h_i^2 = b_{i+1} - b_i, & i = 0, 1, \dots, n-2, \\ 3d_i h_i = c_{i+1} - c_i, & i = 0, 1, \dots, n-1 \\ c_0 = 0, \\ c_n = 0, \\ a_i = f(x_i), & i = 0, 1, \dots, n-1 \end{cases}$$

Odbiera się to częściowo metodą podstawień. Z trzeciej grupy równań wyznaczamy d_i

$$d_i = \frac{c_{i+1} - c_i}{3h_i}, \quad i = 0, 1, \dots, n-1.$$

i podstawiamy do pierwszej grupy

$$b_i + c_i h_i + \frac{c_{i+1} - c_i}{3} h_i = f_{x_i, x_{i+1}}.$$

Stąd obliczamy b_i dla $i = 0, 1, \dots, n-1$

$$\begin{aligned} b_i &= f_{x_i, x_{i+1}} - c_i h_i - \frac{c_{i+1} - c_i}{3} h_i \\ &= f_{x_i, x_{i+1}} - \frac{c_{i+1} + 2c_i}{3} h_i. \end{aligned}$$

Wstawiamy b_i oraz d_i do drugiej grupy równań

$$\begin{aligned} 2c_i h_i + (c_{i+1} - c_i) h_i &= f_{x_{i+1}, x_{i+2}} - \frac{c_{i+2} + 2c_{i+1}}{3} h_{i+1} \\ &\quad - f_{x_i, x_{i+1}} + \frac{c_{i+1} + 2c_i}{3} h_i. \end{aligned}$$

Po uporządkowaniu otrzymujemy

$$\frac{c_i}{3} h_i + 2\frac{c_{i+1}}{3} (h_{i+1} + h_i) + \frac{c_{i+2}}{3} h_{i+1} = f_{x_{i+1}, x_{i+2}} - f_{x_i, x_{i+1}}.$$

Zauważmy, że gdy obie strony podzielimy przez $h_{i+1} + h_i$, to z prawej mamy

$$\frac{f_{x_{i+1}, x_{i+2}} - f_{x_i, x_{i+1}}}{h_{i+1} + h_i} = \frac{f_{x_{i+1}, x_{i+2}} - f_{x_i, x_{i+1}}}{x_{i+2} - x_i} = f_{x_i, x_{i+1}, x_{i+2}}.$$

Ponadto, ponieważ z lewej strony wszystkie wyrazy c_i , c_{i+1} , c_{i+2} są dzielone przez 3, to warto wprowadzić pomocnicze niewiadome $\tilde{c}_i = \frac{c_i}{3}$ dla $i = 0, 1, \dots, n$. Notacja będzie krótsza, gdy przyjmujemy $P_{i+1} = \frac{h_i}{h_{i+1} + h_i}$ oraz $R_{i+1} = \frac{h_{i+1}}{h_{i+1} + h_i}$. Wtedy otrzymamy

$$\tilde{c}_i P_{i+1} + 2\tilde{c}_{i+1} + \tilde{c}_{i+2} R_{i+1} = f_{x_i, x_{i+1}, x_{i+2}}$$

dla $i = 0, \dots, n-2$. Po uwzględnieniu, że $c_0 = c_n = 0$ powstaje układ równań liniowych z niewiadomymi $\tilde{c}_1, \tilde{c}_2, \dots, \tilde{c}_{n-1}$

$$\begin{cases} 2\tilde{c}_1 + \tilde{c}_2 R_1 = f_{x_0, x_1, x_2}, \\ \tilde{c}_1 P_2 + 2\tilde{c}_2 + \tilde{c}_3 R_2 = f_{x_1, x_2, x_3}, \\ \tilde{c}_2 P_3 + 2\tilde{c}_3 + \tilde{c}_4 R_3 = f_{x_2, x_3, x_4}, \\ \vdots \\ \tilde{c}_{n-4} P_{n-3} + 2\tilde{c}_{n-3} + \tilde{c}_{n-2} R_{n-3} = f_{x_{n-4}, x_{n-3}, x_{n-2}}, \\ \tilde{c}_{n-3} P_{n-2} + 2\tilde{c}_{n-2} + \tilde{c}_{n-1} R_{n-2} = f_{x_{n-3}, x_{n-2}, x_{n-1}}, \\ \tilde{c}_{n-2} P_{n-1} + 2\tilde{c}_{n-1} = f_{x_{n-2}, x_{n-1}, x_n}, \end{cases}$$

W postaci macierzowej zapisujemy go następująco

$$\begin{bmatrix} 2 & R_1 & & & & & 0 \\ P_2 & 2 & R_2 & & & & \tilde{c}_1 \\ & P_3 & 2 & R_3 & & & \tilde{c}_2 \\ & & \ddots & \ddots & \ddots & & \vdots \\ & & & P_{n-3} & 2 & R_{n-3} & \tilde{c}_{n-3} \\ & & & & P_{n-2} & 2 & R_{n-2} \\ 0 & & & & & P_{n-1} & 2 \end{bmatrix} \begin{bmatrix} \tilde{c}_1 \\ \tilde{c}_2 \\ \tilde{c}_3 \\ \vdots \\ \tilde{c}_{n-3} \\ \tilde{c}_{n-2} \\ \tilde{c}_{n-1} \end{bmatrix} = \begin{bmatrix} f_{x_0, x_1, x_2} \\ f_{x_1, x_2, x_3} \\ f_{x_2, x_3, x_4} \\ \vdots \\ f_{x_{n-4}, x_{n-3}, x_{n-2}} \\ f_{x_{n-3}, x_{n-2}, x_{n-1}} \\ f_{x_{n-2}, x_{n-1}, x_n} \end{bmatrix}.$$

Jest on specjalnej postaci: z tzw. macierzą *trójdiamondową*. Oznacza to, że oprócz głównej diagonali i dwóch sąsiednich kodiagonali reszta macierzy jest wypełniona zerami. Ponadto ma własność *dominującej diagonali* czyli, że wartości stojące na głównej przekątnej są co do modułu większe niż suma modułów pozostałych wyrazów wiersza. Rzeczywiście

$$P_{i+1} + R_{i+1} = \frac{h_i}{h_{i+1} + h_i} + \frac{h_{i+1}}{h_{i+1} + h_i} = 1 < 2,$$

dla $i = 2, \dots, n-2$, również $R_1 < 2$, $P_{n-1} < 2$. Można udowodnić, że taka macierz jest nieosobliwa oraz zadanie rozwiązywanie układu równań z nią jest dobrze uwarunkowane. Numerycznie poprawny algorytm rozwiązywania tego układu przedstawię w dalszej części wykładu.

Po obliczeniu wszystkich współczynników \tilde{c}_i należy po kolej obliczać

$$\begin{aligned} d_i &= \frac{\tilde{c}_{i+1} - \tilde{c}_i}{h_i}, & i = 0, 1, \dots, n-1, \\ c_i &= 3\tilde{c}_i, & i = 1, 2, \dots, n-1, \end{aligned}$$

potem b_i . Wzorem $b_i = f_{x_i, x_{i+1}} - \frac{c_{i+1} + 2c_i}{3} h_i$ warto się jedynie posłużyć dla $i = 0$, natomiast taniej będzie pozostałe wyrazy obliczyć z zależności rekurencyjnej, która dana jest przez drugą grupę równań

$$2c_i h_i + 3d_i h_i^2 = b_{i+1} - b_i, \quad i = 0, 1, \dots, n-2.$$

Stąd dla $i = 0, 1, \dots, n-2$

$$\begin{aligned} b_{i+1} &= b_i + 2c_i h_i + 3d_i h_i^2 \\ &= b_i + 2c_i h_i + (c_{i+1} - c_i) h_i \\ &= b_i + (c_{i+1} + c_i) h_i. \end{aligned}$$

11.2.3 Algorytm wyznaczenia współczynników splajnu

Algorytm będzie posługiwał się tablicami:

1. x, f — węzły i wartości funkcji (indeksy od 0 do n),
2. a, b, c, d — współczynniki (indeksy od 0 do $n - 1$),
3. h — robocza przechowująca h_i (indeksy od 0 do $n - 1$),
4. P — robocza, do zapamiętania dolnej kodiagonali (indeksy od 2 do $n - 1$),
5. R — robocza, do zapamiętania górnej kodiagonali (indeksy od 1 do $n - 1$).

Krok 1 Wykonaj $a[i]=f[i]$ dla $i = 0, 1, \dots, n - 1$.

Krok 2 Oblicz różnice dzielone

$$\begin{aligned} & f_{x_0, x_1} \\ & f_{x_{i-2}, x_{i-1}, x_i} \text{ dla } i = 2, 3, \dots, n \end{aligned}$$

przez wykonanie dwóch kroków algorytmu różnic dzielonych. Przy okazji zapamiętaj w $h[i]$ różnicę $x_{i+1} - x_i$, a w $R[i]$ wielkość $h_i + h_{i-1} = x_{i+1} - x_{i-1}$.

```
for(i=n;i>=1;i--) {
    h[i-1]=x[i]-x[i-1];
    f[i]=(f[i]-f[i-1])/h[i-1];
}
for(i=n;i>=2;i--) {
    R[i-1]=x[i]-x[i-2];
    f[i]=(f[i]-f[i-1])/R[i-1];
}
```

Po wykonaniu tego kroku

f_{x_0, x_1} jest w $f[1]$
 $f_{x_{i-2}, x_{i-1}, x_i}$ jest w $f[i]$ dla $i = 2, 3, \dots, n$,
 h_i jest w $h[i]$,
 $h_i + h_{i-1}$ jest w $R[i]$ dla $i = 1, 2, \dots, n - 1$.

Krok 3 Oblicz równocześnie kodiagonale, umieść wyniki w tablicach P i R .

```
R[1]=h[1]/R[1];
for(i=2;i<=n-2;i--) {
    P[i]=h[i-1]/R[i];
    R[i]=h[i]/R[i];
}
P[n-1]=h[n-2]/R[n-1];
```

Krok 4 Rozwiąż układ równań z macierzą trójdiamondalną o dominującej przekątnej. Kodiagonala dolna jest zapisana w tablicy P (indeksy od 2 do $n - 1$), Kodiagonala góra w R (indeksy od 1 do $n - 2$), wektor prawej strony w tablicy f (indeksy od 2 do n). Rozwiążanie umieść w tablicy c na pozycjach od indeksu 1 do $n - 1$ (są to liczby \tilde{c}_i). Przypisz $c[0]=0.0$; $c[n]=0.0$;

Krok 5 Oblicz $d_i = (\tilde{c}_{i+1} - \tilde{c}_i)/h_i$ i umieść w $d[i]$ dla $i = 0, 1, \dots, n - 1$. Oblicz $b_0 = f_{x_0, x_1} - \tilde{c}_1 h_0$ i zapisz w $b[0]$.

Krok 6 Wykonaj $c[i]=c[i]*3$; dla $i = 1, 2, \dots, n - 1$.

Krok 7 Oblicz b_{i+1} posługując się zależnością rekurencyjną $b_{i+1} = b_i + (c_{i+1} + c_i)h_i$ dla $i = 0, 2, \dots, n - 2$. Wyniki zapisz w tablicy b .

Tablica h nie jest konieczna — można nadpisać tablicę x , ale trzeba wówczas zmienić sposób obliczania $R[i]$ w algorytmie różnic dzielonych.

Poniższa tabela zawiera koszt czasowy całego algorytmu z podziałem na kroki,

Krok	ops	opm
1	—	—
2	$4n - 2$	$2n - 1$
3	—	$2n - 4$
4	$3n - 6$	$5n - 9$
5	$n + 1$	$n + 1$
6	—	$n - 1$
7	$2n - 2$	$n - 1$
Razem	$10n - 9$	$12n - 15$

Jeśli użyjemy węzłów równoodległych, to koszt obniży się do $(8n - 8)$ ops + $(8n - 7)$ opm. Dla dużych n przedstawiony algorytm jest tańszy niż jakikolwiek stosowany przy interpolacji Lagrange'a. Nie będzie tak jednak dla np. pięciu węzłów.

Algorytmy dla innych sposobów dodania dwóch brakujących równań wyprowadza się podobnie.

11.2.4 Algorytm rozwiązywania układów równań z macierzą trójdiamondalną o dominującej przekątnej

Zajmiemy się obecnie krokiem piątym, czyli zadaniem rozwiązywania układu równań z macierzą trójdiamondalną o dominującej przekątnej. Rozwiążemy zadanie w ogólnej postaci. Założymy, że dana jest macierz

$$\mathbf{A} = \begin{bmatrix} q_1 & r_1 & & & & & & 0 \\ p_2 & q_2 & r_2 & & & & & \\ & p_3 & q_3 & r_3 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & p_{m-2} & q_{m-2} & r_{m-2} & & \\ & & & & p_{m-1} & q_{m-1} & r_{m-1} & \\ 0 & & & & & p_m & q_m & \end{bmatrix}$$

przy czym

$$\begin{aligned} |p_i| + |r_i| &< |q_i|, \quad \text{dla } i = 2, 3, \dots, m - 1, \\ |r_1| &< |q_1|, \\ |p_m| &< |q_m|. \end{aligned}$$

Naszym zadaniem jest rozwiązywanie układu równań

$$\mathbf{A}\vec{x} = \vec{b},$$

przy zadanym wektorze prawej strony $\vec{b} \in \mathbb{R}^m$.

Dane:	p_2, p_3, \dots, p_m — dolna kodiagonala, q_1, q_2, \dots, q_m — diagonalna, r_1, r_2, \dots, r_{m-1} — górna kodiagonala, b_0, b_1, \dots, b_m — wektor prawej strony,
Wyniki:	x_1, x_2, \dots, x_m — wektor rozwiązania

Algorytm działa w dwóch fazach. Pierwsza polega na takim przekształceniu układu równań, by otrzymać równoważny z macierzą dwudiagonalną. W tym celu wielokrotnie wykonamy operację odjęcia od równania innego równania pomnożonego przez stałą. Druga faza to rozwiązanie układu z macierzą dwudiagonalną.

Rozważmy dwa pierwsze równania.

$$\begin{cases} q_1 x_1 + r_1 x_2 = b_1 \\ p_2 x_1 + q_2 x_2 + r_2 x_3 = b_2 \end{cases} \quad | \cdot \frac{p_2}{q_1}$$

Odejmijmy od drugiego pierwsze pomnożone przez $\frac{p_2}{q_1}$.

$$\begin{cases} q_1 x_1 + r_1 x_2 = b_1 \\ (q_2 - r_1 \frac{p_2}{q_1}) x_2 + r_2 x_3 = b_2 - b_1 \frac{p_2}{q_1} \end{cases}$$

Wyrugowaliśmy w ten sposób niewiadomą x_1 z drugiego równania. Podobnie postępujemy z równaniem drugim („nowym”) i trzecim

$$\begin{cases} \tilde{q}_2 x_2 + r_2 x_3 = \tilde{b}_2 \\ p_3 x_2 + q_3 x_3 + r_3 x_4 = b_3 \end{cases} \quad | \cdot \frac{p_3}{q_2}$$

po wyrugowaniu x_2 z trzeciego (przez odjęcie od trzeciego drugiego pomnożonego przez $\frac{p_3}{q_2}$) mamy

$$\begin{cases} \tilde{q}_2 x_2 + r_2 x_3 = \tilde{b}_2 \\ (q_3 - r_2 \frac{p_3}{q_2}) x_3 + r_3 x_4 = b_3 - \tilde{b}_2 \frac{p_3}{q_2}. \end{cases}$$

Tak postępujemy $m - 1$ razy biorąc kolejne równania. Za- uważmy, że elementy r_i nie ulegają zmianie, a wszystkie p_i staną się zerami.

```
for(i=2;i<=m;i++){
    mnoz=p[i]/q[i-1];
    q[i]=q[i]-mnoz*r[i-1];
    b[i]=b[i]-mnoz*b[i-1];
}
```

Po wykonaniu tej fazy układ ma postać

$$\begin{bmatrix} q_1 & r_1 & & 0 & & & \\ \tilde{q}_2 & r_2 & & & & & \\ & \tilde{q}_3 & r_3 & & & & \\ & \ddots & \ddots & & & & \\ & & & \tilde{q}_{m-1} & r_{m-1} & & \\ 0 & & & & & & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{m-1} \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ \tilde{b}_2 \\ \tilde{b}_3 \\ \vdots \\ \tilde{b}_{m-1} \\ \tilde{b}_m \end{bmatrix}.$$

Z ostatniego równania

$$\tilde{q}_m x_m = \tilde{b}_m$$

wyznaczamy

$$x_m = \frac{\tilde{b}_m}{\tilde{q}_m}.$$

Z przedostatniego

$$\tilde{q}_{m-1} x_{m-1} + r_{m-1} x_m = \tilde{b}_{m-1}$$

obliczamy x_{m-1} (bo x_m jest już znane)

$$x_{m-1} = \frac{\tilde{b}_{m-1} - r_{m-1} x_m}{\tilde{q}_{m-1}},$$

i tak dalej, aż otrzymamy x_1 .

```
x[m]=b[m]/q[m];
for(i=m-1;i>=1;i--)
    x[i]=(b[i]-r[i]*x[i+1])/q[i];
```

Można udowodnić, że algorytm jest numerycznie poprawny. Koszt fazy pierwszej to $(2m-2)$ ops + $(3m-3)$ opm, a drugiej $(m-1)$ ops + $(2m-1)$ opm. Całkowity koszt algorytmu wynosi $(3m-3)$ ops + $(5m-4)$ opm.

11.3 Wyznaczenie współczynników kubicznego splajnu interpolacyjnego reprezentowanego w bazie B-splajnów

Nasze rozważania zawężymy do przypadku splajnów kubicznych. Można je uogólnić dla innych rzędów.

Dane:	x_0, x_1, \dots, x_n — węzły, $f(x_{-1}), f(x_0), \dots, f(x_n), f(x_{n+1})$ — wartości, gdzie $x_{-1} = x_0 - (x_1 - x_0)$,
Wyniki:	$b_{-3}, b_{-2}, \dots, b_{n-1}$ — współczynniki w bazie B-splajnów

Warunki interpolacyjne $S(x_i) = y_i$ dla $i = -1, 0, \dots, n+1$ dają równania

$$\sum_{j=-3}^{n-1} b_j B_j^{(3)}(x_i) = y_i \quad \text{dla } i = -1, 0, \dots, n+1.$$

Większość składników powyższej sumy jest zerami, bo nośnik funkcji $B_j^{(3)}$ to $[x_j, x_{j+4}]$, zatem $B_j^{(3)}(x_i) \neq 0$ jedynie dla $j = i-1, i-2, i-3$. Stąd powyższe równania redukują się do

$$\begin{aligned} b_{-3} B_{-3}^{(3)}(x_{-1}) + b_{-2} B_{-2}^{(3)}(x_{-1}) &= f(x_{-1}), \\ b_{i-3} B_{i-3}^{(3)}(x_i) + b_{i-2} B_{i-2}^{(3)}(x_i) + b_{i-1} B_{i-1}^{(3)}(x_i) &= f(x_i), \\ b_{n-2} B_{n-2}^{(3)}(x_{n+1}) + b_{n-1} B_{n-1}^{(3)}(x_{n+1}) &= f(x_{n+1}), \end{aligned} \quad i = 0, 1, \dots, n$$

Należy obliczyć odpowiednie wartości $B_j^{(3)}(x_i)$. Można tego dokonać stosując definicję rekurencyjną B-splajnów. Pozostajemy jako ćwiczenie udowodnienie, że

$$B_j^{(0)}(x_i) = \begin{cases} 1 & \text{dla } j = i, \\ 0 & \text{w.p.}, \end{cases}$$

$$B_j^{(1)}(x_i) = \begin{cases} 1 & \text{dla } j = i-1, \\ 0 & \text{w.p.}, \end{cases}$$

$$B_j^{(2)}(x_i) = \begin{cases} \frac{x_{i+1}-x_i}{x_{i+1}-x_{i-1}} & \text{dla } j = i-2, \\ \frac{x_i-x_{i-1}}{x_{i+1}-x_{i-1}} & \text{dla } j = i-1, \\ 0 & \text{w p.p.,} \end{cases}$$

$$B_j^{(3)}(x_i) = \begin{cases} \frac{(x_{i+1}-x_i)^2}{(x_{i+1}-x_{i-2})(x_{i+1}-x_{i-1})} & \text{dla } j = i-3, \\ \frac{(x_{i+1}-x_i)(x_i-x_{i-2})}{(x_{i+1}-x_{i-2})(x_{i+1}-x_{i-1})} + \\ + \frac{(x_i-x_{i-1})(x_{i+2}-x_i)}{(x_{i+2}-x_{i-1})(x_{i+1}-x_{i-1})} & \text{dla } j = i-2, \\ \frac{(x_i-x_{i-1})^2}{(x_{i+2}-x_{i-1})(x_{i+1}-x_{i-1})} & \text{dla } j = i-1, \\ 0 & \text{w p.p.,} \end{cases}$$

Jeśli zastosujemy pomocnicze oznaczenia

$$\begin{aligned} h_i &= x_{i+1} - x_i, \quad i = -2, -1, \dots, n+1, \\ g_i &= x_{i+2} - x_i, \quad i = -3, -1, \dots, n, \\ G_i &= x_{i+3} - x_i, \quad i = -3, -1, \dots, n, \end{aligned}$$

to możemy zapisać

$$B_j^{(3)}(x_i) = \begin{cases} h_i \frac{h_i}{G_{i-2}g_{i-1}} & \text{dla } j = i-3, \\ g_i \frac{h_{i-1}}{G_{i-1}g_{i-1}} + g_{i-2} \frac{h_i}{G_{i-2}g_{i-1}} & \text{dla } j = i-2, \\ h_{i-1} \frac{h_{i-1}}{G_{i-1}g_{i-1}} & \text{dla } j = i-1. \end{cases}$$

Zatem zadanie wyznaczania współczynników b_{-3}, \dots, b_{n-1} sprowadza się do rozwiązywania układu równań $\mathbf{A}\vec{b} = \vec{f}$, gdzie

$$\mathbf{A} = \begin{bmatrix} q_{-3} & r_{-3} & & & & & 0 \\ p_{-1} & q_{-1} & r_{-1} & & & & \\ p_0 & q_0 & r_0 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & p_{n-3} & q_{n-3} & r_{n-3} & & \\ & & & p_{n-2} & q_{n-2} & r_{n-2} & \\ 0 & & & & p_{n-1} & q_{n-1} & \end{bmatrix}$$

$$p_i = B_{i-3}^{(3)}(x_i), \quad q_i = B_{i-2}^{(3)}(x_i), \quad p_i = B_{i-1}^{(3)}(x_i),$$

$$\vec{b} = \begin{bmatrix} b_{-3} \\ b_{-2} \\ b_{-1} \\ \vdots \\ b_{n-3} \\ b_{n-2} \\ b_{n-1} \end{bmatrix} \quad \vec{f} = \begin{bmatrix} f_{-1} \\ f_0 \\ f_1 \\ \vdots \\ f_{n-1} \\ f_n \\ f_{n+1} \end{bmatrix}$$

Jeśli zastosujemy pomocnicze tablice (wystellarz dwie do zapamiętania h_i i g_i), to wyliczenie elementów macierzy kosztuje $(4n+15)$ ops + $(8n+22)$ opm. Macierz jest trójdiamondalna z dominującą przekątną, koszt rozwiązania układu równań wynosi $(3n+6)$ ops + $(5n+11)$ opm. Całkowity koszt algorytmu to $(7n+21)$ ops + $(13n+33)$ opm, zatem porównywalny z tym dla reprezentacji w lokalnych bazach potęgowych.

Dla węzłów równoodległych układ się znacznie upraszcza, otrzymujemy macierz

$$\frac{1}{6} \cdot \begin{bmatrix} 4 & 1 & & & & 0 \\ 1 & 4 & 1 & & & \\ & 1 & 4 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 4 & 1 \\ 0 & & & & 1 & 4 \end{bmatrix},$$

a koszt redukuje się do $(3n+6)$ ops + $(6n+14)$ opm (taniej niż dla lokalnych baz potęgowych).

11.4 Reszta interpolacyjna

Podobnie jak przy interpolacji wielomianowej wprowadzamy pojęcie reszty interpolacyjnej

$$r(x) = f(x) - S(x)$$

oraz błędu interpolacji

$$R = \sup_{x \in [a,b]} |r(x)| = \|r\|_{[a,b]}.$$

Twierdzenie 15. Niech f będzie klasą $C^2[a,b]$ oraz S będzie splajnem naturalnym interpolującym f w węzłach

$$a = x_0 < x_1 < \dots < x_n = b.$$

Wtedy

$$R \leq 5\|f''\|_{[a,b]} \max_{i=0,\dots,n-1} h_i^2,$$

gdzie $h_i = x_{i+1} - x_i$.

Zauważmy, że oszacowanie reszty jest najmniejsze dla węzłów równoodległych na przedziale $[a, b]$.

Przykład Wyznaczmy oszacowanie reszty dla zadania interpolacji funkcji $f(x) = \ln x$ w węzłach $1, e, e^2$.

Korzystamy z powyższego twierdzenia

$$|r(x)| \leq R \leq 5\|f''\|h^2,$$

gdzie h oznacza maksymalną odległość między sąsiednimi węzłami. Tutaj $h = e^2 - e$. Ponieważ

$$f''(x) = -\frac{1}{x^2}$$

to

$$\|f''\|_{[1,e^2]} = 1$$

i

$$|r(x)| \leq 5e^2(e-1)^2.$$

11.5 Zastosowania B-splajnów w grafice komputerowej

B-splajny znalazły zastosowanie w modelowaniu krzywych i powierzchni. Można mianowicie zamiast współczynników rzeczywistych b_i w przedstawieniu

$$S(x) = \sum_{i=-k}^{n-1} b_i B_i^{(k)}(x)$$

rozuwać punkty płaszczyzny $d_i \in \mathbb{R}^2$ i otrzymać krzywą na płaszczyźnie

$$s(x) = \sum_{i=-k}^{n-1} d_i B_i^{(k)}(x).$$

Węzły są określone jak poprzednio. Parametr x należy do przedziału $[x_0, x_n]$. Punkty d_i nazywają się *punktami kontrolnymi*.

Algorytm de Boora działa tu dokładnie tak samo: wewnątrz pętli obliczamy kombinację wypukłą dwóch punktów, należy napisać osobne instrukcje dla obu współrzędnych.

Zastosowania w grafice krzywe B-sklejane zawdzięczają swym zaletom. Mają one między innymi możliwość wprowadzania lokalnych zmian: przesunięcie jednego punktu kontrolnego nie powoduje zmiany całej krzywej, lecz jedynie okolicy tego punktu. Można stosunkowo tanio dorzucać dodatkowe punkty, zagęszczając siatkę w miejscach o bardzo skomplikowanych kształtach. Pozwala to konstruować krzywe metodą kolejnych przybliżeń. Krzywa leży w otoczeniu wypukłej punktów kontrolnych (można sformułować nawet silniejszą właściwość), dzięki temu konstruowane krzywe interpolacyjne mają kształt zgodny z intuicyjnym oczekiwaniem.

Płaty powierzchni uzyskuje się wprowadzając dwa parametry u, v , dwa ciągi węzłów i prostokątną tablicę punktów kontrolnych. Osoby zainteresowane odsyłam do książki Przemysława Kiciaka „Podstawy modelowania krzywych i powierzchni z zastosowaniami w grafice komputerowej”.

12 Wielomiany trygonometryczne

12.1 Zespolone wielomiany trygonometryczne

Wielomianem trygonometrycznym zespolonym stopnia n nazywamy funkcję

$$T(x) = \sum_{k=0}^n c_k e^{ikx}, \quad c_n \neq 0.$$

W tym rozdziale i oznacza jednostkę urojoną. Współczynniki c_k są zespolone, zatem wielomian jest funkcją o argumentem rzeczywistym i wartościach zespolonych. Z przedstawienia

$$e^{ikx} = \cos(kx) + i \sin(kx)$$

wynika, że wielomian ten jest funkcją okresową o okresie 2π .

12.2 Reprezentacja zespolonych wielomianów trygonometrycznych

Najprościej reprezentować T zapamiętując współczynniki $c_k = a_k + ib_k$, czyli $n+1$ par liczb rzeczywistych $\{a_k, b_k\}_{k=0}^n$. Powstaje pytanie o własności numeryczne, uwarunkowanie tak postawionego zadania. Okazuje się, że miara skośności bazy $\{e^{ikx}\}_{k=0}^n$ jest rzędu n^2 , nie jest więc najgorzej.

12.3 Obliczanie wartości wielomianu zespolonego

Ponieważ T jest funkcją okresową o okresie 2π , to wystarczy umieć wyznaczać wartości T w przedziale $[0, 2\pi]$. Zatem zakładamy dalej, że $x \in [0, 2\pi]$.

Algorytm jest dość oczywisty. Obliczamy najpierw $z = \cos(x) + i \sin(x)$ i zapisujemy T w postaci: $T(x) = \sum_{k=0}^n c_k z^k$.

Dalej możemy się posłużyć algorytmem Hornera. Należy pamiętać, że zamiast działań rzeczywistych pojawiają się zespolone. Koszt $2n$ działań zespolonych, czyli $8n$ działań rzeczywistych. Jakość bardzo dobra — algorytm jest numerycznie poprawny.

12.4 Rzeczywiste wielomiany trygonometryczne

Rzeczywistym wielomianem trygonometrycznym stopnia nie większego niż N nazywamy funkcję rozpiętą na N początkowych elementach bazy:

$$1, \sin(x), \cos(x), \sin(2x), \cos(2x), \sin(3x), \dots$$

Zatem wielomian T można zapisać jako

$$T(x) = \sum_{k=0}^{\lfloor \frac{N}{2} \rfloor} (a_k \cos(kx) + b_k \sin(kx)),$$

gdzie współczynniki a_k, b_k są rzeczywiste oraz przyjmujemy $b_{\lfloor \frac{N}{2} \rfloor} = 0$ dla N parzystych. Przy takim zapisie współczynnik b_0 nie odgrywa żadnej roli, bo stoi przy $\sin(0 \cdot x) = 0$.

Jest to znowu funkcja okresowa o okresie 2π . W dalszej części wprowadzamy oznaczenie $n = \lfloor \frac{N}{2} \rfloor$ i rozpatrujemy wielomiany trygonometryczne postaci

$$T(x) = \sum_{k=0}^n (a_k \cos(kx) + b_k \sin(kx)), \quad \text{gdzie } a_k, b_k \in \mathbb{R}.$$

12.5 Reprezentacja rzeczywistych wielomianów trygonometrycznych

Najłatwiej przechowywać współczynniki a_k, b_k , czyli posłużyć się wyjściową bazą. Miara skośności tej bazy jest rzędu \sqrt{N} , więc bardzo dobra.

12.6 Obliczanie wartości wielomianu rzeczywistego

Algorytm obliczania wyrazu za wyrazem jest dość kosztowny — potrzeba $4n$ działań arytmetycznych oraz $2n$ obliczeń funkcji trygonometrycznych. Te ostatnie kosztują około 30 działań każda.

Pierwsze ulepszenie polega na obliczeniu $\sin(x)$ oraz $\cos(x)$, a następnie potęgowaniu liczby zespolonej $z = \cos(x) + i \sin(x)$. Wtedy:

$$\begin{aligned} z^2 &= \cos(2x) + i \sin(2x), \\ z^3 &= \cos(3x) + i \sin(3x), \\ z^4 &= \cos(4x) + i \sin(4x). \end{aligned} \dots$$

Kolejne potęgi z uzyskuje się wymnażając dwie liczby zespolone kosztem 6 działań. Wtedy całkowity koszt algorytmu wynosi $10n$ działań + koszt jednokrotnego policzenia sinusa i cosinusa. Okazuje się, że można obliczać wartości wielomianu trygonometrycznego jeszcze taniej.

12.6.1 Algorytm Goertzel'a

Zauważmy, że sumę

$$T(x) = \sum_{k=0}^n (a_k \cos(kx) + b_k \sin(kx))$$

można rozdzielić na dwie:

$$T(x) = \sum_{k=0}^n a_k \cos(kx) + \sum_{k=0}^n b_k \sin(kx).$$

Wystarczy więc, że umiemy obliczyć każdą z nich. Zajmiemy się więc zadaniem obliczania sum postaci:

$$C(x) = \sum_{k=0}^n d_k \cos(kx), \quad S(x) = \sum_{k=0}^n d_k \sin(kx),$$

gdzie współczynniki d_k są rzeczywiste. Zauważmy, że $C(x)$ oraz $S(x)$ są odpowiednio częścią rzeczywistą i urojoną wielomianu zmiennej zespolonej

$$w(z) = \sum_{k=0}^n d_k z^k$$

w punkcie $u = \cos(x) + i \sin(x)$, czyli

$$w(u) = C(x) + iS(x), \quad \text{dla } u = \cos(x) + i \sin(x).$$

Podzielmy wielomian w przez trójmian kwadratowy

$$\begin{aligned} (z - u)(z - \bar{u}) &= z^2 - (u + \bar{u}) + |u|^2 \\ &= z^2 - 2z \cos(x) + 1, \end{aligned}$$

otrzymamy:

$$w(z) = (z^2 - 2z \cos(x) + 1) \sum_{k=2}^n e_k z^{k-2} + e_1 z + e_0.$$

Jeśli obliczymy teraz wartość w w punkcie u to otrzymamy, że jest ona równa reszcie z powyższego dzielenia w punkcie u :

$$w(u) = e_1(u) + e_0 = (e_1 \cos(x) + e_0) + ie_1 \sin(x).$$

Algorytm Goertzel'a polega na wyznaczeniu tej reszty z dzielenia. Musimy zatem uzyskać zależność między e_k oraz d_k .

$$\begin{aligned} w(z) &= \sum_{k=0}^n e_k z^k - \sum_{k=2}^n 2 \cos(x) e_k z^{k-1} + \sum_{k=2}^n e_k z^{k-2} \\ &= \sum_{k=0}^n e_k z^k - \sum_{k=1}^{n-1} 2 \cos(x) e_{k+1} z^k + \sum_{k=0}^{n-2} e_{k+2} z^k \\ &= \sum_{k=0}^n (e_k - 2 \cos(x) e_{k+1} + e_{k+2}) + 2 \cos(x) e_1, \end{aligned}$$

przy czym $e_{n+1} = e_{n+2} = 0$. Uzyskujemy stąd zależność rekurencyjną:

$$\begin{aligned} e_{n+1} &= e_{n+2} = 0, \\ e_k &= d_k + 2 \cos(x) e_{k+1} - e_{k+2}, \quad k = n, n-1, \dots, 1, \\ e_0 &= (d_0 + 2 \cos(x) e_1 - e_2) - 2 \cos(x) e_1 \end{aligned}$$

Zatem

$$C(x) = e_1 \cos(x) + e_0, \quad S(x) = e_1 \sin(x).$$

Można już teraz zapisać algorytm:

```
c=cos(x);
e[n+1]=e[n+2]=0;
for(k=n;k>=0;k--)
    e[k]=d[k]+2*c*e[k+1]-e[k+2];
C=e[0]-c*e[1];
S=e[1]*sin(x);
```

Zauważmy, że wystarczy używać trzech zmiennych zamiast tablicy e oraz $2 * c$ należy obliczyć przed pętlą.

```
c=cos(x);
c2=2*c;
e0=e1=0;
for(k=n;k>=0;k--)
{
    pom=e0
    e0=d[k]+c2*e0-e1;
    e1=pom;
}
C=e0-c*e1;
S=e1*sin(x);
```

Koszt wynosi $3n + 4$ działań. Pamiętajmy, że algorytm trzeba wykonać dwukrotnie, raz dla $d_k = a_k$ i drugi dla $d_k = b_k$, razem razem otrzymujemy $6n + 8$ działań. Algorytm jest tańszy niż poprzedni.

Jakość jednak nie zawsze jest zadowalająca. Można udowodnić numeryczną stabilność tego algorytmu, ale jedynie dla argumentów, które nie leżą w otoczeniu $0, \pi, 2\pi$. Problem pochodzi stąd, że używamy zamiast x danej pośredniej $c=\cos(x)$. Jeśli x jest bliskie 0 , to jesteśmy w stanie obliczyć c z błędem bezwzględnym $|f(\cos(x)) - \cos(x)| \leq 3\nu$ i jest to najlepsze oszacowanie, jakie jesteśmy w stanie uzyskać. Zatem informacja o x się zacięta — w okolicy zera funkcja cosinus ma pochodną bliską zeru i jest wiele argumentów spełniających powyższe oszacowanie.

Sprawę tą wyjaśnił Reinch i podał modyfikację algorytmu, uzyskując algorytm numerycznie stabilny. W algorytmie tym zamiast x pojawia się jedna z danych pośrednich: $-4 \sin^2(\frac{x}{2})$ lub $4 \cos^2(\frac{x}{2})$, w zależności od wielkości x . Koszt tego algorytmu nieco większy: $4n + 8$ działań (po dwukrotnym wykonaniu $8n + 16$).

13 Interpolacja wielomianami trygonometrycznymi

13.1 Sformułowanie zadania

Danych mamy n równoodległych węzłów z przedziału $[0, 2\pi]$, oraz n odpowiadających im wartości zespolonych, dokładniej, tabelkę

$$\{x_j, y_j\}_{j=0}^{n-1}, \quad \text{gdzie } x_j = j \frac{2\pi}{n}, \quad y_j \in \mathbb{C}.$$

Należy znaleźć zespolony wielomian trygonometryczny $T(x) = \sum_{k=0}^{n-1} c_k e^{ikx}$ spełniający warunki interpolacyjne:

$$T(x_j) = y_j, \quad j = 0, \dots, n-1,$$

zatem rozwiązać układ równań:

$$y_j = \sum_{k=0}^{n-1} c_k e^{ikj \frac{2\pi}{n}} \quad j = 0, \dots, n-1,$$

ze względu na niewiadome c_k . Jest to układ równań liniowych o macierzy $A = [e^{ikj \frac{2\pi}{n}}]_{k,j=0}^{n-1}$. Macierz ta ma szczególne właściwości. Po pierwsze jest symetryczna i nieosobliwa. Po drugie, można udowodnić, że $A^{-1} = \frac{1}{n} \bar{A}$. Zatem układ $A\vec{c} = \vec{y}$ ma rozwiązanie $\vec{c} = \frac{1}{n} \bar{A} \vec{y}$, czyli

$$c_j = \frac{1}{n} \sum_{k=0}^{n-1} y_k e^{-ikj \frac{2\pi}{n}} \quad j = 0, \dots, n-1.$$

Zadaniem numerycznym jest tutaj wyznaczenie współczynników c_j .

13.2 Wyznaczanie współczynników wielomianu interpolacyjnego

Wprowadźmy oznaczenie:

$$w_n = e^{-i \frac{2\pi}{n}}.$$

Wtedy możemy zapisać:

$$\begin{aligned} c_j &= \frac{1}{n} \sum_{k=0}^{n-1} y_k w_n^{kj} \\ &= \frac{1}{n} \sum_{k=0}^{n-1} y_k (w_n^j)^k. \end{aligned}$$

Po obliczeniu najpierw wartości w_n , potem w_n^j , każdy ze współczynników c_j okazuje się być wartością zespolonego wielomianu o współczynnikach y_k , zapisanego w bazie potęgowej, w punkcie w_n^j . Zatem można n -krotnie zastosować schemat Hornera (w dziedzinie zespolonej) i kosztem około $8n^2$ działań obliczyć wszystkie współczynniki c_j .

Można to jednak zrobić znacznie taniej — stosując algorytm FFT. Założymy obecnie, że $n = 2^p$. Zauważmy, że

$$\begin{aligned} c_j &= \frac{1}{n} \sum_{k=0}^{n-1} y_k w_n^{kj} \\ &= \frac{1}{n} \sum_{k=0}^{\frac{n}{2}-1} y_{2k} w_n^{2jk} + \frac{1}{n} \sum_{k=0}^{\frac{n}{2}-1} y_{2k+1} w_n^{j(2k+1)} \\ &= \frac{1}{n} \sum_{k=0}^{\frac{n}{2}-1} y_{2k} (w_n^2)^{jk} + w_n^j \frac{1}{n} \sum_{k=0}^{\frac{n}{2}-1} y_{2k+1} (w_n^2)^{jk}. \end{aligned}$$

Dalej, stosując definicję w_n stwierdzamy, że:

$$w_n^2 = e^{-2i \frac{2\pi}{n}} = w_{\frac{n}{2}}.$$

Zatem

$$c_j = \frac{1}{n} \sum_{k=0}^{\frac{n}{2}-1} y_{2k} w_{\frac{n}{2}}^{jk} + w_n^j \frac{1}{n} \sum_{k=0}^{\frac{n}{2}-1} y_{2k+1} w_{\frac{n}{2}}^{jk}.$$

Przyjmując oznaczenia:

$$g_j = \frac{1}{n} \sum_{k=0}^{\frac{n}{2}-1} y_{2k} w_{\frac{n}{2}}^{jk}, \quad h_j = \frac{1}{n} \sum_{k=0}^{\frac{n}{2}-1} y_{2k+1} w_{\frac{n}{2}}^{jk},$$

otrzymujemy, że

$$c_j = g_j + w_n^j h_j,$$

g_j oraz h_j mają podobną postać j do c_j : zmienia się jedynie liczba elementów do zsumowania oraz współczynniki. Możliwe więc jest zastosowanie rekurencji. Dodatkowo, zauważmy, że

$$\begin{aligned} g_{j+\frac{n}{2}} &= \frac{1}{n} \sum_{k=0}^{\frac{n}{2}-1} y_{2k} w_{\frac{n}{2}}^{(j+\frac{n}{2})k} \\ &= \frac{1}{n} \sum_{k=0}^{\frac{n}{2}-1} y_{2k} w_{\frac{n}{2}}^{jk} \\ &= g_j, \end{aligned}$$

podobnie $h_{j+\frac{n}{2}} = h_j$. Równocześnie można więc obliczać c_j oraz $c_{j+\frac{n}{2}}$. Możliwa jest taka organizacja obliczeń, by wszystkie wyniki pośrednie trzymać w jednej tablicy oraz zamienić wywołania rekurencyjne na iterację. Potrzeba wtedy dokonać odpowiedniej permutacji danych wejściowych lub wyników.

Łączny koszt obliczenia wszystkich współczynników c_j jest rzędu $n \log_2 n$. Algorytm jest numerycznie stabilny.

Aproksymacja

Dorota Dąbrowska, UKSW

2017/18

Spis treści

14 Klasyczne zadanie aproksymacji	74
14.1 Sformułowanie zadania	74
14.2 Istnienie i jednoznaczność rozwiązania	74
14.3 Własności elementów optymalnych i błędu aproksymacji	75
14.4 Aproksymacja a interpolacja	75
14.5 Zbieżność ciągu zadań aproksymacji	75
14.6 Inne sposoby postawienia zadania aproksymacji	75
15 Przestrzenie unitarne i wielomiany ortogonalne	77
15.1 Definicja przestrzeni unitarnej	77
15.2 Ciągi ortogonalne	78
15.3 Algorytmy ortogonalizacyjne	78
15.4 Wielomiany ortogonalne	79
15.5 Reguła trójczłonowa	80
15.6 Własności wielomianów ortogonalnych	81
16 Aproksymacja w przestrzeniach unitarnych	81
16.1 Sformułowanie zadania aproksymacji w przestrzeniach unitarnych	81
16.2 Twierdzenie o elemencie optymalnym	81
16.3 Konstrukcja elementu optymalnego	82
16.4 Numeryczne rozwiązanie zadania aproksymacji w przestrzeniach unitarnych	82
16.5 Aproksymacja średniookwadratowa wielomianami — szczególny przypadek aproksymacji w przestrzeniach unitarnych	83
16.6 Ciągi podprzestrzeni przestrzeni unitarnej . . .	83
17 Aproksymacja jednostajna	84
17.1 Sformułowanie zadania	84
17.2 Twierdzenie Czebyszewa o alternansie	84
17.3 Aproksymacja w przestrzeniach funkcji określonych na skończonej liczbie punktów	86
17.3.1 Przypadek $V = \Pi_{n-1}$	87

14 Klasyczne zadanie aproksymacji

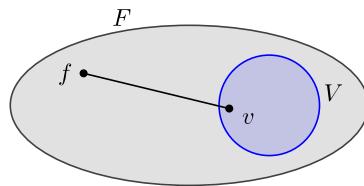
Na poprzednich wykładach rozważyliśmy zadania interpolacji. Wszystkie służyły temu by pewną funkcję trudną obliczeniowo, bądź znaną jedynie na podstawie próbek, przybliżyć inną funkcją obliczeniowo łatwą.

Interpolacja to nie jedyny sposób przybliżania funkcji. Do kładniejszym, lecz zwykle też droższym, jest aproksymacja.

14.1 Sformułowanie zadania

Dana jest przestrzeń unormowana¹ F oraz jej podprzestrzeń $V \subset F$, która ma skończony wymiar. Wybierzmy $f \in F$. Szukamy takiego $v_f \in V$, że

$$\|f - v_f\| = \inf_{v \in V} \|f - v\|.$$



Równość powyższą można interpretować następująco. Norma służy do obliczania odległości między elementami przestrzeni F . Dla każdego elementu v z V badamy jego odległość od f i wybieramy najmniejszą. Szukany v_f jest tym elementem V , który minimalizuje odległość od f .

Definicja 1. Element $v_f \in V$ taki, że

$$\|f - v_f\| = \inf_{v \in V} \|f - v\|$$

nazywamy *elementem optymalnym dla f względem V* , a

$$e(f, V) = \|f - v_f\| = \inf_{v \in V} \|f - v\|$$

błędem aproksymacji f względem V .

Zadanie aproksymacji podsumowuje poniższa tabela.

Dane:	F , $\ \cdot\ $,
	$V \subset F$, $\dim V < \infty$,
	$f \in F$
Wyniki:	$v_f \in V$ taki, że $\ f - v_f\ = \inf_{v \in V} \ f - v\ $ oraz $e(f, V) = \ f - v_f\ $

F jest przestrzenią funkcji rzeczywistych lub zespolonych („trudnych”), które chcemy przybliżać, a podprzestrzeń V funkcji („łatwy”), którymi będziemy przybliżali. Elementami przestrzeni V , tak jak przy interpolacji, powinny być funkcje, dla których tanio i dobrze można obliczać wartości. Norma wyznacza sposób mierzenia odległości między funkcjami. Zwykle jest to norma supremum na pewnym zbiorze lub normy generowane przez iloczyn skalarny.

¹Przestrzeń unormowana to taka przestrzeń liniowa, w której określona jest norma.

14.2 Istnienie i jednoznaczność rozwiązania

W punkcie tym odpowiemy na następujące pytania.

1. Czy element optymalny zawsze istnieje?
2. Czy jest wyznaczony jednoznacznie?
3. Czy v_f zależy od f w sposób ciągły?²

Odpowiedź na pytanie pierwsze jest twierdząca. Okazuje się tu kluczowym założeniem o skończonym wymiarze V . Dowód pominię.

Jednoznaczność nie zawsze jest gwarantowana. Przyjmijmy następującą definicję.

Definicja 2. Przestrzeń F nazywamy *silnie wypukłą*, jeśli dla dowolnych $f, g \in F$ takich, że $f \neq g$ oraz $\|f\| = \|g\| = 1$ zachodzi

$$\left\| \frac{f+g}{2} \right\| < 1.$$

Okazuje się, że własność ta opisuje przestrzenie, w których rozwiązania zadania aproksymacji są zawsze jednoznaczne.

Twierdzenie 1. Następujące warunki są równoważne:

1. przestrzeń F jest silnie wypukła,
2. dla dowolnej podprzestrzeni skończonej wymiarowej $V \subset F$ i dla każdego $f \in F$ istnieje dokładnie jeden $v_f \in V$ taki, że v_f jest elementem optymalnym dla f względem V .

Zatem w przestrzeniach F , które są silnie wypukłe, niezależnie jak wybierzemy $f \in F$ oraz skończenie wymiarową $V \subset F$, zadanie aproksymacji ma jednoznaczne rozwiązanie. Jeśli zaś przestrzeń F nie jest silnie wypukła, to jednoznaczność nie musi zachodzić. Niemniej czasami daje się tak dobrać V , by rozwiązania były jednoznaczne.

Pozostaje odpowiedź na pytanie trzecie. Okazuje się, że jeśli F i V mają taką własność, że dla dowolnego f element optymalny jest wyznaczony jednoznacznie, to v_f zależy od f w sposób ciągły.

Twierdzenie 2. Niech F i V będą takie, że dla dowolnego $f \in F$ element optymalny v_f jest wyznaczony jednoznacznie. Wtedy odwzorowanie

$$\begin{aligned} \phi : F &\rightarrow V, \\ \phi(f) &= v_f \end{aligned}$$

jest ciągłe.

²To pytanie można sformułować też tak: czy niewielkie zmiany elementu, który aproksymujemy (np. na skutek błędów reprezentacji) powodują niewielkie zmiany elementu optymalnego? Jest to istotne pytanie przy obliczeniach numerycznych.

14.3 Własności elementów optymalnych i błędu aproksymacji

Ustalmy $F, V \subset F, f \in F$. Wiemy, że nie zawsze zbiór

$$V_f = \{v_f \in V : \|f - v_f\| = e(f, V)\}$$

jest jednoelementowy, bo nie zawsze element optymalny jest wyznaczony jednoznacznie. Zbiór V_f ma jednak następującą cechę.

Fakt 1. Zbiór V_f jest wypukły.³

Błąd aproksymacji ma zbliżone własności do normy, choć nią nie jest.

Fakt 2. Niech \mathbb{K} będzie ciałem, nad którym jest zdefiniowana przestrzeń F . Zachodzą własności

1. $e(f, V) = 0$ wtedy i tylko, gdy $f \in V$,
2. $\forall_{\alpha \in \mathbb{K}} \forall_{f \in F} e(\alpha f, V) = |\alpha| e(f, V)$,
3. $\forall_{f, g \in F} e(f + g, V) = e(f, V) + e(g, V)$.

Przy ustalonym V można by $e(f, V)$ traktować jako normę w F , gdyby nie punkt pierwszy. W definicji normy wymaga się, by norma była równa zeru tylko wtedy, gdy element jest równy zeru. Tu tak nie jest, ale powyższe własności upoważniają nazywanie $e(\cdot, V)$ pseudonormą.

14.4 Aproksymacja a interpolacja

Załóżmy, że F jest przestrzenią funkcji ciągłych o dziedzinie $B \subset \mathbb{R}$, który jest zwarty, czyli domknięty i ograniczony. W tej przestrzeni tradycyjnie określamy normę supremum $\|\cdot\|$ na B :

$$\|f\| = \sup_{x \in B} |f(x)|.$$

Niech V będzie skończoną podprzestrzenią F , a x_0, x_1, \dots, x_n różnymi węzłami z B . Ustalmy funkcję $f \in F$.

Załóżmy, że istnieje dokładnie jeden $w \in V$ interpolujący f w zadanych węzłach. Ponieważ $w \in V$, to błąd interpolacji (liczony na B) spełnia

$$R = \|f - w\| \geq \inf_{v \in V} \|f - v\|.$$

Niech $v_f \in V$ będzie elementem optymalnym dla f względem V . Z definicji

$$e(f, V) = \|f - v_f\| = \inf_{v \in V} \|f - v\|.$$

Stąd

$$R \geq e(f, V).$$

Zatem interpolacja nie może dać lepszych wyników niż aproksymacja. W przypadku interpolacji Lagrange'a w węzłach Czebyszewa okazuje się, że interpolacja nie jest znacznie gorsza od aproksymacji, o ile n nie jest zbyt duże. Jednak dla węzłów równoodległych już tak nie jest.

³Zbiór A jest wypukły jeśli dla dowolnych $a_1, a_2 \in A$ i dowolnego $\alpha \in (0, 1)$ zachodzi $\alpha a_1 + (1 - \alpha) a_2 \in A$.

14.5 Zbieżność ciągu zadań aproksymacji

Zastanówmy się obecnie nie nad jednym zadaniem aproksymacji, ale nad ciągiem takich zadań. Dokładniej dana jest unormowana przestrzeń F , element $f \in F$ oraz ciąg skończenie wymiarowych podprzestrzeni przestrzeni F

$$V_1 \subsetneq V_2 \subsetneq V_3 \subsetneq V_4 \subsetneq \dots$$

Symbol \subsetneq oznacza inkluzję właściwą, czyli wykluczającą równość zbiorów. Rozważamy ciąg zadań zdefiniowanych przez te podprzestrzenie. Nietrudno zauważać, że

$$e(f, V_n) = \inf_{v \in V_n} \|f - v\| \geq \inf_{v \in V_{n+1}} \|f - v\| = e(f, V_{n+1}),$$

zatem wraz ze wzrostem n jakość aproksymacji się nie pogarsza. Intuicyjnie wydaje się, że błąd powinien być coraz mniejszy. Czy tak jest? A nawet jeśli błąd się zmniejsza, to czy dąży do zera? Czy dąży do zera szybko, czy wolno? Odpowiedź na ostatnie pytanie częściowo daje następujące twierdzenie.

Twierdzenie 3. (Berstaina) Niech F będzie przestrzenią Banacha.⁴ Dany jest ciąg skończenie wymiarowych podprzestrzeni przestrzeni F

$$V_1 \subsetneq V_2 \subsetneq V_3 \subsetneq V_4 \subsetneq \dots$$

oraz ciąg liczb rzeczywistych

$$a_1 \geq a_2 \geq a_3 \geq \dots$$

zbieżny do zera. Wtedy istnieje takie $f \in F$, że

$$e(f, V_k) = a_k \text{ dla } k = 1, 2, 3, \dots$$

Twierdzenie mówi o istnieniu „złośliwych” elementów, dla których mimo stosowania coraz większych podprzestrzeni dowolnie wolno zmniejsza się błąd aproksymacji (choć dąży dla nich do zera). Rzeczywiście, ciąg a_n jest dowolnym nierosnącym zbiegającym do zera. W szczególności może mieć pierwsze milion wyrazów równych 1000. Z praktycznego punktu widzenia taka wolna zbieżność jest tak samo zła jak brak zbieżności.

Zwróćmy uwagę, że twierdzenie to nie odpowiada na pytanie, czy dla każdego układu podprzestrzeni, które spełniają zapisane inkluzje i dowolnego elementu błąd aproksymacji dąży do zera.. W ogólności tak nie jest. Twierdzenie informuje jedynie, że istnieją takie elementy, ale zbieżność jest dowolnie wolna.

14.6 Inne sposoby postawienia zadania aproksymacji

Zadanie aproksymacji można też postawić w nieco odmienej formie. Mianowicie zamiast pytać o element optymalny

⁴Przestrzenie Banacha są przestrzeniami unormowanymi posiadającymi własność, że każdy ciąg Cauchy'ego elementów tej przestrzeni jest zbieżny. Ciąg a_n jest ciągiem Cauchy'ego jeśli dla dowolnego $\epsilon > 0$ istnieje $n_0 \in \mathbb{N}$, że dla wszystkich $m, n \geq n_0$ zachodzi $\|a_n - a_m\| < \epsilon$. Przykładami przestrzeni Banacha są np. \mathbb{R}^n , $C(B)$ dla B zwartej (czyli przestrzeń funkcji ciągłych na domkniętym i ograniczonym zbiorze B).

v_f , zadawalamy się elementem $v_{f,\varepsilon}$ być może gorszym, który jest odległy od aproksymowanego elementu f nie bardziej niż zadane $\varepsilon > 0$, czyli spełnia nierówność

$$\|f - v_{f,\varepsilon}\| \leq \varepsilon.$$

Podsumujmy zadanie w tabeli.

Dane: $F, \|\cdot\|,$
 $V \subset F, \dim V < \infty,$
 $f \in F,$
 $\varepsilon > 0$

Wyniki: $v_{f,\varepsilon} \in V$ taki, że $\|f - v_{f,\varepsilon}\| \leq \varepsilon$

Zadanie to można uznać za łatwiejsze od klasycznego, bo zając element optymalny v_f dla f względem V wystarczy zaproponować, by on był rozwiązaniem, czyli $v_{f,e} = v_f$. Zauważmy jednak, że jest to dobre postępowanie tylko gdy $e(f, V) \leq \varepsilon$, bo inaczej zachodziłoby

$$\|f - v_{f,\varepsilon}\| = \|f - v_f\| = e(f, V) > \varepsilon.$$

Oczywiście w tym przypadku znalezienie $v_{f,e}$ nie jest w ogóle możliwe. W pewnym więc sensie to drugie sformułowanie daje zadanie „trudniejsze”, bo nie zawsze mamy rozwiązanie. Tu bowiem żądamy, by znaleźć element dobrze przybliżający, a element optymalny mimo, że jest najlepszy, wcale nie musi tego dobrze robić.

Dla sformułowania przedstawionego w tym punkcie opracowuje się inne, tańsze algorytmy, niż wyszukujące element optymalny. Jeśli przyjmiemy na przykład odpowiednie założenia o przestrzeni F , a $V = \Pi_n$, to możemy się posłużyć interpolacją Lagrange'a w węzłach Czebyszewa, o ile ε nie będzie zbyt małe.

Zadaniem z pewnością trudniejszym jest następujące. Dana jest przestrzeń unormowana F i ciąg podprzestrzeni skończenie wymiarowych

$$V_1 \subsetneq V_2 \subsetneq V_3 \subsetneq V_4 \subsetneq \dots$$

mających taką własność, że dla dowolnego $f \in F$ zachodzi $\lim_{n \rightarrow \infty} e(f, V_n) = 0$. Dla ustalonego $\varepsilon > 0$ i $f \in F$ należy znaleźć n oraz $v_{f,n} \in V_n$ takie, że

$$\|f - v_{f,n}\| \leq \varepsilon,$$

zadanie to jest ujęte w poniższej tabeli.

Dane: $F, \|\cdot\|,$
 $V_1 \subsetneq V_2 \subsetneq V_3 \subsetneq \dots$ takie, że
1. $V_n \subset F$,
2. $\dim V_n < \infty$,
3. $\forall g \in F \quad \lim_{n \rightarrow \infty} e(g, V_n) = 0$

$f \in F,$
 $\varepsilon > 0$

Wyniki: n oraz $v_{f,n} \in V_n$ taki, że $\|f - v_{f,n}\| \leq \varepsilon$

Oczywiście można w celu rozwiązania tego zadania rozpatrywać po kolej elementy optymalne dla f względem przestrzeni V_n tak długo, aż żądana własność będzie spełniona. Może jednak okazać się, że takie postępowanie jest zbyt drogie i należy szukać innych algorytmów.

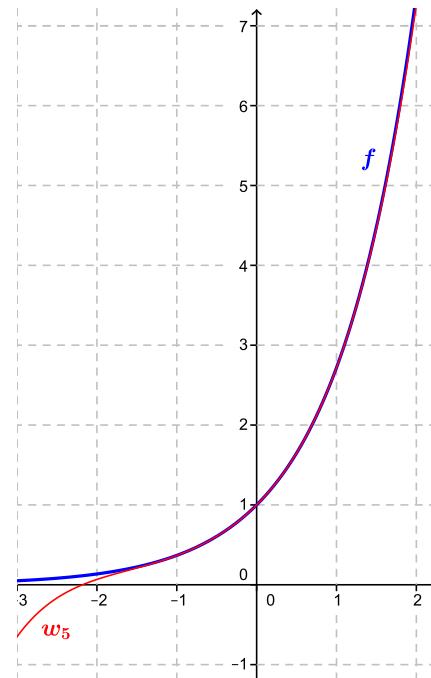
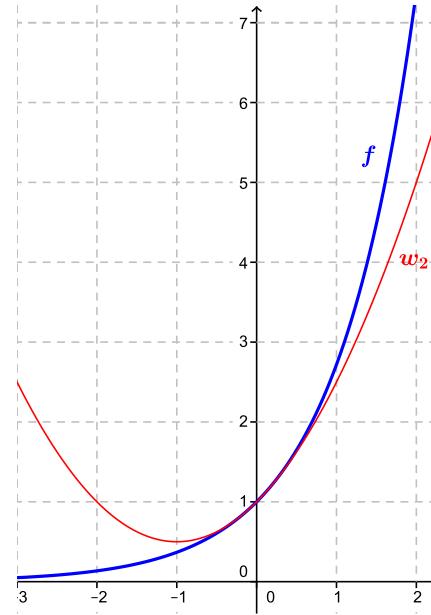
Przykład Rozważmy przestrzeń $F = C[a, b]$ z normą supremum na $[a, b]$ i $V_n = \Pi_n$. Chcemy obliczać wartości funkcji $f(x) = e^x$ na przedziale $[a, b]$. Dysponujemy jedynie czterema działaniami arytmetycznymi i skończonym czasem. Przypomnijmy, że

$$f(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!}.$$

Jednym ze sposobów rozwiązania tego zadania aproksymacji jest zastąpienie funkcji f skońzoną sumą

$$w_n(x) = \sum_{k=0}^n \frac{x^k}{k!}.$$

Przybliżenie to będzie dobre, o ile n jest dostatecznie duże. Zauważmy, że lepsze przybliżenia uzyskujemy w otoczeniu zera, a im $|x|$ większe tym potrzeba większego n , by uzyskać tę samą jakość.



Zastąpiliśmy zatem funkcję e^x wielomianem $w_n(x)$, którego wartości potrafimy obliczać. Jeśli odpowiednio dobierzemy wartość n , to spełniony będzie warunek $\|f - w_n\| < \varepsilon$.

15 Przestrzenie unitarne i wielomiany ortogonalne

15.1 Definicja przestrzeni unitarnej

W przestrzeni liniowej można zdefiniować działanie na wektorach nazywane iloczynem skalarnym.

Definicja 3. Ustalmy przestrzeń liniową F nad \mathbb{R} . Funkcję, która każdym dwóm elementom f, g przestrzeni F przyporządkowuje liczbę oznaczaną przez (f, g) nazywamy *iloczynem skalarnym*, jeśli spełnione są trzy warunki⁵

1. $\forall_{f \in F} (f, f) \geq 0$, przy czym $(f, f) = 0 \iff f = 0$,
2. $\forall_{f,g \in F} (f, g) = (g, f)$,
3. $\forall_{f,g,h \in F} \forall_{\alpha,\beta \in \mathbb{R}} (f\alpha + g\beta, h) = (f, h)\alpha + (g, h)\beta$.

Definicja 4. Przestrzeń unitarną nazywamy przestrzeń liniową z iloczynem skalarnym. Normą generowaną przez iloczyn skalarny nazywamy normę zdefiniowaną następująco

$$\forall_{f \in F} \|f\| = \sqrt{(f, f)}.$$

Przykład Poniżej zamieszczam opisy przestrzeni unitarnych używanych na tym wykładzie.

1. W przestrzeni \mathbb{R}^n definiujemy „zwykły” iloczyn skalarny

$$\forall_{\vec{x}, \vec{y} \in \mathbb{R}^n} (\vec{x}, \vec{y}) = \sum_{i=1}^n x_i y_i.$$

2. W \mathbb{R}^n można iloczyn skalarny zdefiniować ogólniej. Wybieramy n liczb dodatnich $\rho_1, \rho_2, \dots, \rho_n$, nazywanych *wagami* i przyjmujemy

$$\forall_{\vec{x}, \vec{y} \in \mathbb{R}^n} (\vec{x}, \vec{y}) = \sum_{i=1}^n \rho_i x_i y_i.$$

3. Niech l^2 oznacza przestrzeń wszystkich ciągów rzeczywistych⁶ $a = (a_i)_{i=1}^\infty$, dla których $\sum_{i=1}^\infty |a_i|^2 < \infty$

$$l^2 = \{a : a = (a_i)_{i=1}^\infty, a_i \in \mathbb{R}, \sum_{i=1}^\infty |a_i|^2 < \infty\}$$

⁵Definicję tę łatwo uogólnić na przypadek przestrzeni liniowej nad ciałem \mathbb{C} . Warunek drugi musi być wówczas zastąpiony następującym $\forall_{f,g \in F} (f, g) = \overline{(g, f)}$.

⁶Można tę definicję uogólnić dla ciągów zespolonych. W iloczynie skalarnym należy wtedy dodać sprzężenie nad b_i . Występujący w sumie $\sum_{i=1}^\infty |a_i|^2$ moduł jest zbędny dla liczb rzeczywistych, ale konieczny w przypadku zespolonych.

i iloczyn skalarny jest określony przez równość

$$\forall_{a,b \in l^2} (a, b) = \sum_{i=1}^\infty a_i b_i.$$

4. Dany jest ciąg $\rho = (\rho_i)_{i=1}^\infty$ o wyrazach dodatnich. Niech l_ρ^2 oznacza przestrzeń⁷

$$l_\rho^2 = \{a : a = (a_i)_{i=1}^\infty, a_i \in \mathbb{R}, \sum_{i=1}^\infty \rho_i |a_i|^2 < \infty\}$$

wraz z iloczynem skalarnym

$$\forall_{a,b \in l^2} (a, b) = \sum_{i=1}^\infty \rho_i a_i b_i.$$

5. Elementami przestrzeni $l^2(X)$ są wszystkie funkcje rzeczywiste⁸ o dziedzinie X , która ma skończenie wiele elementów, czyli $X = \{x_1, x_2, \dots, x_N\}$ dla pewnego $N \geq 1$. Zatem

$$l^2(X) = \{f : f : X \rightarrow \mathbb{R}\},$$

a iloczynem skalarnym jest

$$\forall_{f,g \in l^2(X)} (f, g) = \sum_{i=1}^N f(x_i) g(x_i).$$

6. Uogólnimy poprzedni przykład definiując przestrzeń unitarną $l_\rho^2(X)$. Dany jest $X = \{x_1, x_2, \dots, x_N\}$ dla pewnego $N \geq 1$ oraz dodatnia funkcja wagowa $\rho : X \rightarrow \mathbb{R}_+$. Przyjmujemy⁹

$$l_\rho^2(X) = \{f : f : X \rightarrow \mathbb{R}\}$$

z iloczynem skalarnym

$$\forall_{f,g \in l^2(X)} (f, g) = \sum_{i=1}^N \rho(x_i) f(x_i) g(x_i).$$

7. Elementami przestrzeni $L^2(a, b)$ są wszystkie funkcje rzeczywiste¹⁰ o dziedzinie (a, b) ,¹¹ dla których istnieje $\int_a^b |f(x)|^2 dx$:

$$L^2(a, b) = \left\{ f : (a, b) \rightarrow \mathbb{R} : \int_a^b |f(x)|^2 dx < +\infty \right\},$$

a iloczynem skalarnym¹² jest

$$\forall_{f,g \in L^2(a, b)} (f, g) = \int_a^b f(x) g(x) dx.$$

⁷Jak poprzednio można tę definicję uogólnić dla ciągów zespolonych (w iloczynie skalarnym dodać sprzężenie nad b_i).

⁸Dla funkcji zespolonych dodać sprzężenie nad $g(x_i)$.

⁹jak wyżej

¹⁰Znów można rozpatrywać funkcje zespolone. Wtedy moduł w całce $\int_a^b |f(x)|^2$ jest niezbędny, a w iloczynie skalarnym należy dodać sprzężenie nad $g(x)$.

¹¹Dopuszczamy, by $a = +\infty$ lub $b = -\infty$.

¹²Okazuje się, że ta definicja nie jest poprawna, bo istnieje więcej niż jedna funkcja f , dla której $(f, f) = 0$. Aby temu zaradzić utożsamia się pewne funkcje. Jeśli zawężymy przestrzeń tylko do funkcji ciągłych problem znika.

8. Dana jest nieujemna funkcja $\rho : (a, b) \rightarrow \mathbb{R}_+ \cup \{0\}$, dla której całka $\int_a^b \rho^2(x) dx$ istnieje i jest dodatnia. Elementami przestrzeni $L_\rho^2(a, b)$ są funkcje o dziedzinie (a, b) , dla których istnieje $\int_a^b \rho(x)|f(x)|^2 dx$:

$$L_\rho^2(a, b) = \left\{ f : (a, b) \rightarrow \mathbb{R} : \int_a^b \rho(x)|f(x)|^2 dx < +\infty \right\},$$

a iloczynem skalarnym jest

$$\forall_{f,g \in L^2(a,b)} (f, g) = \int_a^b \rho(x)f(x)g(x) dx.$$

9. Ostatnie dwie definicje iloczynów skalarnych są dobre również w przestrzeniach: funkcji ciągłych na $[a, b]$, Π_n , Π_n , bo są one podprzestrzeniami $L^2(a, b)$ i $L_\rho^2(a, b)$.

15.2 Ciągi ortogonalne

Definicja 5. Niech F będzie przestrzenią unitarną. Dwa wektory $f, g \in F$ nazywamy *ortogonalnymi (prostopadłymi)* w F , jeśli $(f, g) = 0$.

Przykład Wektory $[1, 0, 0, 0]^T$ oraz $[0, 0, 1, 0]^T$ są ortogonalne w \mathbb{R}^4 ze zwykłym iloczynem skalarnym. Są też ortogonalne w \mathbb{R}^4 z iloczynem skalarnym z wagą, przy dowolnym wyborze wagi.

Przykład Wielomiany 1 oraz x nie są ortogonalne w $L^2[0, 1]$, gdyż

$$(1, x) = \int_0^1 1 \cdot x dx = \frac{1}{2},$$

ale są ortogonalne w $L^2(-1, 1)$ bo

$$(1, x) = \int_{-1}^1 1 \cdot x dx = 0.$$

Definicja 6. Ciąg f_1, f_2, \dots , (skończony lub nieskończony) niezerowych elementów z F nazywamy *ciągiem (układem) ortogonalnym* w przestrzeni unitarnej F , jeśli

$$(f_i, f_j) = 0 \text{ dla } i \neq j.$$

Jeśli ponadto

$$(f_i, f_i) = 1,$$

dla każdego i , to taki ciąg nazywamy *ciągiem (układem) ortonormalnym*.

Można udowodnić, że każdy układ ortogonalny jest liniowo niezależny.

Przykład Ciągiem ortogonalnym w przestrzeni \mathbb{R}^3 ze zwykłym iloczynem skalarnym jest

$$[0, 0, 2]^T, [4, 0, 0]^T, [0, 3, 0]^T.$$

Nie jest on ortonormalny. Ciągiem ortonormalnym jest

$$[0, 0, 1]^T, [1, 0, 0]^T, [0, 1, 0]^T.$$

Przykład Ciągiem ortogonalnym w przestrzeni \mathbb{R}^3 z iloczynem $(v, w) = \frac{1}{16}v_1w_1 + \frac{1}{9}v_2w_2 + \frac{1}{4}v_3w_3$ jest

$$[0, 0, 1]^T, [1, 0, 0]^T, [0, 1, 0]^T.$$

Nie jest on ortonormalny. Ciągiem ortonormalnym jest

$$[0, 0, 2]^T, [4, 0, 0]^T, [0, 3, 0]^T.$$

15.3 Algorytmy ortogonalizacyjne

Jeśli mamy dany w przestrzeni F dowolny układ liniowo niezależny f_1, f_2, \dots, f_n , to istnieje układ ortogonalny g_1, g_2, \dots, g_n , który rozpina tę samą przestrzeń

$$\text{span}\{f_1, f_2, \dots, f_n\} = \text{span}\{g_1, g_2, \dots, g_n\}.$$

Rozważmy zadanie znalezienia takiego układu.

Dane: f_1, f_2, \dots, f_n - elementy F

Wyniki: g_1, g_2, \dots, g_n takie, że

$$\text{span}\{f_1, f_2, \dots, f_n\} = \text{span}\{g_1, g_2, \dots, g_n\}$$

oraz $(g_i, g_j) = 0$ dla $i \neq j$

Układ ten można skonstruować np. metodą *ortogonalizacji Grama–Schmidta*. Elementy g_i określamy tu rekurencyjnie

$$\begin{aligned} g_1 &= f_1, \\ g_k &= f_k - \sum_{i=1}^{k-1} \frac{(f_k, g_i)}{(g_i, g_i)} g_i, \quad k = 2, \dots, n. \end{aligned}$$

Jeśli potrzebny jest układ ortonormalny, to wystarczy wziąć wektory

$$\frac{g_1}{\|g_1\|}, \frac{g_2}{\|g_2\|}, \dots, \frac{g_n}{\|g_n\|},$$

albo lepiej nieco zmodyfikować powyższy sposób tak, by „nabieżąco” normować g_k

$$\begin{aligned} g_1 &= \frac{f_1}{\|f_1\|}, \\ \begin{cases} p_k = f_k - \sum_{i=1}^{k-1} (f_k, g_i)g_i, \\ g_k = \frac{p_k}{\|p_k\|} \end{cases} & k = 2, \dots, n. \end{aligned}$$

Przykład Rozpatrzmy ciąg liniowo niezależny w \mathbb{R}^3

$$[2, 0, 0]^T, [3, 4, 0]^T, [1, 1, 1]^T.$$

Nie jest on ortogonalny. Zastosujmy metodę pierwszą, by otrzymać ciąg ortogonalny.

$$\begin{aligned} g_1 &= [2, 0, 0]^T, \\ g_2 &= [3, 4, 0]^T - \frac{([3, 4, 0]^T, g_1)}{(g_1, g_1)} g_1 \\ &= [3, 4, 0]^T - \frac{6}{4}[2, 0, 0]^T = [0, 4, 0]^T, \\ g_3 &= [1, 1, 1]^T - \frac{([1, 1, 1]^T, g_1)}{(g_1, g_1)} g_1 - \frac{([1, 1, 1]^T, g_2)}{(g_2, g_2)} g_2 \\ &= [1, 1, 1]^T - \frac{2}{4}[2, 0, 0]^T - \frac{4}{16}[0, 4, 0]^T = [0, 0, 1]^T. \end{aligned}$$

Metoda druga da ciąg ortonormalny.

$$\begin{aligned} g_1 &= [1, 0, 0]^T, \\ p_2 &= [3, 4, 0]^T - ([3, 4, 0]^T, g_1)g_1 \\ &= [3, 4, 0]^T - 3[1, 0, 0]^T = [0, 4, 0]^T, \\ g_2 &= [0, 1, 0]^T \\ p_3 &= [1, 1, 1]^T - ([1, 1, 1]^T, g_1)g_1 - ([1, 1, 1]^T, g_2)g_2 \\ &= [1, 1, 1]^T - [1, 0, 0]^T - [0, 1, 0]^T = [0, 0, 1]^T \\ g_3 &= [0, 0, 1]^T. \end{aligned}$$

Metodę ortonormalizacji Grama–Schmidta można łatwo za-programować. Założymy, że dana jest funkcja `skal` obliczająca iloczyn skalarny dwóch wektorów (lub jego przybliżenie). Ciąg wektorów f_i znajduje się w tablicy `f`, a obliczane wyniki g_i są zapisywane w tablicy `g`. Obie tablice są indeksowane od 1. Ponadto zdefiniowane są odpowiednie operatory `*`, `/` mnożenia i dzielenia wektora przez liczbę oraz `-` odejmowania dwóch wektorów.

```
g[1]=f[1]/sqrt(skal(f[1],f[1]));
for(k=2;k<=n;k++){
    g[k]=f[k];
    for(i=1;i<k;i++)
        g[k]=g[k]-skal(f[k],g[i])*g[i];
    g_k=g_k/sqrt(skal(g[k],g[k]));
}
```

Zasadniczym kosztem jest tu koszt obliczania iloczynów ska-larnych. Zauważmy, że w pętli wewnętrznej jest łącznie $k-1$ obliczeń iloczynów skalarnych, jedno jest za nią. Pętla ze-wewnętrzna wykonuje się dla $k = 2, 3, \dots, n$ zatem jest w niej $2+3+\dots+n$ iloczynów, dodatkowo jeden przed nią. Razem jest $1+2+3+\dots+n = \frac{n(n+1)}{2}$ obliczeń iloczynów skalarnych.

Analizę tego algorytmu należy osobno przeprowadzać dla każ-dej przestrzeni unitarnej. W \mathbb{R}^n lub \mathbb{C}^n , błędy zaokrągleń po-wodują, że ten algorytm jest numerycznie niestabilny, zatem nie można go stosować. Okazuje się jednak, że jeśli ten algo-rytm wykonamy dwukrotnie: pierwszy raz danymi będą wek-tory f_i , a drugi obliczone w pierwszym kroku g_i , to algorytm będzie stabilny w tych przestrzeniach, niemniej koszt zwiększy się dwukrotnie. Zamiast tego używa się więc innego algo-rytmu, tzw. zmodyfikowanego algorytmu Grama–Schmidta.

```
for(k=1;k<=n;k++){
    g[k]=f[k]/sqrt(skal(f[k],f[k]));
    for(i=k+1;i<=n;i++)
        f[i]=f[i]-skal(f[i],g[k])*g[k];
}
```

Koszt tego algorytmu wynosi $\frac{n(n+1)}{2}$ obliczeń iloczynów ska-larnych. Jakość należy badać osobno dla każdej przestrzeni. Udowodniono, że algorytm jest stabilny w \mathbb{R}^n , \mathbb{C}^n oraz w zastosowaniu do ortonormalizacji wielomianów w przestrzeni $L^2(a, b)$.

15.4 Wielomiany ortogonalne

Definicja 7. Ciąg wielomianów P_0, P_1, P_2, \dots , gdzie P_k jest wielomianem stopnia dokładnie k , nazywamy *ciągiem wielomianów ortogonalnych na przedziale (a, b)* z wagą ρ jeśli tworzą one ciąg ortogonalny w przestrzeni $L_\rho^2(a, b)$.

Definicja 8. Ciąg wielomianów P_0, P_1, P_2, \dots , gdzie P_k jest wielomianem stopnia dokładnie k , nazywamy *ciągiem wielomianów ortogonalnych na skończonym zbiorze X* z wagą ρ jeśli tworzą one ciąg ortogonalny w przestrzeni $l_\rho^2(X)$.

Aby skonstruować ciąg wielomianów ortogonalnych wystarczy zortogonalizować np. układ $1, x, x^2, \dots$ jedną z opisanych metod. Poniżej przedstawiam ciągi wielomianów ortogonalnych w wybranych przestrzeniach. Są to tzw. *klasyczne wielomiany ortogonalne*. Wyprowadzono dla nich jawne wzory rekurencyjne pozwalające wyznaczać kolejne wielomiany bez konieczności liczenia iloczynów skalarnych.

1. *Wielomiany Legendre'a* są ortogonalne w $L_\rho^2(-1, 1)$, gdzie $\rho(x) = 1$. Można udowodnić, że wielomiany te spełniają zależność rekurencyjną

$$\begin{aligned} P_0(x) &= 1, \\ P_1(x) &= x, \\ P_k(x) &= \frac{2k-1}{k}xP_{k-1}(x) - \frac{k-1}{k}P_{k-2}(x), \end{aligned}$$

dla $k = 2, 3, \dots$ oraz, że $(P_k, P_k) = \frac{2}{2k+1}$.

2. *Wielomiany Czebyszewa* tworzą ciąg ortogonalny w $L_\rho^2(-1, 1)$ z wagą $\rho(x) = \frac{1}{\sqrt{(1-x^2)}}$. Wielomiany te spełniają zależność rekurencyjną

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_k(x) &= 2xT_{k-1}(x) - T_{k-2}(x) \end{aligned}$$

dla $k = 2, 3, \dots$. Udowodniono, że $(T_0, T_0) = \pi$ oraz $(T_k, T_k) = \frac{\pi}{2}$ dla $k > 0$.

3. *Wielomiany Czebyszewa drugiego rodzaju* są ciągiem or-toogonalnym w $L_\rho^2(-1, 1)$ z wagą $\rho(x) = \sqrt{(1-x^2)}$ i

$$\begin{aligned} U_0(x) &= 1, \\ U_1(x) &= 2x, \\ U_k(x) &= 2xU_{k-1}(x) - U_{k-2}(x) \end{aligned}$$

dla $k = 2, 3, \dots$. Ponadto $(U_k, U_k) = \frac{\pi}{2}$ dla $k \geq 0$.

4. *Wielomiany Hermite'a* tworzą ciąg ortogonalny w $L_\rho^2(-\infty, \infty)$, waga to $\rho(x) = e^{-x^2}$. Wielomiany te spełniają zależność rekurencyjną

$$\begin{aligned} H_0(x) &= 1, \\ H_1(x) &= 2x, \\ H_k(x) &= 2xH_{k-1}(x) - (2k-2)H_{k-2}(x) \end{aligned}$$

dla $k = 2, 3, \dots$. Udowodniono, że $(H_k, H_k) = \sqrt{\pi}2^k k!$.

5. Wielomiany Laguerre'a są ortogonalne w $L_p^2(0, \infty)$ z wagą $\rho(x) = x^p e^{-x}$, gdzie $p \in (-1, \infty)$ jest parametrem. Można udowodnić, że zachodzi

$$\begin{aligned} L_0(x) &= 1, \\ L_1(x) &= -x + p + 1, \\ L_k(x) &= \left(-\frac{x}{k} + \frac{2k+p-1}{k}\right)L_{k-1}(x) - \frac{k+p-1}{k}L_{k-2}(x). \end{aligned}$$

dla $k = 2, 3, \dots$. Okazuje się, że $(L_k, L_k) = \frac{\Gamma(k+p+1)}{\Gamma(k+1)}$, gdzie Γ oznacza tzw. funkcję Gamma¹³ o definicji

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt \quad \text{dla } x > 0.$$

Jeśli p jest liczbą całkowitą, to $(L_k, L_k) = \frac{(k+p)!}{k!}$.

6. Wielomiany Jacobiego definiujemy przy użyciu dwóch parametrów $p, q \in (-1, \infty)$. Są one ortogonalne w przestrzeni $L_p^2(-1, 1)$, gdzie $\rho(x) = (1-x)^p(1+x)^q$. Spełniają one wzór rekurencyjny

$$\begin{aligned} P_0(x) &= 1, \\ P_1(x) &= \frac{(p+q+2)x + p - q}{2} \\ P_k(x) &= (\alpha_k x + \beta_k)P_{k-1}(x) - \gamma_k P_{k-2}(x), \end{aligned}$$

dla $k = 2, 3, \dots$, przy czym przyjmując $s = p + q$

$$\begin{aligned} \alpha_k &= \frac{(2k+s-1)(2k+s)}{2k(k+s)}, \\ \beta_k &= \frac{(p^2-q^2)(2k+s-1)}{2k(k+s)(2k+s-2)}, \\ \gamma_k &= \frac{(k+p-1)(k+q-1)(2k+s)}{k(k+s)(2k+s-2)}. \end{aligned}$$

Dodatkowo

$$(P_k, P_k) = \frac{2^{s+1}\Gamma(k+p+1)\Gamma(k+q+1)}{(2k+s+1)\Gamma(k+1)\Gamma(k+s+1)}.$$

15.5 Reguła trójczłonowa

Zauważmy, że wszystkie klasyczne wielomiany ortogonalne spełniają zależności rekurencyjne, które są podobne w formie: dla $k > 2$ każdy wielomian wyraża się przy pomocy dwóch swoich poprzedników. Okazuje się, że nie jest to przypadek — istnieją takie zależności dla dowolnego ciągu wielomianów ortogonalnych.

¹³Można definicję funkcji Γ uogólnić dla wszystkich $x \in \mathbb{C}$

$$\Gamma(x) = \lim_{n \rightarrow \infty} \frac{n! n^{x-1}}{x(x+1)(x+2)\dots(x+n-1)}.$$

Podstawowymi własnościami tej funkcji są

- (a) $\Gamma(x+1) = x\Gamma(x)$,
- (b) $\Gamma(n) = (n-1)!$ dla n całkowitych dodatnich,
- (c) $\Gamma(x)\Gamma(1-x) = \frac{\pi}{\sin(\pi x)}$,
- (d) $\Gamma(x)\Gamma(x+\frac{1}{2}) = \frac{\sqrt{\pi}}{2^{2x-1}}\Gamma(2x)$.

Można udowodnić, że jeśli ustalimy przestrzeń, to ciąg wielomianów ortogonalnych jest wyznaczony z dokładnością do stałych. Zatem jeśli określmy współczynnik przy najwyższej potędze dla każdego wielomianu z ciągu ortogonalnego

$$P_k(x) = d_k x^k + \dots, \quad k = 1, 2, \dots,$$

to taki ciąg jest wyznaczony jednoznacznie. Wtedy zachodzi tzw. reguła trójczłonowa

$$\begin{aligned} P_{-1}(x) &= 0, \\ P_0(x) &= d_0, \\ P_k(x) &= (\alpha_k x + \beta_k)P_{k-1}(x) + \gamma_k P_{k-2}(x), \end{aligned}$$

dla $k = 1, 2, \dots$, gdzie współczynniki $\alpha_k, \beta_k, \gamma_k$ wynoszą

$$\begin{aligned} \alpha_k &= \frac{d_k}{d_{k-1}}, & k = 1, 2, \dots, \\ \beta_k &= -\alpha_k \frac{(xP_{k-1}, P_{k-1})}{(P_{k-1}, P_{k-1})}, & k = 1, 2, \dots, \\ \gamma_k &= -\frac{\alpha_k}{\alpha_{k-1}} \frac{(P_{k-1}, P_{k-1})}{(P_{k-2}, P_{k-2})}, & k = 2, 3, \dots, \\ \gamma_1 & \text{jest dowolne (bo } P_{-1} \equiv 0\text{).} \end{aligned}$$

Występująca tu funkcja P_{-1} ma jedynie charakter pomocniczy — dzięki niej wzory są zwięzłe. Nie jest ona wyrazem ciągu wielomianów ortogonalnych. Zauważmy też, że współczynniki α_k, γ_k są zawsze różne od zera.

Wyznaczenie wszystkich wielomianów ortogonalnych stopnia nie większego niż n z reguły trójczłonowej wymaga obliczenia jedynie $2n$ iloczynów skalarnych — ponad $\frac{n}{4}$ razy mniej niż posługując się zmodyfikowanym algorytmem ortogonalizacji Grama–Schmidta.

Nie warto zapisywać wielomianów ortogonalnych w bazie potęgowej. Znajomość współczynników α_k, β_k i γ_k wystarczy do wyznaczania wartości wielomianów ortogonalnych w zadanym punkcie. Ogólniej, jeśli dowolny wielomian z przestrzeni Π_n zapisany jest w bazie, którą tworzą pewne wielomiany ortogonalne, to do obliczenia wartości tego wielomianu można stosować algorytm wynikający wprost z reguły trójczłonowej, lub tańszy — algorytm Clenshawa. Oba zostały już omówione przy okazji zadania obliczania wartości wielomianu zapisanego w bazie Czebyszewa. Należy je uogólnić.

Przykład Stosując regułę trójczłonową policzę zerowy, pierwszy i drugi wielomian ortogonalny na przedziale $(-1, 1)$ z wagą $\rho(x) = \sqrt{|x|}$. Przyjmę, że współczynniki przy najwyższej potędze równe są 1. Wtedy $\alpha_k = 1$ dla $k = 1, 2, \dots$

Zerowy wielomian ortogonalny to $P_0(x) = 1$. Aby policzyć P_1 wystarczy wyznaczyć

$$\beta_1 = -\frac{(xP_0, P_0)}{(P_0, P_0)}.$$

Ponieważ

$$\begin{aligned} (P_0, P_0) &= \int_{-1}^1 \sqrt{|x|} dx = 2 \int_0^1 \sqrt{x} dx = \frac{4}{3}, \\ (xP_0, P_0) &= \int_{-1}^1 x\sqrt{|x|} dx = 0, \end{aligned}$$

to $\beta_1 = 0$ i pierwszym wielomianem ortogonalnym jest $P_1(x) = x$.

Aby obliczyć P_2 trzeba znać

$$\beta_2 = -\frac{(xP_1, P_1)}{(P_1, P_1)} \text{ oraz } \gamma_2 = -\frac{(P_1, P_1)}{(P_0, P_0)}.$$

W tym celu obliczam

$$(P_1, P_1) = \int_{-1}^1 x^2 \sqrt{|x|} dx = 2 \int_0^1 x^{\frac{5}{2}} dx = \frac{4}{7}$$

$$(xP_1, P_1) = \int_{-1}^1 x^3 \sqrt{|x|} dx = 0.$$

Stąd $\beta_2 = 0$, $\gamma_2 = -\frac{3}{7}$ i drugim wielomianem ortogonalnym jest $P_2(x) = x^2 - \frac{3}{7}$.

15.6 Własności wielomianów ortogonalnych

Oto wybrane własności wielomianów ortogonalnych. Niech P_0, P_1, P_2, \dots tworzą ciąg wielomianów ortogonalnych w pewnej przestrzeni $L_\rho^2(a, b)$ lub $l_\rho^2(X)$ (zbiór X jest skończony).

1. Wielomiany P_0, \dots, P_n stanowią bazę Π_n .
2. Dowolny wielomian stopnia niższego niż k jest prostopadły do wielomianu P_k .
3. Ciąg wielomianów ortogonalnych jest wyznaczony z dokładnością do mnożników liczbowych.
4. Każdy ciąg wielomianów ortogonalnych w $l_\rho^2(X)$ jest skończony.
5. Niech P_0, P_1, \dots będzie ciągiem wielomianów ortogonalnych w $L_\rho^2(a, b)$. Wielomian P_k ma dokładnie k różnych pierwiastków. Są one rzeczywiste, jednokrotne i leżą w (a, b) .
6. (formuła Christoffler'a-Darboux) Wielomiany ortogonalne spełniają dla $x \neq y$ równość

$$\sum_{i=0}^k \frac{P_i(x)P_i(y)}{(P_i, P_i)} = \frac{P_{k+1}(x)P_k(y) - P_k(x)P_{k+1}(y)}{\alpha_{k+1}(x-y)(P_k, P_k)},$$

gdzie α_k jest odpowiednim współczynnikiem reguły trójczłonowej.

7. Rozpatrzmy ciąg wielomianów P_0, P_1, P_2, \dots ortogonalnych w $L_\rho^2(a, b)$. Zdefinujmy funkcję

$$\mathcal{K}_k(x, y) = \sum_{i=0}^k \frac{P_i(x)P_i(y)}{(P_i, P_i)}.$$

Wtedy prawdziwe są poniższe stwierdzenia.

- (a) Dla dowolnego $w \in \Pi_n$ zachodzi

$$w(x) = \int_a^b \rho(t) \mathcal{K}_k(x, t) w(t) dt.$$

- (b) Dla dowolnego y funkcja $K_k^{(y)}(x) = \mathcal{K}_k(x, y)$ jest wielomianem stopnia k (tzw. *wielomiany jądrowe*).
- (c) Dla ustalonego y wielomiany $K_0^{(y)}, K_1^{(y)}, K_2^{(y)}, \dots$ tworzą ciąg wielomianów ortogonalnych na (a, b) z wagą $\mu(x) = (x-y)\rho(x)$.
- (d) Przy ustalonym y wielomian $K_k^{(y)}$ ma w przestrzeni $L_\rho^2(a, b)$ normę równą $\sqrt{K_k^{(y)}(y)}$.
- (e) Ustalmy liczby y oraz c . W zbiorze

$$W = \{w \in \Pi_n : w(y) = c\}$$

najmniejszą normę w $L_\rho^2(a, b)$ ma wielomian $\frac{a}{K_n^{(y)}(y)} K_n^{(y)}(x)$.

16 Aproksymacja w przestrzeniach unitarnych

16.1 Sformułowanie zadania aproksymacji w przestrzeniach unitarnych

Przypomnijmy, że zadanie aproksymacji klasycznej jest postawione następująco.

Dane: $F, \|\cdot\|$,
 $V \subset F$, $\dim V < \infty$,
 $f \in F$

Wyniki: $v_f \in V$ taki, że $\|f - v_f\| = \inf_{v \in V} \|f - v\|$
oraz $e(f, V) = \|f - v_f\|$

Obecnie zawężymy nasze zainteresowanie do przypadku, gdy F jest przestrzenią unitarną z normą generowaną przez iloczyn skalarny

$$\forall f \in F \quad \|f\| = \sqrt{(f, f)}.$$

Zajmiemy się tylko przestrzeniami rzeczywistymi, choć wszystkie rozumowania łatwo przenosi się na przypadek zespolony.

Mozna udowodnić, że każda przestrzeń unitarna jest silnie wypukła. W myśl Twierdzenia 1 ze str. 74 dla dowolnej podprzestrzeni skończonej wymiarowej V i dowolnego $f \in F$ istnieje dokładnie jeden element optymalny $v^* \in F$ dla f . Dodatkowo z Twierdzenia 2 ze str. 74 mamy zagwarantowaną ciągłą zależność elementów optymalnych od funkcji, które aproksymują. Zatem zadanie aproksymacji w przestrzeniach unitarnych jest dobrze postawione.

16.2 Twierdzenie o elemencie optymalnym

Sformułujemy i udowodnimy twierdzenie, które charakteryzuje element optymalny dla f względem V w przestrzeniach unitarnych. Stanowi ono punkt wyjścia konstrukcji elementu optymalnego, tym samym algorytmów go wyznaczających.

Twierdzenie 4. Niech F będzie rzeczywistą przestrzenią unitarną oraz v^* będzie elementem optymalnym dla $f \in F$ względem V . Wtedy,

1. $\forall_{v \in V} (f - v^*, v) = 0$,
2. $e(f, V) = \sqrt{\|f\|^2 - \|v^*\|^2}$.

Dowód Weźmy dowolny element $w \in V$ oraz dowolną liczbę rzeczywistą t . Ponieważ V jest przestrzenią liniową, to $v^* + tw$ też należy do V . Ponieważ v^* jest elementem optymalnym dla f , więc z definicji

$$\forall_{v \in V} \|f - v^*\| \leq \|f - v\|,$$

w szczególności

$$\|f - v^*\| \leq \|f - (v^* + tw)\|.$$

Podnieśmy obie strony do kwadratu i skorzystajmy z definicji normy oraz własności iloczynu skalarnego

$$\begin{aligned} \|f - v^*\|^2 &\leq \|f - (v^* + tw)\|^2 \\ &= (f - (v^* + tw), f - (v^* + tw)) \\ &= ((f - v^*) - tw, (f - v^*) - tw) \\ &= (f - v^*, f - v^*) - 2(f - v^*, tw) + (tw, tw) \\ &= \|f - v^*\|^2 - 2t(f - v^*, w) + t^2\|w\|^2. \end{aligned}$$

Zatem

$$\|f - v^*\|^2 \leq \|f - v^*\|^2 - 2t(f - v^*, w) + t^2\|w\|^2,$$

a stąd jeśli $t > 0$, to

$$(f - v^*, w) \leq \frac{1}{2}t\|w\|^2,$$

a gdy $t < 0$, to

$$(f - v^*, w) \geq \frac{1}{2}t\|w\|^2.$$

Ponieważ wartość $|t|$ może być dowolnie mała, więc powyższe nierówności są możliwe do spełnienia jedynie gdy $(f - v^*, w) = 0$. Pierwsza część twierdzenia została udowodniona.

Druga część udowodnimy korzystając z definicji błędu $e(f, V) = \|f - v^*\|$ oraz wykazanej własności. Zauważmy, że

$$\|f - v^*\|^2 = (f - v^*, f - v^*) = (f - v^*, f) - (f - v^*, v^*).$$

Ponieważ dla dowolnego $v \in V$ zachodzi $(f - v^*, v) = 0$, więc również $(f - v^*, v^*) = 0$. Zatem

$$\begin{aligned} \|f - v^*\|^2 &= (f - v^*, f) \\ &= (f, f) - (v^*, f - v^* + v^*) \\ &= \|f\|^2 - (v^*, f - v^*) - (v^*, v^*) \\ &= \|f\|^2 - \underbrace{(f - v^*, v^*)}_{0} - \|v^*\|^2 \end{aligned}$$

Stąd

$$e(f, V) = \sqrt{\|f\|^2 - \|v^*\|^2}.$$

16.3 Konstrukcja elementu optymalnego

Konstrukcja elementu optymalnego wynika z udowodnionego twierdzenia. Oznaczmy wymiar V przez n i wybierzmy bazę v_1, v_2, \dots, v_n przestrzeni V . Każdy element v przestrzeni V daje się jednoznacznie przedstawić w tej bazie. Aby wyznaczyć v^* wystarczy obliczyć współczynniki α_i takie, że

$$v^* = \alpha_1 v_1 + \alpha_2 v_2 + \dots + \alpha_n v_n = \sum_{i=0}^n \alpha_i v_i.$$

Ponieważ dla dowolnego $v \in V$ zachodzi $(f - v^*, v) = 0$, więc prawdziwe są równości:

$$\begin{aligned} (f - v^*, v_1) &= 0, \\ (f - v^*, v_2) &= 0, \\ &\vdots \\ (f - v^*, v_n) &= 0. \end{aligned}$$

Z własności iloczynu skalarnego są one równoważne następującym

$$\begin{aligned} (v^*, v_1) &= (f, v_1), \\ (v^*, v_2) &= (f, v_2), \\ &\vdots \\ (v^*, v_n) &= (f, v_n). \end{aligned}$$

Podstawiając $v^* = \alpha_1 v_1 + \alpha_2 v_2 + \dots + \alpha_n v_n$ otrzymujemy układ równań

$$\left\{ \begin{array}{l} a_1(v_1, v_1) + a_2(v_2, v_1) + \dots + a_n(v_n, v_1) = (f, v_1), \\ a_1(v_1, v_2) + a_2(v_2, v_2) + \dots + a_n(v_n, v_2) = (f, v_2), \\ \vdots \\ a_1(v_1, v_n) + a_2(v_2, v_n) + \dots + a_n(v_n, v_n) = (f, v_n). \end{array} \right.$$

Niewiadomymi są tu współczynniki α_i . Można udowodnić, że układ ten ma jednoznaczne rozwiązanie. Układ ten nazywamy *układem równań normalnych*. Zwykle zapisujemy go w postaci macierzowej

$$\mathbf{G}\vec{a} = \vec{d},$$

gdzie

$$\begin{aligned} \mathbf{G} &= \begin{bmatrix} (v_1, v_1) & (v_2, v_1) & \dots & (v_n, v_1) \\ (v_1, v_2) & (v_2, v_2) & \dots & (v_n, v_2) \\ \vdots & \vdots & & \vdots \\ (v_1, v_n) & (v_2, v_n) & \dots & (v_n, v_n) \end{bmatrix}, \\ \vec{a} &= \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix}, \quad \vec{d} = \begin{bmatrix} (f, v_1) \\ (f, v_2) \\ \vdots \\ (f, v_n) \end{bmatrix}. \end{aligned}$$

Macierz \mathbf{G} nazywamy *macierzą Grama* układu wektorów v_1, v_2, \dots, v_n .

16.4 Numeryczne rozwiązanie zadania aproksymacji w przestrzeniach unitarnych

W poprzednim punkcie nic nie zakładaliśmy o bazie V — jej wybór był dowolny. W każdej bazie umiemy wyznaczyć

współczynniki wielomianu optymalnego poprzez rozwiązywanie układu równań normalnych (po uprzednim obliczeniu wszystkich elementów macierzy Grama).

W obliczeniach numerycznych nie każda baza jest jednakowo dobra. Problemem jest to, że układ równań normalnych może być bardzo źle uwarunkowany. Okazuje się, że dobrymi bazami (ze względu na koszt i jakość rozwiązania) są bazy, które tworzą układy ortonormalne.

Bazę ortonormalną umiemy już konstruować. Wystarczy взять dowolną bazę, a następnie zmodyfikowanym algorytmem Grama-Schmidta przeprowadzić jej ortonormalizację. Niech wynikiem będzie baza ortonormalna v_1, \dots, v_n . Wtedy z definicji układu ortonormalnego mamy

$$(v_i, v_j) = \begin{cases} 1 & \text{gdy } i = j, \\ 0 & \text{w p.p.,} \end{cases}$$

i macierz Grama jest macierzą identycznosciową, $\mathbf{G} = \mathbf{I}$. Układ równań normalnych przybiera postać

$$\begin{cases} \alpha_1 = (f, v_1), \\ \alpha_2 = (f, v_2), \\ \vdots \\ \alpha_n = (f, v_n), \end{cases}$$

w którym niewiadome α_i są już wyznaczone. Zatem

$$v^* = \sum_{i=1}^n (f, v_i) v_i.$$

Policzmy jeszcze błąd aproksymacji. Wiemy, że $e(f, V) = \sqrt{\|f\|^2 - \|v^*\|^2}$. Korzystając z własności iloczynu skalarnego mamy

$$\begin{aligned} \|v^*\|^2 &= \left(\sum_{i=1}^n (f, v_i) v_i, \sum_{j=1}^n (f, v_j) v_j \right) \\ &= \sum_{i=1}^n (f, v_i) \left(v_i, \sum_{j=1}^n (f, v_j) v_j \right) \\ &= \sum_{i=1}^n (f, v_i) \sum_{j=1}^n (f, v_j) (v_i, v_j) \\ &= \sum_{i=1}^n (f, v_i)^2. \end{aligned}$$

Zatem

$$e(f, V) = \sqrt{\|f\|^2 - \sum_{i=1}^n (f, v_i)^2}.$$

Rozwiązywanie zadania kosztuje jedynie $n + 1$ obliczeń iloczynów skalarnych (plus ewentualny koszt ortonormalizacji bazy, wtedy całkowity koszt wynosi $\frac{(n+1)(n+2)}{2}$). Jakość numeryczną należy badać osobno dla każdej przestrzeni V .

16.5 Aproksymacja średniokwadratowa wielomianami — szczególny przypadek aproksymacji w przestrzeniach unitarnych

Zajmiemy się obecnie przypadkiem, gdy $F = L_\rho^2(a, b)$ (lub $F = l_\rho^2(X)$) oraz $V = \Pi_n$. Wtedy element optymalny dla f nazywamy *n-tym wielomianem optymalnym dla f w sensie aproksymacji średniokwadratowej w $L_\rho^2(a, b)$* (lub $l_\rho^2(X)$). Do obliczenia elementu optymalnego można stosować metodę opisaną w poprzednim punkcie, lecz zamiast ortonormalizacji zmodyfikowanym algorytmem Grama-Schmidta zdecydowanie lepiej jest wykorzystać regułę trójczłonową do wyznaczania i reprezentowania bazy wielomianów ortonormalnych.

16.6 Ciągi podprzestrzeni przestrzeni unitarnej

W wielu zadaniach interesuje nas nie tyle wyznaczenie elementu optymalnego względem pewnej podprzestrzeni, lecz znalezienie przybliżenia z błędem mniejszym od zadanego $\varepsilon > 0$. Przybliżając np. e^x nie zależy nam na znalezieniu optymalnego wielomianu siódmego stopnia, lecz na znalezieniu wielomianu możliwie niskiego stopnia, dla którego błąd aproksymacji nie przekracza np. 10^{-15} .

Podobne zadanie zostało sformułowane na wykładzie wprowadzającym pojęcie aproksymacji klasycznej. W przypadku przestrzeni unitarnych warto je postawić następująco. Mamy dany ciąg podprzestrzeni skończenie wymiarowych

$$V_1 \subsetneq V_2 \subsetneq V_3 \subsetneq V_4 \subsetneq \dots$$

takich, że $\dim V_n = n$ oraz element $f \in F$. Szukamy najmniejszego n spełniającego nierówność

$$e(f, V_n) \leq \varepsilon$$

oraz elementu optymalnego v_n^* dla f względem V_n .

Okazuje się, że jeśli $V_n = \Pi_{n-1}$, to zadanie ma zawsze rozwiązanie. W ogólnym przypadku rozwiązanie nie musi istnieć.

Podam jeden z algorytmów praktycznego obliczania rozwiązania, o ile ono istnieje. Niech w_1, w_2, \dots będzie ciągiem ortonormalnym takim, że

$$V_n = \text{span}\{w_1, w_2, \dots, w_n\}.$$

Taki ciąg zawsze istnieje, bo

$$\forall_{i=1,2,\dots} (V_i \subset V_{i+1} \wedge \dim V_i = i).$$

Udowodniliśmy w poprzednim punkcie, że

$$\begin{aligned} v_n^* &= \sum_{i=1}^n (f, w_i) w_i, \\ v_{n+1}^* &= \sum_{i=1}^{n+1} (f, w_i) w_i, \end{aligned}$$

stąd

$$v_{n+1}^* = v_n^* + (f, w_{n+1}) w_{n+1}.$$

Podobnie, ponieważ

$$\begin{aligned} e^2(f, V_n) &= \|f\|^2 - \sum_{i=1}^n (f, w_i)^2, \\ e^2(f, V_{n+1}) &= \|f\|^2 - \sum_{i=1}^{n+1} (f, w_i)^2, \end{aligned}$$

to

$$e^2(f, V_{n+1}) = e^2(f, V_n) - (f, w_{n+1})^2$$

Algorytm polega na wyznaczaniu z powyższych zależności rekurencyjnych elementów optymalnych v_n^* oraz błędów $e(f, V_n)$, dopóki błąd nie będzie mniejszy niż ε (dokładniej będziemy analizować kwadraty błędów). Schemat algorytmu można zapisać jak niżej, przy czym zakładamy, że dana jest funkcja realizująca iloczyn skalarny i zdefiniowane są operatory dodawania elementów z V oraz mnożenia ich przez stałą. Tablica w zawiera elementy w_i ciągu ortonormalnego.

```
eps_kw=eps*eps;
blad_kw=skal(f,f);
n=0;
vopt=0;
do{
    n++;
    pom=skal(f,w[n]);
    vopt=vopt+pom*w[n];
    blad_kw=blad_kw+pom*pom;
}while(blad_kw>eps_kw)
```

17 Aproksymacja jednostajna

17.1 Sformułowanie zadania

Obecnie przyjmiemy, że $F = C(B)$, gdzie B jest zbiorem zwarteim, czyli domkniętym i ograniczonym. Mówimy wtedy o *zadaniu aproksymacji jednostajnej na B* . W przestrzeni $C(B)$ mamy zdefiniowaną normę supremum

$$\|f\| = \sup_{x \in B} |f(x)|.$$

W dalszym ciągu będziemy się głównie zajmowali przestrzeniami $C(B)$, gdzie B jest przedziałem domkniętym lub zbiorem skończonym.

Łatwo wykazać, że $C(B)$ nie jest silnie wypukła. Zatem nie dla każdej skończonej wymiarowej podprzestrzeni V mamy zagwarantowaną jednoznaczność rozwiązania zadania aproksymacji. Niemniej, jeśli V spełnia warunek, o którym mowa poniżej, to dla dowolnej funkcji $f \in C(B)$ istnieje dokładnie jeden element optymalny z V .

Definicja 9. Mówimy, że przestrzeń $V \subset C(B)$ wymiaru n spełnia warunek Haara¹⁴, jeżeli każda funkcja $v \in V \setminus \{0\}$ ma co najwyżej $n-1$ miejsc zerowych w B .

Przykład Przestrzeń Π_n spełnia warunek Haara, bo każdy wielomian $w \in \Pi_n \setminus \{0\}$ ma co najwyżej n różnych pierwiastków, a wymiarem Π_n jest $n+1$.

Twierdzenie 5. *Zadanie aproksymacji jednostajnej dla dowolnego elementu $f \in F$ względem V ma jednoznaczne rozwiązanie wtedy i tylko wtedy, gdy V spełnia warunek Haara.*

Zwróćmy uwagę, że jeśli V nie spełnia warunku Haara, to mimo to mogą (choć nie muszą) istnieć funkcje f , dla których elementy optymalne są wyznaczone jednoznacznie.

Podobnie jak w przypadku aproksymacji średniokwadratowej, wyróżniamy przypadek, gdy $V = \Pi_n$. Wtedy element optymalny dla $f \in C(B)$ względem Π_n nazywamy *n-tym wielomianem optymalnym dla f w sensie aproksymacji jednostajnej na B* .

17.2 Twierdzenie Czebyszewa o alternansie

Aproksymacja jednostajna jest zadaniem trudniejszym do rozwiązania od aproksymacji w przestrzeniach unitarnych. Dzieje się tak, gdyż nie istnieje w ogólnym przypadku metoda, która by po skończonej liczbie kroków dawała zawsze element optymalny. Znamy jedynie algorytmy, które konstruują ciąg elementów zbieżny do optymalnego. Podstawowym twierdzeniem charakteryzującym elementy optymalne oraz będącym podstawą algorytmów jest następujące.

Twierdzenie 6. *(Czebyszewa o alternansie) Rozważmy zadanie aproksymacji jednostajnej elementu $f \in C(B)$ względem podprzestrzeni $V \subset C(B)$ spełniającej warunek Haara, dla której $\dim V = n$. Niech B zawiera co najmniej $n+1$ punktów.*

Element $v^ \in V$ jest elementem optymalnym dla f wtedy i tylko wtedy, gdy istnieje $\sigma = 1$ lub $\sigma = -1$ oraz istnieje $n+1$ punktów $x_0 < x_1 < \dots < x_n$ w zbiorze B takich, że zachodzą równości*

$$f(x_i) - v^*(x_i) = (-1)^i \sigma \|f - v^*\|, \quad i = 0, 1, \dots, n.$$

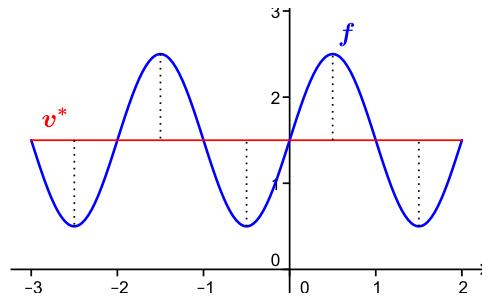
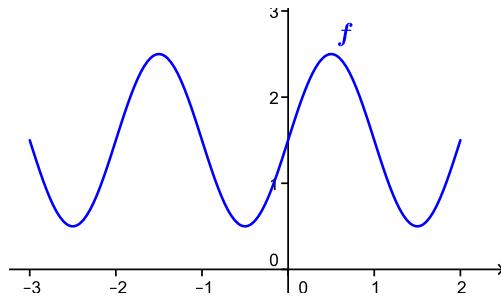
Zbiór punktów x_0, \dots, x_n nazywamy *alternansem*, a równania występujące w twierdzeniu *równaniami alternansu*.

Ponieważ $|(-1)^i \sigma| = 1$, to wartości bezwzględne prawych stron równań alternansu są takie same i wynoszą $\|f - v^*\|$. Zatem w punktach alternansu wykresy f i v^* muszą być najbardziej oddalone od siebie (te odległości są równe). Czynnik $(-1)^i$ powoduje, że jeśli w pewnym punkcie alternansu wykres f znajduje się nad wykresem v^* , to w kolejnym punkcie ma być na odwrót, a potem znów następuje zmiana i tak dalej. Liczba σ informuje, że nie jest istotne, czy w punkcie x_0 wykres f znajduje się nad wykresem v^* , czy też jest odwrotnie.

Ponieważ twierdzenie charakteryzuje element optymalny v^* , to z jego definicji wiemy, że $e(f, V) = \|f - v^*\|$. W punktach alternansu jest zatem „wybijany” błąd aproksymacji.

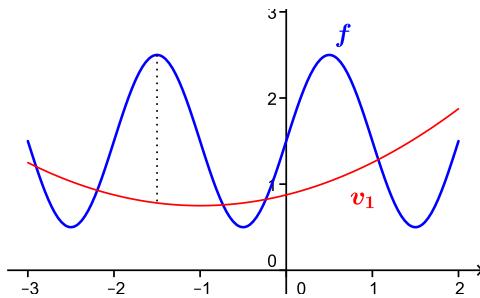
Przykład Rozpatrzmy przestrzeń $C[-3, 2]$ oraz funkcję $f(x) = \sin(\pi x) + \frac{3}{2}$.

¹⁴Takie podprzestrzenie nazywa się też *podprzestrzeniami Czebyszewa*.



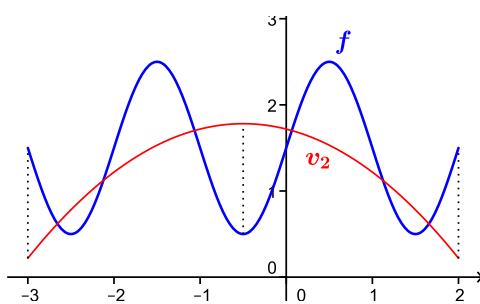
Szukamy drugiego wielomianu optymalnego, stąd $V = \Pi_2$ oraz $n = 3$. Skorzystamy z twierdzenia Czebyszewa o alternansie. Zauważmy, że możemy to zrobić, bo zbiór $B = [-3, 2]$ zawiera więcej niż $n+1 = 4$ punkty. Spróbujmy odgadnąć rozwiązanie na podstawie wykresu funkcji f , czyli znaleźć takie $v^* \in \Pi_2$, dla którego istnieją 4 punkty alternansu.

Wykresami funkcji należących do Π_2 są parabole lub linie proste. Zobaczmy, czy funkcja, której wykres jest parabolą może być rozwiązaniem, zacznijmy od v_1 z poniższego rysunku.



Norma $\|f - v_1\|$, czyli maksymalna odległość między wykresami f i v_1 , to długość odcinka zaznaczonego linią przerwaną. Jest ona osiągana tylko w jednym punkcie. Zatem mogliby być tylko jeden punkt alternansu $-\frac{3}{2}$, a potrzeba czterech.

Spróbujmy przesuwać parabolę, „wyginać” jej ramiona lub odbijać symetrycznie względem osi OX. Okazuje się, że w ten sposób można uzyskać maksymalnie 3 punkty spełniające równania alternansu, np. tak jak na rysunku poniżej.



Punktami alternansu mogłyby być $x_0 = -3$, $x_1 = -\frac{1}{2}$, $x_2 = 2$. Rzeczywiście, w tych punktach „wybijana” jest norma $\|f - v_2\|$ oraz wyrażenia $f(x_0) - v_2(x_0)$, $f(x_1) - v_2(x_1)$, $f(x_2) - v_2(x_2)$ mają naprzemienne znaki. Ale tych punktów jest za mało.

Poszukajmy obecnie rozwiązań wśród wielomianów, których wykresem jest linia prosta. Rozpatrzmy wielomian $v^*(x) = \frac{3}{2}$.

Dla niego można znaleźć aż 5 punktów alternansu: $x_0 = -\frac{5}{2}$, $x_1 = -\frac{3}{2}$, $x_2 = -\frac{1}{2}$, $x_3 = \frac{1}{2}$, $x_4 = \frac{3}{2}$. Ponieważ wystarczą cztery, to jeden można odrzucić (x_0 albo x_4). W myśl twierdzenia Czebyszewa $v^*(x) = \frac{3}{2}$ jest szukanym wielomianem optymalnym dla f względem Π_3 .

Zauważmy, że równocześnie v^* jest zerowym, pierwszym, drugim i trzecim wielomianem optymalnym dla f . Nie jest natomiast czwartym wielomianem optymalnym (w tym przypadku potrzeba sześciu punktów alternansu).

Przykład Znajdźmy pierwszy wielomian optymalny dla funkcji $f(x) = e^x$ w przestrzeni $C[-1, 1]$. Skorzystamy z twierdzenia Czebyszewa o alternansie. Szukamy takiego wielomianu $v^* \in \Pi_1$, by istniały dla niego trzy punkty alternansu w przedziale $[-1, 1]$. Ponieważ $v^* \in \Pi_1$, to $v^*(x) = ax + b$ dla pewnych $a, b \in \mathbb{R}$. Nieznane punkty alternansu oznaczmy przez x_0, x_1 i x_2 .

Z równań alternansu

$$f(x_i) - v^*(x_i) = (-1)^i \sigma \|f - v^*\|, \quad i = 0, 1, 2,$$

otrzymujemy

$$|f(x_i) - v^*(x_i)| = \|f - v^*\|, \quad i = 0, 1, 2.$$

Ponieważ

$$\|f - v^*\| = \sup_{x \in [-1, 1]} |f(x) - v^*(x)|,$$

to x_0, x_1, x_2 muszą być ekstremami globalnymi na przedziale $[-1, 1]$ funkcji $|g(x)|$, gdzie

$$g(x) = f(x) - v^*(x) = e^x - ax - b.$$

„Kandydatami” na ekstrema globalne są ekstrema lokalne funkcji $g(x)$ oraz krańce przedziału $[-1, 1]$. Warunkiem koniecznym istnienia ekstremum lokalnego jest zerowanie się pierwsiowej pochodnej

$$g'(x) = e^x - a,$$

czyli szukamy takich x , że

$$e^x = a.$$

Powyższa równość jest możliwa jedynie, gdy $a > 0$ i wtedy $x = \ln a$. Uzyskaliśmy zatem co najwyżej jedno ekstremum lokalne. Potrzebujemy trzech ekstremów globalnych (bo trzy punkty alternansu), zatem krańce przedziału (punkty -1 i 1) oraz $\ln a$ muszą nimi być. Stąd punktami alternansu są

$$x_0 = -1, \quad x_1 = \ln a, \quad x_2 = 1.$$

Wróćmy do równań alternansu

$$f(x_i) - v^*(x_i) = (-1)^i \sigma \|f - v^*\|, \quad i = 0, 1, 2.$$

Po podstawieniu $f(x) = e^x$, $v^*(x) = ax + b$, oraz punktów alternansu mamy

$$\begin{cases} e^{-1} + a - b = \sigma \|f - v^*\|, \\ e^{\ln a} - a \ln a - b = -\sigma \|f - v^*\|, \\ e^1 - a - b = \sigma \|f - v^*\|. \end{cases}$$

Wprowadźmy oznaczenie $E = \sigma \|f - v^*\|$. Wtedy

$$\begin{cases} e^{-1} + a - b = E, \\ a - a \ln a - b = -E, \\ e - a - b = E. \end{cases}$$

Otrzymaliśmy układ trzech równań z trzema niewiadomymi, niestety nie jest on liniowy, zatem nie istnieje ogólna metoda jego rozwiązywania. W tym przypadku jeśli od trzeciego równania odejmiemy pierwsze, to otrzymamy

$$a = \frac{1}{2}(e - e^{-1}) = \frac{e^2 - 1}{2e},$$

a po dodaniu mamy

$$E = \frac{1}{2}(e + e^{-1}) - b = \frac{e^2 + 1}{2e} - b.$$

Podstawmy a i E do drugiego równania

$$\frac{e^2 - 1}{2e} - \frac{e^2 - 1}{2e} \ln \frac{e^2 - 1}{2e} - b = -\frac{e^2 + 1}{2e} + b,$$

stąd

$$b = \frac{e}{2} - \frac{e^2 - 1}{4e} \ln \frac{e^2 - 1}{2e}.$$

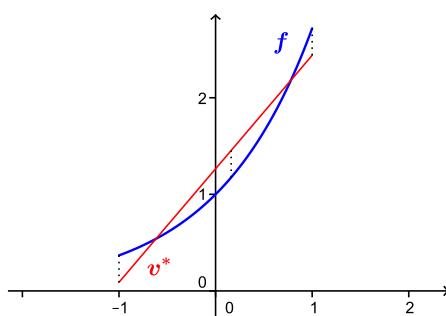
Zatem

$$v^*(x) = \frac{e^2 - 1}{2e} \left[x - \ln \sqrt{\frac{e^2 - 1}{2e}} \right] + \frac{e}{2},$$

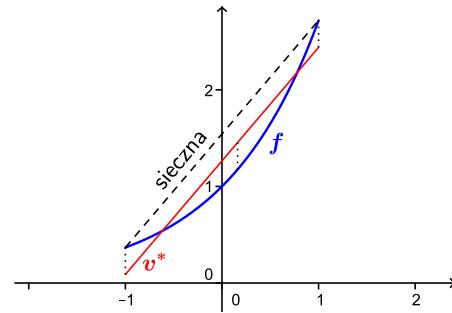
błąd aproksymacji wynosi

$$|E| = \left| \frac{1}{2e} + \frac{e^2 - 1}{4e} \ln \frac{e^2 - 1}{2e} \right|.$$

Pewne informacje, które zostały uzyskane analitycznie można odczytać z wykresu. Wśród wszystkich prostych tylko ta z rysunku może dać trzy punkty alternansu.



Stąd wynika, że punktami alternansu muszą być -1 i 1 . Dodatkowo prosta ta jest równoległa do siecznej funkcji f przechodzącej przez punkty $(-1, e^{-1})$, $(1, e)$.



Zatem sieczna i v^* mają ten sam współczynnik kierunkowy

$$\begin{aligned} a &= \frac{f(1) - f(-1)}{1 - (-1)} \\ &= \frac{e - e^{-1}}{2} \\ &= \frac{e^2 - 1}{2e}. \end{aligned}$$

Przykład powyższy ilustruje problemy, które pojawiają się przy próbie wyprowadzenia algorytmu znajdującego element optymalny na podstawie twierdzenia Czebyszewa. Mianowicie oprócz równań alternansu potrzebna rozwiązać zadanie znalezienia ekstremum globalnego funkcji, której parametrami są szukane współczynniki elementu optymalnego.

17.3 Aproksymacja w przestrzeniach funkcji określonych na skończonej liczbie punktów

W tym punkcie zajmiemy się szczególnym przypadkiem zbioru B tzn. takim, który ma skończoną liczbę elementów

$$B = \{b_0, b_1, \dots, b_k\}.$$

Ustalmy funkcję $f \in C(B)$ oraz podprzestrzeń spełniającą warunek Haara $V \subset C(B)$, gdzie $\dim V = n$. Szukamy elementu optymalnego v^* dla f względem V .

Będzie nas interesował jedynie przypadek, gdy $k = n$, bo wówczas będzie on wykorzystany przy sformułowaniu algorytmu w następnym punkcie.

Dane: $B = \{b_0, b_1, \dots, b_n\}$,
 $V \subset C(B)$ spełniająca warunek Haara,
 $n = \dim V$,
 $f \in C(B)$

Wyniki: $v^* \in V$ taki, że $\|f - v^*\| = \inf_{v \in V} \|f - v\|$
oraz $e(f, V) = \|f - v^*\|$

Z twierdzenia Czebyszewa wynika, że jeśli v^* jest elementem optymalnym, to istnieje $n+1$ punktów alternansu, więc muszą być nimi wszystkie punkty zbioru B . Założymy dla wygody, że $b_0 < b_1 < \dots < b_n$, zatem v^* spełnia równania

$$f(b_i) - v^*(b_i) = \sigma(-1)^i \|f - v^*\|, \quad i = 0, 1, \dots, n.$$

Wprowadźmy oznaczenie

$$E = \sigma \|f - v^*\|,$$

wtedy równania alternansu można zapisać w postaci

$$v^*(b_i) + (-1)^i E = f(b_i), \quad i = 0, 1, \dots, n.$$

Jeśli ustalimy bazę v_1, v_2, \dots, v_n przestrzeni V to aby wyznaczyć v^* wystarczy znaleźć jego współczynniki $\alpha_1, \alpha_2, \dots, \alpha_n$ w tej bazie

$$v^*(x) = \alpha_1 v_1(x) + \alpha_2 v_2(x) + \dots + \alpha_n v_n(x).$$

Podstawmy powyższą postać do równań alternansu. Otrzymujemy układ

$$\begin{cases} \alpha_1 v_1(b_0) + \alpha_2 v_2(b_0) + \dots + \alpha_n v_n(b_0) - (-1)^0 E = f(b_0), \\ \alpha_1 v_1(b_1) + \alpha_2 v_2(b_1) + \dots + \alpha_n v_n(b_1) - (-1)^1 E = f(b_1), \\ \vdots \\ \alpha_1 v_1(b_n) + \alpha_2 v_2(b_n) + \dots + \alpha_n v_n(b_n) - (-1)^n E = f(b_n), \end{cases}$$

z niewiadomymi $\alpha_1, \alpha_2, \dots, \alpha_n$ oraz E . Jest to układ równań liniowych, który posiada jednoznaczne rozwiązanie, bo postawione zadanie aproksymacji na dokładnie jedno rozwiązanie. W zależności od bazy należy dobrać odpowiednią metodę jego rozwiązania.¹⁵

Zauważmy, że rozwiązujeając układ obliczamy oprócz v^* również błąd aproksymacji, bo $e(f, V) = |E|$.

17.3.1 Przypadek $V = \Pi_{n-1}$

Szczególnie łatwo można rozwiązać powyższy układ, gdy $V = \Pi_{n-1}$. Udowodniono bowiem, że

$$E = \frac{f_{b_0, b_1, \dots, b_n}}{g_{b_0, b_1, \dots, b_n}}.$$

gdzie funkcja $g \in C(B)$ jest taka, że

$$g(b_i) = (-1)^i.$$

We wzorze występują różnice dzielone dla funkcji f i g , do ich wyliczenia wystarczy jedynie znajomość wartości funkcji w punktach ze zbioru B . Niewiadomą E da się więc wyznaczyć jako pierwszą.

Po obliczeniu E możemy równania

$$v^*(b_i) = f(b_i) - (-1)^i E, \quad i = 0, 1, \dots, n.$$

potraktować jako równania interpolacyjne Lagrange'a w węzłach b_0, b_1, \dots, b_n , wielomianem interpolacyjnym jest v^* . Równań jest o jedno za dużo, bo v^* należy do Π_{n-1} , ale są niesprzeczne. Wykreślamy więc ostatnie (można dowolne) i rozwiązujemy zadanie interpolacji Lagrange'a dla tabeli

b_0	b_1	\dots	b_{n-1}
$f(b_0) - E$	$f(b_1) + E$	\dots	$f(b_{n-1}) - (-1)^{n-1} E$

¹⁵ Metody rozwiązywania układów równań nie zostały omówione na tym wykładzie. Zwykle stosowany algorytm Gaussa ma koszt $\Theta(n^3)$, gdzie n jest liczbą niewiadomych.

Jeśli zadanie rozwiązujemy w bazie Newtona, to można postąpić następująco.

1. Przeprowadź algorytm różnic dzielonych dla f .
2. Przeprowadź algorytm różnic dzielonych dla g .
3. Oblicz $E = \frac{f_{b_0, b_1, \dots, b_n}}{g_{b_0, b_1, \dots, b_n}}$.
4. Dla $i = 0, 1, \dots, n-1$ oblicz

$$\alpha_i = f_{b_0, b_1, \dots, b_i} - E g_{b_0, b_1, \dots, b_i}.$$

Zauważmy, że w punkcie czwartym nie trzeba trzeci raz wykonywać algorytmu różnic dzielonych, lecz wystarczy posłużyć się wynikami z pierwszego i drugiego punktu.

17.4 Algorytm Remeza

Algorytm ten został opracowany w roku 1931. Ma on zastosowanie w aproksymacji w przestrzeni $C[a, b]$ względem podprzestrzeni $V \subset C[a, b]$ spełniającej warunek Haara takiej, że $\dim V = n$.

Dane:	$[a, b]$, $V \subset C[a, b]$ spełniająca warunek Haara, $n = \dim V$, $f \in C[a, b]$
Wyniki:	$v^* \in V$ taki, że $\ f - v^*\ = \inf_{v \in V} \ f - v\ $ oraz $e(f, V) = \ f - v^*\ $

Algorytm konstruuje ciąg układów punktów

$$x_0^j < x_1^j < \dots < x_n^j$$

dla $j = 0, 1, 2, \dots$ zbieżnych do alternansu i ciąg elementów $v_j \in V$ zbieżnych do elementu optymalnego. Układ $x_0^j, x_1^j, \dots, x_n^j$ jest zatem kolejnym przybliżeniem alternansu, element v_j przybliżeniem elementu optymalnego.

W dalszym ciągu tego punktu $\|\cdot\|_B$ oznacza zawsze normę supremum na zbiorze B . Algorytm można zapisać następująco.

Algorytm

1. wybierz dowolny początkowy układ punktów

$$a \leq x_0^0 < x_1^0 < \dots < x_n^0 \leq b$$

2. dla $j = 0, 1, 2, \dots$ rób

- (a) skonstruuj element $v_j \in V$ optymalny dla f w przestrzeni $C(B_j)$, gdzie $B_j = \{x_0^j, x_1^j, \dots, x_n^j\}$
- (b) oblicz błąd aproksymacji w $C(B_j)$, czyli wielkość

$$e_j = \|f - v_j\|_{B_j}$$

- (c) oblicz $E_j = \|f - v_j\|_{[a,b]}$
- (d) jeśli $E_j = e_j$ to zakończ, bo v_j jest wielomianem optymalnym dla f w przestrzeni $C[a,b]$
- (e) skonstruuj nowe przybliżenie alternansu, w tym celu wymień przynajmniej jeden punkt z układu

$$a \leq x_0^j < x_1^j < \dots < x_n^j \leq b$$

tak, by nowy układ

$$a \leq x_0^{j+1} < x_1^{j+1} < \dots < x_n^{j+1} \leq b$$

spełniał

- i. dla $i = 0, \dots, n$

$$|f(x_i^{j+1}) - v_j(x_i^{j+1})| \geq e_j$$

- ii. istnieje co najmniej jeden punkt x_k taki, że

$$|f(x_k^{j+1}) - v_j(x_k^{j+1})| = E_j$$

- iii. dla wszystkich $i = 0, 1, \dots, n-1$

$$\text{znak wyrażenia } f(x_i^{j+1}) - v_j(x_i^{j+1})$$

jest przeciwny do

$$\text{znaku wyrażenia } f(x_{i+1}^{j+1}) - v_j(x_{i+1}^{j+1})$$

Uwagi do algorytmu Remeza

1. Algorytm działa przy dowolnym wyborze początkowego układu punktów

$$a \leq x_0^0 < x_1^0 < \dots < x_n^0 \leq b,$$

ale im bliżej są one alternansu, tym mniej kroków musi wykonać algorytm (a każdy krok jest bardzo kosztowny). Jeśli nie dysponujemy żadną wiedzą o alternansie, to można wziąć węzły Czebyszewa.

2. (a) Konstrukcja elementu v_j została opisana w poprzednim punkcie. Srowadza się ona do rozwiązywania układu równań liniowych.
- (b) Podczas rozwiązywania tego układu od razu jest obliczany błąd e_j .
- (c) Błąd E_j jest zwykle trudny do obliczenia. Często jest on zastępowany prostym obliczeniowo przybliżeniem: badamy funkcję $f - v_j$ we wszystkich punktach gęstej siatki z $[a, b]$ i wybieramy jej największą co do modułu wartość. Algorytm może dać wtedy wynik nieznacznie różniący się od elementu optymalnego (dokładność zależy od gęstości siatki).
- (d) Warunek $E_j = e_j$ w praktyce zastępowany jest przez $E_j < e_j + \varepsilon$, dla pewnego małego $\varepsilon > 0$, bo zarówno e_j jak i E_j są przy obliczeniach numerycznych obarczone błędami. Problem dotyczy zwłaszcza E_j gdy jest zastępowane przybliżeniem opisanym w 2c.

- (e) Wymiana tylko jednego punktu jest wprawdzie wystarczająca dla zbieżności, ale powoduje konieczność wykonywania większej liczby kroków. Najlepiej wymienić punkty tak, by jak najwięcej spełniało

$$|f(x_i^{j+1}) - v_j(x_i^{j+1})| \approx E_j.$$

W praktyce stosowana jest metoda polegająca na wyszukiwaniu w sąsiedztwie punktu x_i^j takiego punktu x_i^{j+1} , że wielkość

$$|f(x_i^{j+1}) - v_j(x_i^{j+1})|$$

jest możliwie duża przy zachowaniu jednakowych znaków wyrażeń

$$f(x_i^{j+1}) - v_j(x_i^{j+1}) \text{ i } f(x_i^j) - v_j(x_i^j)$$

Można nowych punktów szukać podczas wykonywania punktu 2c.

Ciąg v_j jest zbieżny do elementu optymalnego dla f w $C[a, b]$ o ile obliczenia są wykonywane dokładnie (błędy zaokrąglone powodują, że wynik uzyskujemy tylko z pewną dokładnością). Zbieżność może być jednak dowolnie wolna.

Algorytm ten jest niezwykle kosztowny. W każdym kroku trzeba rozwiązać układ równań, a ponadto obliczyć wartość E_j . Zastosowanie tego algorytmu jest ograniczone.

17.5 Wyznaczanie elementów prawie optymalnych

Ze względu na koszt algorytmu Remeza często zamiast szukać n -tego wielomianu optymalnego dla funkcji $f \in C[a, b]$ zadowalamy się znalezieniem „dobrze” przybliżającego wielomianu, czyli takiego $w \in \Pi_n$, że

$$\|f - w\| \leq K_n e(f, \Pi_n),$$

dla pewnego wolno rosnącego ciągu K_n . Takie wielomiany nazywamy *prawie optymalnymi*. Stosowane są różne metody wyznaczania wielomianów prawie optymalnych, omówię trzy.

Dane: $[a, b]$,
 $f \in C[a, b]$

Wyniki: $w \in \Pi_n$ taki, że $\|f - w\| \leq K_n e(f, \Pi_n)$, gdzie
 K_n jest wolno rosnącym ciągiem

Metoda 1 W przypadku $[a, b] = [-1, 1]$ funkcję f przybliżamy wielomianem $w_1 \in \Pi_n$, który jest n -tym wielomianem optymalnym w sensie aproksymacji średniokwadratowej w przestrzeni $L_\rho[-1, 1]$ z wagą $\rho(x) = \frac{1}{\sqrt{1-x^2}}$.

Gdy $[a, b]$ jest dowolny, to należy dokonać zamiany zmiennej, czyli aproksymować funkcję

$$g(x) = f\left(\frac{b-a}{2}x + \frac{a+b}{2}\right)$$

w przestrzeni $L_\rho[-1, 1]$. Znaleziony wielomian w optymalny dla g w sensie aproksymacji średniokwadratowej pozwala wyznaczyć szukany wielomian w_1 przybliżający f

$$w_1(x) = w\left(\frac{x - \frac{a+b}{2}}{\frac{b-a}{2}}\right).$$

Metoda 2 Funkcję f przybliżamy wielomianem $w_2 \in \Pi_n$, który interpoluje f w jednokrotnych węzłach Czebyszewa.

Metoda 3 Funkcję f przybliżamy wielomianem $w_3 \in \Pi_n$ interpolującym f w jednokrotnych węzłach, które są ekstremami globalnymi na $[a, b]$ wielomianu

$$\tilde{T}_{n-1} = T_{n-1} \left(\frac{x - \frac{a+b}{2}}{\frac{b-a}{2}} \right).$$

Można udowodnić twierdzenie.

Twierdzenie 7. Dla dowolnej funkcji $f \in C[a, b]$

$$\begin{aligned}\|f - w_1\| &\leq A_n e(f, \Pi_n), \\ \|f - w_2\| &\leq B_n e(f, \Pi_n), \\ \|f - w_3\| &\leq C_n e(f, \Pi_n),\end{aligned}$$

gdzie

$$\begin{aligned}A_n &= \frac{4}{\pi^2} \ln(n) + O(1), \\ B_n &= \frac{2}{\pi} \ln(n) + O(1)\end{aligned}$$

oraz

$$C_n = B_{n-1}$$

dla n nieparzystych oraz

$$\begin{aligned}C_n &\leq B_n, \\ C_n &\geq B_{n-2} - \frac{1}{n} \tan\left(\frac{\pi}{4n}\right)\end{aligned}$$

dla n parzystych. Ponadto istnieją niezerowe funkcje f , dla których

$$\begin{aligned}\|f - w_1\| &= A_n e(f, \Pi_n), \\ \|f - w_2\| &= B_n e(f, \Pi_n), \\ \|f - w_3\| &= C_n e(f, \Pi_n).\end{aligned}$$

Wielkości A_n, B_n, C_n można obliczać numerycznie. W tabeli poniżej zamieszczam ich przybliżenia dla wybranych n .¹⁶

n	1	10	20	50	100	500	1000	5000
A_n	2.44	3.22	3.49	3.86	4.14	4.79	5.07	5.72
B_n	2.41	3.49	3.90	4.47	4.90	5.92	6.36	7.38
C_n	2.00	3.43	3.87	4.45	4.89	5.92	6.36	7.38

Metoda pierwsza jest nieco lepsza od pozostałych (zwłaszcza dla dużych n), ale jest też istotnie droższa.

17.6 Zwiększenie stopni wielomianu aproksymującego

Można udowodnić, że $\lim_{n \rightarrow \infty} e(f, \Pi_n) = 0$. Zbieżność ta może być jednak bardzo wolna, jeśli założymy jedynie ciągłość f . Nie można zwykle podać ile wynosi n jeśli chcemy, aby $e(f, \Pi_n) \leq \varepsilon$, a o to właśnie chodzi.

¹⁶Tabela pochodzi z książki Marka Kowalskiego „Approximation Theory”.

Jeśli założymy większą regularność funkcji możemy otrzymać pewne oszacowania. Na przykład, jeśli

$$\sup_{x \in [a, b]} |f^{(n+1)}(x)| \leq M,$$

to

$$e(f, \Pi_n) \leq \frac{2M}{(n+1)!} \left(\frac{b-a}{4} \right)^{n+1},$$

bo błąd aproksymacji jest nie większy niż błąd interpolacji w węzłach Czebyszewa. Jeśli zatem żądamy, by funkcja f z przestrzeni $C[-2, 2]$, dla której $M = 5$ była przybliżona z dokładnością 10^{-10} , to wystarczy spełnić warunek

$$\frac{10}{(n+1)!} \leq 10^{-10},$$

stąd

$$(n+1)! \geq 10^{11}.$$

Minimalne takie n wynosi 15. Użycie wielomianu piętnastego stopnia gwarantuje żądaną dokładność, nawet jeśli zamiast optymalnego znajdziemy wielomian prawie optymalny jedną z przedstawionych metod.

Kwadratury

Dorota Dąbrowska, UKSW

2017/18

Spis treści

18 Numeryczne obliczanie całki oznaczonej — kwadratury	90
18.1 Ogólna postać kwadratur	90
18.2 Ciągi kwadratur	91
18.3 Rząd i reszta kwadratury	91
19 Kwadratury interpolacyjne	91
19.1 Definicja	91
19.2 Przykłady kwadratur interpolacyjnych	92
19.3 Reszta kwadratur interpolacyjnych	93
19.4 Rząd kwadratur interpolacyjnych	94
19.5 Kwadratury Gaussa	94
20 Kwadratury złożone (powtarzalne)	96
20.1 Zasada budowy	96
20.2 Przykłady kwadratur złożonych	97
20.3 Złożone kwadratury Newtona–Cotesa	97
20.4 Kwadratury złożone Gaussa	98

18 Numeryczne obliczanie całki oznaczonej — kwadratury

18.1 Ogólna postać kwadratur

W tym dziale zajmiemy się zadaniem przybliżonego obliczania całki oznaczonej

$$I(f) = \int_a^b f(x) dx.$$

Będziemy zakładać, że f jest całkowalna, czyli $I(f)$ istnieje oraz, że $a < b$. Dopuszczamy również nieskończone granice całkowania ($a = -\infty$ lub $b = \infty$). Czasami rozważać będziemy zadanie ogólniejsze, tj. całkowanie z funkcją wagową

$$I(f) = \int_a^b \rho(x)f(x) dx,$$

gdzie ρ jest funkcją ciągłą na $[a, b]$ taką, że $\forall_{x \in [a, b]} \rho(x) \geq 0$ i $\int_a^b \rho(x) dx > 0$. Waga może być dana z góry, albo mając konkretną funkcję f może okazać się wygodne „wyodrębnienie” z niej wagi i zastosowanie metody specyficznej dla całek z wagą.

Przypomnijmy, że operacja całkowania jest liniowa ze względu na funkcje, tzn. dla dowolnych funkcji f i g , dla których znane są całki $I(f)$ i $I(g)$ oraz dowolnych $\alpha, \beta \in \mathbb{R}$ zachodzi

$$I(\alpha f + \beta g) = \alpha I(f) + \beta I(g).$$

Obliczanie całek poprzez wyznaczanie funkcji pierwotnej jest często zadaniem trudnym, a czasami funkcji pierwotnej nie da się wyrazić przy pomocy funkcji elementarnych. Przybliżone obliczanie całek oznaczonych nie wymaga wyznaczenia funkcji pierwotnej, lecz jedynie znajomości wartości funkcji.

Dokładniej będziemy zajmować się metodami, które oprócz wartości funkcji w skończonej liczbie węzłów mogą wykorzystywać też wartości pochodnych w tych węzłach. Dany są (podobnie jak przy interpolacji Hermite'a)

1. węzły $x_0, x_1, \dots, x_m \in [a, b]$,
2. ich krotności $k_0, k_1, \dots, k_m \in \mathbb{N} \setminus \{0\}$,
3. wartości funkcji i pochodnych w tych węzłach

$$\begin{array}{llll} f(x_0), & f(x_1), & \dots & f(x_m), \\ f'(x_0), & f'(x_1), & \dots & f'(x_m), \\ \vdots & \vdots & & \vdots \\ f^{(k_0-1)}(x_0), & f^{(k_1-1)}(x_1), & \dots & f^{(k_m-1)}(x_m). \end{array}$$

Wszystkie rozważane metody będą polegały na sumowaniu powyższych wartości funkcji i pochodnych pomnożonych przez odpowiednio dobrane stałe. Jeśli więc przez $Q(f)$ oznaczymy przybliżoną wartość całki $I(f)$, to interesować nas będą tylko metody, dla których

$$Q(f) = \sum_{i=0}^m \sum_{j=0}^{k_i-1} A_{ij} f^{(j)}(x_i)$$

dla pewnych stałych $A_{ij} \in \mathbb{R}$. Metody różnią się wyborem węzłów, krotności i stałych.

Definicja 1. Funkcję Q przyporządkowującą każdemu f , dla którego istnieje $I(f)$ liczbę

$$Q(f) = \sum_{i=0}^m \sum_{j=0}^{k_i-1} A_{ij} f^{(j)}(x_i)$$

będącą przybliżeniem całki nazywamy *kwadraturą*. Liczby A_{ij} nazywamy *wagami kwadratury*.

Zauważmy, że podobnie jak operacja całkowania, Q jest funkcją liniową ze względu na funkcje, tzn. dla dowolnych f i g oraz $\alpha, \beta \in \mathbb{R}$

$$Q(\alpha f + \beta g) = \alpha Q(f) + \beta Q(g).$$

18.2 Ciągi kwadratur

Kwadratury konstruowane są tak, by w pewnej klasie funkcji można je stosować do dobrego przybliżania całek. Naturalne jest rozważanie nie tylko pojedynczych kwadratur, lecz ich ciągów Q_n takich, że im większe n tym lepsze przybliżenie daje Q_n . Wtedy mając dane $\varepsilon > 0$ oraz f można stawiać pytanie jaka jest najmniejsza liczba n taka, że

$$|I(f) - Q_n(f)| \leq \varepsilon.$$

Aby takie postępowanie miało sens ciąg $Q_n(f)$ musi być zbieżny do $I(f)$. Zajmiemy się dalej ciągami kwadratur, które są konstruowane dla węzłów jednokrotnych, przy czym n -ta kwadratura oparta jest na $n+1$ węzłach czyli

$$Q_n(f) = \sum_{i=0}^n A_i^{(n)} f(x_i^{(n)}).$$

Poniższe twierdzenia opisują jakie własności powinny spełniać takie ciągi kwadratur, by zachodziła zbieżność dla funkcji ciągłych.

Twierdzenie 1. (Luzina) Rozpatrzmy ciąg kwadratur pośtaci $Q_n(f) = \sum_{i=0}^n A_i^{(n)} f(x_i^{(n)})$. Dla dowolnej $f \in C[a, b]$ ciąg $Q_n(f)$ jest zbieżny do $I(f) = \int_a^b \rho(x)f(x) dx$ wtedy i tylko wtedy, gdy spełnione są dwa warunki

1. dla dowolnego wielomianu w ciągu $Q_n(w)$ jest zbieżny do $I(w)$,
2. istnieje stała $K > 0$ taka, że dla dowolnego n

$$\sum_{i=0}^n |A_i^{(n)}| < K.$$

Dowody zbieżności ciągów kwadratur (dla funkcji ciągłych) można zatem sprowadzić do badania, czy są one zbieżne dla wielomianów, oraz czy sumy modułów wag kwadratur są wspólnie ograniczone.

Drugie twierdzenie dotyczy kwadratur z wagami dodatnimi.

Twierdzenie 2. Niech $Q_n(f) = \sum_{i=0}^n A_i^{(n)} f(x_i^{(n)})$, gdzie wszystkie wagi $A_i^{(n)}$ są dodatnie. Dla dowolnej $f \in C[a, b]$ ciąg $Q_n(f)$ jest zbieżny do $I(f) = \int_a^b \rho(x)f(x) dx$ wtedy i tylko wtedy, gdy dla dowolnego wielomianu w ciągu $Q_n(w)$ jest zbieżny do $I(w)$.

18.3 Rząd i reszta kwadratury

W obydwu powyższych twierdzeniach istotna okazuje się zbieżność kwadratur dla wielomianów. Między innymi to było powodem wprowadzenia pojęcia rzędu kwadratur, który charakteryzuje zachowanie się kwadratury dla wielomianów.

Definicja 2. Kwadratura Q jest *rzędu n* , jeśli

1. dla dowolnego $w \in \Pi_{n-1}$ zachodzi $Q(w) = I(w)$,
2. istnieje $w \in \Pi_n$ taki, że $Q(w) \neq I(w)$.

Rząd informuje, że wielomianów stopni mniejszych niż rząd mamy gwarancję, że kwadratura oblicza dokładną wartość całki. Im większy rząd kwadratury, tym lepiej.

Istotniejszymi miarami z praktycznego punktu widzenia są reszta i błąd kwadratury.

Definicja 3. Resztą kwadratury Q dla funkcji f nazywamy

$$R(f) = I(f) - Q(f).$$

Błędem kwadratury Q w klasie funkcji F jest

$$R_F = \sup_{f \in F} R(f).$$

Obydwie powyższe wielkości opisują, jak dobrze kwadratura przybliża całkę. Pierwsza określa jakość dla pojedynczej funkcji, druga najlepsze zachowanie błędu wśród wszystkich funkcji pewnej klasy.

19 Kwadratury interpolacyjne

19.1 Definicja

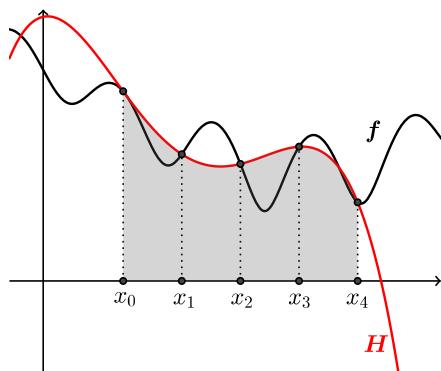
Rozważmy zadanie obliczania całki

$$I(f) = \int_a^b \rho(x)f(x) dx.$$

Będziemy obecnie konstruować kwadratury następująco. Dla zadanych węzłów $x_0, x_1, \dots, x_m \in [a, b]$, ich krotności $k_0, k_1, \dots, k_m \in \mathbb{N} \setminus \{0\}$, budujemy wielomian Hermite'a H

interpolującą funkcję f . Zamiast całkować f całkujemy H i w ten sposób przybliżamy $I(f)$, czyli

$$Q(f) = \int_a^b \rho(x)H(x) dx.$$



Należy w tym momencie udowodnić, że powyższy wzór rzeczywiście określa kwadraturę czyli, istnieją takie wagи A_{ij} , że

$$Q(f) = \sum_{i=0}^m \sum_{j=0}^{k_i-1} A_{ij} f^{(j)}(x_i).$$

W przypadku węzłów jednokrotnych uzasadnienie jest proste. Wielomian H jest wtedy postaci

$$H(x) = \sum_{i=0}^m f(x_i)l_i(x),$$

więc

$$\begin{aligned} Q(f) &= \int_a^b \rho(x) \sum_{i=0}^m f(x_i)l_i(x) dx \\ &= \sum_{i=0}^m f(x_i) \int_a^b \rho(x)l_i(x) dx \\ &= \sum_{i=0}^m f(x_i)A_{ij}, \end{aligned}$$

gdzie

$$A_{ij} = \int_a^b \rho(x)l_i(x) dx.$$

Jeśli węzły są wielokrotne, uzasadnienie może wyglądać tak. Niech $n = \sum_{i=0}^n k_i - 1$. Zapiszmy wielomian Hermite'a w bazie Newtona. Jego współczynnikami są odpowiednie różnice dzielone, czyli

$$H(x) = \sum_{l=0}^n f_{x_0 \dots x_l} N_l(x),$$

gdzie N_i jest i -tym wielomianem bazy Newtona o węzłach

$$\underbrace{x_0, \dots, x_0}_{k_0}, \underbrace{x_1, \dots, x_1}_{k_1}, \dots, \underbrace{x_{m-1}, \dots, x_{m-1}}_{k_{m-1}}, \underbrace{x_m, \dots, x_m}_{k_{m-1}}$$

W tabeli różnic dzielonych wykorzystujemy tylko wartości funkcji i pochodnych w węzłach. Każda różnica dzielona jest sumą tych wartości pomnożonych przez stałe, tj.

$$f_{x_0 \dots x_l} = \sum_{i=0}^m \sum_{j=0}^{k_i-1} \alpha_{ij}^{(l)} f^{(j)}(x_i),$$

$$\begin{aligned} Q(f) &= \int_a^b \rho(x) \sum_{l=0}^m \sum_{i=0}^m \sum_{j=0}^{k_i-1} \alpha_{ij}^{(l)} f^{(j)}(x_i) N_l(x) dx \\ &= \sum_{i=0}^m \sum_{j=0}^{k_i-1} \left(\sum_{l=0}^m \alpha_{ij}^{(l)} \int_a^b \rho(x) N_l(x) dx \right) f^{(j)}(x_i) \\ &= \sum_{i=0}^m \sum_{j=0}^{k_i-1} A_{ij} f^{(j)}(x_i), \end{aligned}$$

gdzie

$$A_{ij} = \sum_{l=0}^m \alpha_{ij}^{(l)} \int_a^b \rho(x) N_l(x) dx.$$

Możemy zatem napisać definicję.

Definicja 4. Dla zadania obliczania całki

$$I(f) = \int_a^b \rho(x)f(x) dx$$

kwadratura interpolacyjna nazywamy kwadraturę, która funkcji f przyporządkowuje całkę $I(H)$, gdzie H jest wielomianem Hermite'a interpolującym f w węzłach z $[a, b]$.

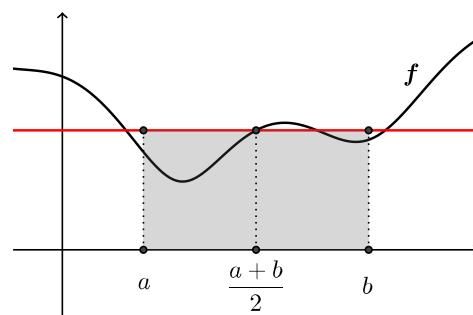
19.2 Przykłady kwadratur interpolacyjnych

Poniższe przykłady dotyczą zadania obliczania całki

$$I(f) = \int_a^b f(x) dx.$$

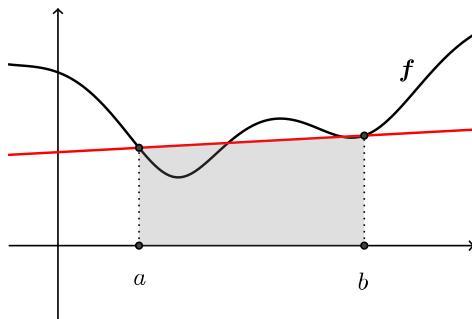
Wyprowadzenie podanych wzorów pozostawiamy jako ćwiczenie.

Kwadratura prostokątów Interpolujemy f w jednokrotnym węźle $x_0 = \frac{b-a}{2}$.



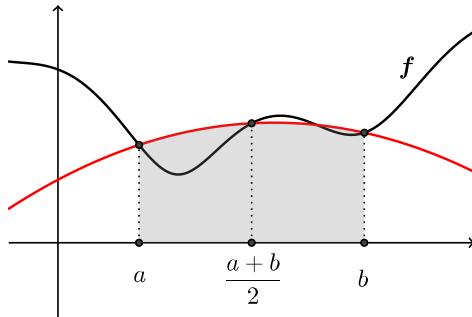
$$Q(f) = (b-a)f\left(\frac{a+b}{2}\right)$$

Kwadratura trapezów Interpolujemy f w dwóch jednokrotnych węzłach $x_0 = a$, $x_1 = b$.



$$Q(f) = (b - a) \frac{f(a) + f(b)}{2}$$

Kwadratura parabol (Simpsona) Interpolujemy f w trzech jednokrotnych węzłach $x_0 = a$, $x_1 = \frac{b-a}{2}$, $x_2 = b$.



$$Q(f) = (b - a) \frac{f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)}{6}$$

Ostatnie dwa przykłady kwadratur konstruowane są wg ogólnego schematu: dokonujemy interpolacji funkcji f w $n+1$ równoodległych pojedynczych węzłach z przedziału $[a, b]$, następnie całkujemy wielomian interpolacyjny. Tu n wynosiło kolejno 1, 2. Jeśli dopuścimy dowolne n , to takie kwadratury nazywamy *kwadraturami Newtona–Cotesa*.¹

Kwadratury te są stosunkowo dobrze zbadane. Okazuje się między innymi, że ich wagi nie zawsze są dodatnie. Dokładniej, dla każdego $n \geq 10$ istnieje waga ujemna. Poza tym jeśli rozpatrzymy maksymalny moduł wag przy ustalonym n , to okazuje się, że wraz ze wzrostem n dąży on do nieskończoności. Te cechy powodują ograniczenia zastosowań kwadratur Newtona–Cotesa gdy n jest duże.

Poniżej zamieszczam tabelę z wagami kwadratur Newtona–Cotesa dla $n = 1, 2, 3, 4, 5, 6$. Można udowodnić, że przy ustalonym n wagi i -ta i $(n-i)$ -ta są takie same, zatem zamieszczam tylko te dla $i \leq \lceil \frac{n}{2} \rceil$.

n	A_0	A_1	A_2	A_3
1	$\frac{1}{2}(b-a)$			
2	$\frac{1}{6}(b-a)$	$\frac{4}{6}(b-a)$		
3	$\frac{1}{8}(b-a)$	$\frac{3}{8}(b-a)$		
4	$\frac{7}{90}(b-a)$	$\frac{32}{90}(b-a)$	$\frac{12}{90}(b-a)$	
5	$\frac{19}{288}(b-a)$	$\frac{75}{288}(b-a)$	$\frac{50}{288}(b-a)$	
6	$\frac{41}{840}(b-a)$	$\frac{216}{840}(b-a)$	$\frac{27}{840}(b-a)$	$\frac{272}{840}(b-a)$

19.3 Reszta kwadratur interpolacyjnych

W tym punkcie zakładamy, że całkę

$$I(f) = \int_a^b \rho(x)f(x) dx$$

przybliżamy kwadraturą interpolacyjną opartą na węzłach

$$x_0, x_1, \dots, x_m \in [a, b]$$

o krotnościach

$$k_0, k_1, \dots, k_m.$$

Przyjmujemy, że $n = \sum_{i=0}^m k_i - 1$,

$$\tilde{N}_{n+1}(x) = \prod_{i=0}^m (x - x_i)^{k_i}.$$

oraz $\|\cdot\|$ oznacza normę supremum na $[a, b]$.

Korzystając z twierdzeń, które opisują zachowanie reszty dla interpolacji Hermite'a można udowodnić następujące twierdzenia.

Twierdzenie 3. Jeżeli funkcja f ma $n+1$ ciągły pochodny na przedziale $[a, b]$ to istnieje funkcja $\zeta : [a, b] \rightarrow [a, b]$ taka, że

$$R(f) = \frac{1}{(n+1)!} \int_a^b \rho(x)f^{(n+1)}(\zeta(x)) \tilde{N}_{n+1}(x) dx.$$

Pierwsza część poniższego twierdzenia wynika bezpośrednio z Twierdzenia 3, bo dla dowolnego $x \in [a, b]$

$$|f^{(n+1)}(\zeta(x))| \leq \|f^{(n+1)}\|.$$

Twierdzenie 4. Jeżeli funkcja f ma $n+1$ ciągły pochodny na przedziale $[a, b]$ to

$$|R(f)| \leq \frac{\|f^{(n+1)}\|}{(n+1)!} \int_a^b \rho(x) |\tilde{N}_{n+1}(x)| dx,$$

gdzie

$$\tilde{N}_{n+1}(x) = \prod_{i=0}^m (x - x_i)^{k_i}.$$

Dodatkowo istnieją funkcje f , dla których $\|f^{(n+1)}\| \neq 0$ i w powyższym oszacowaniu nierówność można zastąpić równością.

¹Dokładniej są to *zamknięte* kwadratury Newtona–Cotesa. Rozważa się też kwadratury Newtona–Cotesa do obliczania całek z wagą oraz takie, gdzie węzły są równooddalone, ale krańcowe węzły nie pokrywają się z krańcami przedziału całkowania.

Ponieważ

$$\forall_{x \in [a,b]} |\tilde{N}_{n+1}(x)| \leq (b-a)^{n+1},$$

to możemy zapisać poniższy wniosek.

Wniosek 1. Jeżeli funkcja f ma $n+1$ ciągły pochodnych na przedziale $[a,b]$ to

$$|R(f)| \leq \frac{\|f^{(n+1)}\|}{(n+1)!} (b-a)^{n+1} \int_a^b \rho(x) dx.$$

Dla kwadratur prostokątów, trapezów i parabol i kolejnych kwadratur Newtona–Cotesa można podać następujące oszacowania reszty. Niektóre z nich są dokładniejsze, niż te uzyskane z Wniosku 1 — dotyczy to linii pierwszej, trzeciej, piątej i siódmej. Zwróćmy uwagę, że wtedy wymagana jest też wyższa regularność funkcji niż we Wniosku 1 (istnienie i ciągłość pochodnej, której obliczamy normę).

Kwadratura	Oszacowanie
prostokątów	$R(f) \leq \frac{1}{24} \left(\frac{b-a}{1}\right)^3 \ f''\ $
trapezów — Newtona–Cotesa dla $n=1$	$R(f) \leq \frac{1}{12} \left(\frac{b-a}{1}\right)^3 \ f''\ $
parabol — Newtona–Cotesa dla $n=2$	$R(f) \leq \frac{1}{90} \left(\frac{b-a}{2}\right)^5 \ f^{(4)}\ $
Newtona–Cotesa dla $n=3$	$R(f) \leq \frac{3}{80} \left(\frac{b-a}{3}\right)^5 \ f^{(4)}\ $
Newtona–Cotesa dla $n=4$	$R(f) \leq \frac{8}{945} \left(\frac{b-a}{4}\right)^7 \ f^{(6)}\ $
Newtona–Cotesa dla $n=5$	$R(f) \leq \frac{275}{12096} \left(\frac{b-a}{5}\right)^7 \ f^{(6)}\ $
Newtona–Cotesa dla $n=6$	$R(f) \leq \frac{9}{1400} \left(\frac{b-a}{6}\right)^9 \ f^{(8)}\ $

19.4 Rząd kwadratur interpolacyjnych

Zauważmy, że jeśli $w \in \Pi_n$, to $\forall_{x \in \mathbb{R}} w^{(n+1)}(x) = 0$. Korzystając z Twierdzenia 3 oraz definicji błędu interpolacji otrzymujemy, że jeśli Q jest kwadraturą interpolacyjną, to

$$\forall_{w \in \Pi_n} Q(w) = I(w).$$

Z tego wynika następujący fakt.

Fakt 1. Rząd kwadratury interpolacyjnej jest większy bądź równy $n+1$, gdzie $n+1$ jest sumą krotności węzłów.

Można też łatwo udowodnić własność mówiącą o tym, że kwadratury wysokich rzędów muszą być kwadraturami interpolacyjnymi.

Fakt 2. Jeśli dowolna kwadratura

$$Q(f) = \sum_{i=0}^m \sum_{j=0}^{k_i-1} A_{ij} f^{(j)}(x_i)$$

ma rząd większy bądź równy $n+1$, gdzie $n+1$ jest sumą krotności węzłów, to jest ona kwadratura interpolacyjna oparta na węzłach x_0, x_1, \dots, x_m o krotnościach k_0, k_1, \dots, k_m .

Na koniec przytoczę twierdzenie dotyczące rzędów kwadratur Newtona–Cotesa.

Twierdzenie 5. Rząd kwadratury Newtona–Cotesa opartej na $n+1$ węzłach wynosi $n+2$ dla n parzystych oraz $n+1$ dla n nieparzystych.

19.5 Kwadratury Gaussa

Niech

$$I(f) = \int_a^b \rho(x) f(x) dx.$$

Można postawić pytanie jaki jest w ogóle maksymalny rzząd kwadratury przybliżającej tę całkę? Ponieważ zachodzi Fakt 2, to taka kwadratura musi być interpolacyjna. Jak więc należy dobrąć węzły, by ten maksymalny rzząd uzyskać? Odpowiedź jest w nastepnym twierdzeniu.

Twierdzenie 6. Maksymalny rzząd kwadratury

$$Q(f) = \sum_{i=0}^m \sum_{j=0}^{k_i-1} A_{ij} f^{(j)}(x_i)$$

przybliżającej całkę

$$I(f) = \int_a^b \rho(x) f(x) dx$$

wynosi $2n+2$, gdzie $n+1$ jest sumą krotności węzłów. Istnieje dokładnie jedna taka kwadratura, jest ona oparta na jednokrotnych węzłach, które są pierwiastkami $(n+1)$ -go wielomianu ortogonalnego^a w przestrzeni $L_p^2[a,b]$.

^aCzyli wielomianu ortogonalnego stopnia $n+1$.

Dowód Niech Q będzie kwadraturą maksymalnego rzędu. Oznaczmy go przez r . Z faktów 1 i 2 wynika, że Q musi być kwadraturą interpolacyjną oraz

$$r \geq n+1.$$

Niech x_i , $i = 0, 1, \dots, m$ o krotnościach k_i będą węzłami tej kwadratury oraz

$$p(x) = \prod_{i=1}^m (x - x_i)^{k_i}.$$

Rozpatrzmy dowolny wielomian w , który jest stopnia mniejszego niż r . Przedstawmy w w postaci

$$w(x) = u(x)p(x) + v(x),$$

gdzie u jest wynikiem dzielenia w przez p , a $v \in \Pi_n$ jest resztą z tego dzielenia. Suma stopni u i p jest równa stopniowi w lub $u \equiv 0$, stąd $u \in \Pi_{r-n-2}$. Zauważmy, że

$$\begin{aligned} I(w) &= \int_a^b \rho(x)w(x) dx \\ &= \int_a^b \rho(x)u(x)p(x) dx + \int_a^b \rho(x)v(x) dx \\ &= (u, p) + I(v) \end{aligned}$$

gdzie (\cdot, \cdot) jest iloczynem skalarnym w $L_\rho^2(a, b)$. Ponadto, ponieważ $w, v \in \Pi_{r-1}$, to z definicji rzędu $I(v) = Q(v)$ oraz $I(w) = Q(w)$, więc

$$Q(w) = (u, p) + Q(v).$$

Korzystając z ogólnej postaci kwadratury mamy, że

$$\begin{aligned} Q(w) &= \sum_{i=0}^m \sum_{j=0}^{k_i-1} A_{ij} w^{(j)}(x_i) \\ &= \sum_{i=0}^m \sum_{j=0}^{k_i-1} A_{ij} (u \cdot p)^{(j)}(x_i) + \sum_{i=0}^m \sum_{j=0}^{k_i-1} A_{ij} v^{(j)}(x_i) \\ &= Q(v). \end{aligned}$$

Ostatnia równość jest prawdziwa, bo dla dowolnego i oraz każdego $j = 0, 1, \dots, k_i - 1$ mamy $(u \cdot p)^{(j)}(x_i) = 0$.

Otrzymujemy więc, że $(u, p) = 0$. Ponieważ $w \in \Pi_{r-1}$ był wybrany dowolnie, to dla każdego $u \in \Pi_{r-n-2}$ mamy $(u, p) = 0$. Wielomian p jest stopnia $n+1$ więc równość ta jest możliwa wtedy i tylko wtedy gdy

$$n+1 > r - n - 2$$

oraz p jest $(n+1)$ -szym wielomianem ortogonalnym w $L_\rho^2(a, b)$. Ponieważ rząd r jest maksymalny, to

$$r = 2n + 2,$$

a x_i muszą być węzłami o krotności 1, które są pierwiastkami tego wielomianu.

W twierdzeniu wykorzystuje się własność, że n -ty wielomian ortogonalny w $L_\rho^2[a, b]$ ma n różnych pierwiastków, które leżą w przedziale $[a, b]$. Zatem rzeczywiście jest to kwadratura interpolacyjna przybliżająca całkę $I(f)$. Kwadratury, o których mowa w twierdzeniu nazywamy *kwadraturami Gaussa*.

Konstrukcja kwadratur Gaussa jest następująca.

1. Znajdź węzły x_0, x_1, \dots, x_n , które są pierwiastkami $(n+1)$ -go wielomianu ortogonalnego w przestrzeni $L_\rho^2[a, b]$.
2. Oblicz całkę z wielomianu interpolacyjnego Lagrange'a opartego na tych węzłach.

Otrzymujemy zatem kwadraturę

$$Q(f) = \sum_{i=0}^n A_i f(x_i),$$

dla której

$$A_i = \int_a^b \rho(x)l_i(x) dx.$$

Dla niektórych przestrzeni $L_\rho^2[a, b]$ można podać jawnie wzory opisujące pierwiastki wielomianów ortogonalnych. W innych przypadkach należy posłużyć się metodami przybliżonymi (które nie są omawiane na tym wykładzie).

Wagi też czasami są dane jawnie. Jeśli tak nie jest, to wyznaczenie ich w sposób przybliżony nastręcza nieco kłopotów. Korzystanie bezpośrednio ze wzoru

$$A_k = \int_a^b \rho(x)l_k(x) dx = I(l_k)$$

i stosowanie jakiejś innej kwadratury nie jest dobre pod względem numerycznym. Problem sprawia uwarunkowanie zadania dla funkcji podcałkowych, które nie są nieujemne. Okazuje się jednak, że zachodzi równość

$$A_k = \int_a^b \rho(x)l_k^2(x) dx = I(l_k^2).$$

Rzeczywiście, l_k^2 jest stopnia $2n$, zatem

$$I(l_k^2) = Q(l_k^2) = \sum_{i=0}^n A_i l_k^2(x_i) = A_k.$$

Ten drugi wzór jest lepszy i jest podstawą opracowania metody uzyskania przybliżenia. Zauważmy przy okazji, że A_i są zawsze dodatnie.

Kwadratury Gaussa często stosuje się dla początkowych wartości n , wtedy można węzły i wagi obliczyć „ręcznie”. Poniżej zamieszczam informacje o węzłach i wagach dla kwadratur wykorzystujących wybrane klasyczne wielomiany ortogonalne. Podane są też wzory na resztę interpolacyjną, które w tym wypadku można sprowadzić do prostszej postaci niż wynikająca z Twierdzenia 3. We wszystkich przypadkach zakładają się, że f posiada $2n+2$ ciągłych pochodnych. Do nazwy tych kwadratur dołącza się zwykle nazwę wielomianów ortogonalnych.

Kwadratury Gaussa–Legendre'a stosowane są do obliczania przybliżenia całki

$$I(f) = \int_{-1}^1 f(x) dx.$$

Oznaczmy przez P_i wielomian Legendre'a stopnia i . Liczba ζ należy do przedziału $(-1, 1)$.

Węzły	x_i — pierwiastki P_{n+1}
Wagi	$A_i = -\frac{2}{(n+2)P_{n+2}(x_i)P'_{n+1}(x_i)}$
Reszta	$R(f) = \frac{2^{2n+3}((n+1)!)^4}{(2n+3)((2n+2)!)^3} f^{(2n+2)}(\zeta)$

Wartości węzłów i wag dla $n = 1, 2, 3, 4$

n	i	x_i	A_i
1	0	$-\frac{1}{\sqrt{3}}$	1
	1	$\frac{1}{\sqrt{3}}$	1
2	0	$-\sqrt{\frac{3}{5}}$	$\frac{5}{9}$
	1	0	$\frac{8}{9}$
	2	$\sqrt{\frac{3}{5}}$	$\frac{5}{9}$
3	0	≈ -0.861136	≈ 0.347855
	1	≈ -0.339981	≈ 0.652145
	2	≈ 0.339981	≈ 0.652145
	3	≈ 0.861136	≈ 0.347855
4	0	≈ -0.906180	≈ 0.236927
	1	≈ -0.538469	≈ 0.478629
	2	0	≈ 0.568889
	3	≈ 0.538469	≈ 0.478629
	4	≈ 0.906180	≈ 0.236927

n	i	x_i	A_i
1	0	$-\frac{1}{\sqrt{2}}$	$\frac{\sqrt{\pi}}{2}$
	1	$\frac{1}{\sqrt{2}}$	$\frac{\sqrt{\pi}}{2}$
2	0	$-\sqrt{\frac{3}{2}}$	$\frac{\sqrt{\pi}}{6}$
	1	0	$\frac{2\sqrt{\pi}}{3}$
	2	$\sqrt{\frac{3}{2}}$	$\frac{\sqrt{\pi}}{6}$
3	0	$-\sqrt{\frac{3+\sqrt{6}}{2}}$	≈ 0.081313
	1	$-\sqrt{\frac{3-\sqrt{6}}{2}}$	≈ 0.804914
	2	$\sqrt{\frac{3-\sqrt{6}}{2}}$	≈ 0.804914
	3	$\sqrt{\frac{3+\sqrt{6}}{2}}$	≈ 0.081313
4	0	≈ -2.020183	≈ 0.019953
	1	≈ -0.958572	≈ 0.393619
	2	0	≈ 0.945309
	3	≈ 0.958572	≈ 0.393619
	4	≈ 2.020183	≈ 0.019953

Kwadratury Gaussa–Czebyszewa stosowane są do obliczania całki

$$I(f) = \int_{-1}^1 \frac{1}{\sqrt{1-x^2}} f(x) dx$$

Niech T_i będzie wielomianem Czebyszewa stopnia i . Liczba ζ należy do przedziału $(-1, 1)$.

Węzły	$x_i = \cos \frac{(2i+1)\pi}{2n+2}$ — pierwiastki T_{n+1}
Wagi	$A_i = \frac{\pi}{n+1}$
Reszta	$R(f) = \frac{\pi}{2^{2n+1}(2n+2)!} f^{(2n+2)}(\zeta)$

Kwadratury Gaussa–Hermite'a służą do obliczania przybliżenia całki

$$I(f) = \int_{-\infty}^{\infty} e^{-x^2} f(x) dx$$

Przez H_i oznaczam i -ty wielomian Hermite'a. Liczba ζ jest rzeczywista.

Węzły	x_i — pierwiastki H_{n+1}
Wagi	$A_i = -\frac{2^{n+2}(n+1)!\sqrt{\pi}}{H'_{n+1}(x_i)H_{n+2}(x_i)}$
Reszta	$R(f) = \frac{\sqrt{\pi}(n+1)!}{2^{n+1}(2n+2)!} f^{(2n+2)}(\zeta)$

Wartości węzłów i wag dla $n = 1, 2, 3, 4$

Kwadratury Gaussa–Laguerre'a stosowane są do obliczania przybliżenia całki

$$I(f) = \int_0^\infty e^{-x} f(x) dx$$

Wielomiany Laguerre'a oznaczam przez L_i , $\zeta \in (0, \infty)$.

Węzły	x_i — pierwiastki L_{n+1}
Wagi	$A_i = -\frac{((n+1)!)^2}{L'_{n+1}(x_i)L_{n+2}(x_i)}$
Reszta	$R(f) = \frac{((n+1)!)^2}{(2n+2)!} f^{(2n+2)}(\zeta)$

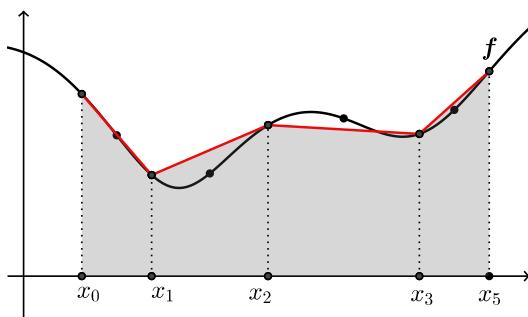
n	i	x_i	A_i
1	0	$2 - \sqrt{2}$	$\frac{2+\sqrt{2}}{4}$
	1	$2 + \sqrt{2}$	$\frac{2-\sqrt{2}}{4}$
2	0	≈ 0.415775	≈ 0.711093
	1	≈ 2.294280	≈ 0.278518
	2	≈ 6.289945	≈ 0.010389
3	0	≈ 0.322548	≈ 0.603154
	1	≈ 1.745761	≈ 0.357419
	2	≈ 4.536620	≈ 0.038888
	3	≈ 9.395071	≈ 0.000539
4	0	≈ 0.263560	≈ 0.521756
	1	≈ 1.413403	≈ 0.398667
	2	≈ 3.596426	≈ 0.075942
	3	≈ 7.085810	≈ 0.003612
	4	≈ 12.640801	≈ 0.000032

20 Kwadratury złożone (powtarzalne)

20.1 Zasada budowy

Wzory na resztę kwadratur bądź jej oszacowanie zawierają czynniki zależne od kolejnych pochodnych funkcji, rozmiiesz-

czenia węzłów, długości przedziału $[a, b]$. Jeśli funkcje nie posiadają wielu pochodnych, lub ich pochodne mają coraz większe normy supremum, to zwiększenie liczby węzłów przy kwadraturach interpolacyjnych niekoniecznie prowadzi do lepszych rezultatów. Zależność błędu kwadratury od długości przedziału nasuwa pomysł, by przedział całkowania podzielić na mniejsze odcinki i w każdym z nich zastosować kwadraturę interpolacyjną (tego samego typu). Taki sposób postępowania prowadzi do budowy *kwadratur złożonych*.



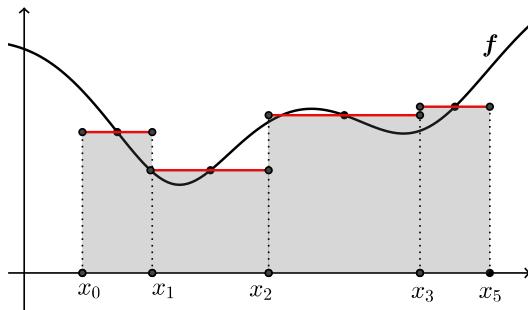
20.2 Przykłady kwadratur złożonych

Przybliżamy całkę

$$I(f) = \int_a^b f(x) dx.$$

Przedział $[a, b]$ dzielimy na N części, w każdej z nich stosujemy kwadraturę interpolacyjną (do każdej części tego samego typu). Wyprowadzenie wzorów zostawiamy jako ćwiczenie.

Kwadratura złożona prostokątów W każdym przedziale stosujemy kwadraturę prostokątów. Kwadratura jest oparta na N węzłach.



Przy podziale na dowolne odcinki $x_0 = a$, $x_N = b$

$$P_N(f) = \sum_{i=0}^{N-1} (x_{i+1} - x_i) f\left(\frac{x_i + x_{i+1}}{2}\right),$$

przy odcinkach równych wyznaczonych przez liczby $x_i = a + ih$, gdzie $h = \frac{b-a}{N}$, $i = 0, 1, \dots, N$

$$P_N(f) = h \sum_{i=0}^{N-1} f\left(a + h \frac{2i+1}{2}\right).$$

Zauważmy, że węzłami tej kwadratury nie są liczby x_i , lecz $\frac{x_i+x_{i+1}}{2}$.

Kwadratura złożona trapezów W każdym przedziale stosujemy kwadraturę trapezów. Kwadratura jest oparta na $N + 1$ węzłach.

Przy podziale na dowolne odcinki, $x_0 = a$, $x_N = b$

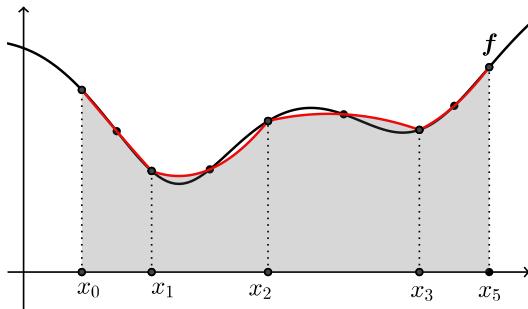
$$\begin{aligned} T_N(f) &= \frac{1}{2}(x_1 - x_0)f(x_0) + \frac{1}{2}(x_N - x_{N-1})f(x_N) \\ &\quad + \frac{1}{2} \sum_{i=1}^{N-1} (x_{i+1} - x_{i-1})f(x_i) \end{aligned}$$

przy odcinkach równych wyznaczonych przez liczby $x_i = a + ih$, gdzie $h = \frac{b-a}{N}$, $i = 0, 1, \dots, N$

$$T_N(f) = h \left[\frac{f(a) + f(b)}{2} + \sum_{i=1}^{N-1} f(a + ih) \right].$$

Węzłami są tu liczby x_i .

Kwadratura złożona parabol (Simpsona) W każdym przedziale stosujemy kwadraturę parabol. Kwadratura jest oparta na $2N + 1$ węzłach.



Kwadraturę tę można wyrazić przez złożone kwadratury prostokątów i trapezów:

$$S_N(f) = \frac{1}{3}T_N(f) + \frac{2}{3}P_N(f).$$

Przy podziale na odcinki wyznaczone przez liczby $x_i = a + ih$, gdzie $h = \frac{b-a}{N}$, $i = 0, 1, \dots, N$

$$S_N(f) = \frac{h}{3} \left[\frac{f(a) + f(b)}{2} + \sum_{i=1}^{N-1} f(a + ih) + 2 \sum_{i=0}^{N-1} f\left(a + h \frac{2i+1}{2}\right) \right].$$

Węzłami są tu liczby x_i i $\frac{x_i+x_{i+1}}{2}$.

20.3 Złożone kwadratury Newtona–Cotesa

Złożonymi kwadraturami Newtona–Cotesa nazywamy takie kwadratury złożone, w którym przedział $[a, b]$ dzielimy na

N równych odcinków i w każdym stosujemy kwadraturę Newtona–Cotesa stopnia n . Oznacza to wybór w każdym z N odcinków $n+1$ równoodległych węzłów, zatem otrzymamy $Nn+1$ węzłów (w każdym z N odcinków $n+1$ węzłów, ale wewnętrznych $N-1$ się powtarza). Przykładami złożonych kwadratur Newtona–Cotesa są T_N i S_N (przy podziale na równe odcinki).

Można udowodnić, że przy N dążącym do nieskończoności złożone kwadratury Newtona–Cotesa są zbieżne dla dowolnej funkcji ciągłej. Poniżej podaję oszacowania błędów dla początkowych n . Dodatkowo dołączam oszacowanie dla kwadratury prostokątów choć nie jest ona Newtona–Cotesa.

Kwadratura	Oszacowanie
złożona prostokątów	$R(f) \leq \frac{b-a}{24} \left(\frac{b-a}{N}\right)^2 \ f''\ $
złożona Newtona–Cotesa dla $n=1$ (trapezów)	$R(f) \leq \frac{b-a}{12} \left(\frac{b-a}{N}\right)^2 \ f''\ $
złożona Newtona–Cotesa dla $n=2$ (parabol)	$R(f) \leq \frac{b-a}{180} \left(\frac{b-a}{2N}\right)^4 \ f^{(4)}\ $
złożona Newtona–Cotesa dla $n=3$	$R(f) \leq \frac{b-a}{80} \left(\frac{b-a}{3N}\right)^4 \ f^{(4)}\ $
złożona Newtona–Cotesa dla $n=4$	$R(f) \leq \frac{2(b-a)}{945} \left(\frac{b-a}{4N}\right)^6 \ f^{(6)}\ $
złożona Newtona–Cotesa dla $n=5$	$R(f) \leq \frac{55(b-a)}{12096} \left(\frac{b-a}{5N}\right)^6 \ f^{(6)}\ $
złożona Newtona–Cotesa dla $n=6$	$R(f) \leq \frac{3(b-a)}{2800} \left(\frac{b-a}{6N}\right)^8 \ f^{(8)}\ $

20.4 Kwadratury złożone Gaussa

Podobne postępowanie jak poprzednio można powtórzyć dla kwadratur Gaussa. Dzielimy przedział całkowania $[a, b]$ na N równych części i w każdym z nich stosujemy kwadraturę Gaussa (tę samą). Aby móc to zrobić należy w każdym podprzedziale zastosować liniową zamianę zmiennych tak, aby go sprowadzić do właściwego dla wybranej kwadratury Gaussa (np. dla Gaussa–Czebyszewa lub Gaussa–Legendre'a jest to $[-1, 1]$).

Rezultatem dla kwadratur Gaussa–Legendre'a oraz Gaussa–Czebyszewa przy $n=1$ jest wzór

$$Q_N(f) = Ah \sum_{i=0}^{N-1} \left[f(a+h(2i+1-\alpha)) + f(a+h(2i+1+\alpha)) \right],$$

gdzie $h = \frac{b-a}{2N}$, przy czym $A = 1$, $\alpha = \frac{1}{\sqrt{3}}$ dla Legendre'a oraz $A = \frac{\pi}{2}$, $\alpha = \frac{1}{\sqrt{2}}$ dla Czebyszewa. Obie kwadratury mają po $2N$ węzłów.

Oszacowanie błędu dla kwadratury Gaussa–Legendre'a przy $n=1$ ma postać

$$R(f) \leq \frac{b-a}{270} \left(\frac{b-a}{2N}\right)^4 \|f^{(4)}\|.$$

Równania nieliniowe i układy równań nieliniowych

Dorota Dąbrowska, UKSW

2017/18

Spis treści

32 Równania nieliniowe	144
32.1 Sformułowanie zadania	144
32.2 Metoda stycznych	144
32.3 Kula i promień zbieżności	144
32.4 Wykładnik zbieżności	145
32.5 Wykładnik zbieżności dla metody stycznych . .	145
32.6 Metoda siecznych	146
32.7 Metoda bisekcji	146
32.8 Metody Dekkera oraz Brenta	147
32.9 Metody interpolacyjne	147
32.10 Uwagi o obliczaniu zer wielokrotnych	147
32.11 Właściwości numeryczne metod rozwiązywania równań nieliniowych	148
32.12 Kryteria stopu	148
32.13 Uwagi o metodach zbieżnych globalnie	148
33 Wyznaczanie zer wielomianów	149
33.1 Uwarunkowanie zadania wyznaczania zer wie- lomianów	149
33.2 Lokalizacja zer wielomianu	149
33.2.1 Liczba pierwiastków rzeczywistych . .	149
33.2.2 Lokalizacja zer rzeczywistych	150
33.2.3 Lokalizacja zer zespolonych	151
33.3 Metody przybliżonego wyznaczania zer wielo- mianów	151
33.4 Deflacja	152
34 Układy równań nieliniowych	152
34.1 Sformułowanie zadania	152
34.2 Macierz Jacobiego	153
34.3 Metoda Newtona	153
34.4 Metoda siecznych	153
34.5 Rozwiązywanie układów równań nieliniowych w Octave	154
34.5.1 Implementacja funkcji \vec{f} w Octave . .	154

32 Równania nieliniowe

32.1 Sformułowanie zadania

Dział ten zajmuje się rozwiązywaniem w sposób przybliżony równań nieliniowych. Zadanie jest sformułowane następująco.

Definicja 1. Dana jest funkcja $f : D \rightarrow \mathbb{R}$, gdzie $D \subset \mathbb{R}$. Równaniem nieliniowym nazywamy równanie postaci

$$f(x) = 0,$$

gdzie f nie jest funkcją liniową. Rozwiązaniem równania nazywamy każdą liczbę $\alpha \in D$, taką że

$$f(\alpha) = 0.$$

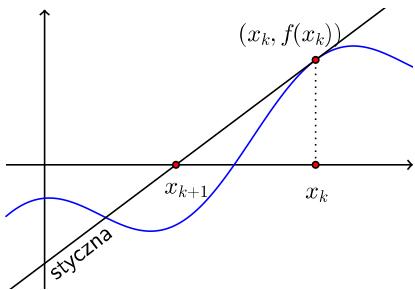
Jeśli nie zakładalibyśmy nic o funkcji f , to niemożliwe byłoby podanie jakichkolwiek metod. Zazwyczaj wymaga się, by funkcja była regularna (czyli ciągła lub kilkakrotnie różniczkowalna) przynajmniej w otoczeniu miejsca zerowego. Rodzaj regularności zależy od stosowanej metody.

Większość metod polega na poprawianiu kolejnych przybliżeń pierwiastka. Startując one z jakiegoś początkowego przybliżenia x_0 rozwiązania i generując ciąg x_k zbieżny do miejsca zerowego. Kolejne przybliżenie generuje się wykorzystując wartości ostatniego lub kilku ostatnich przybliżeń. W drugim przypadku potrzeba kilku punktów startowych.

Metody te często dają ciągi zbieżne do miejsca zerowego jedynie wtedy, gdy zaczynamy od „dobrego” punktów startowych. Nie dla każdego zadania potrafimy te „dobre” punkty wyznaczyć.

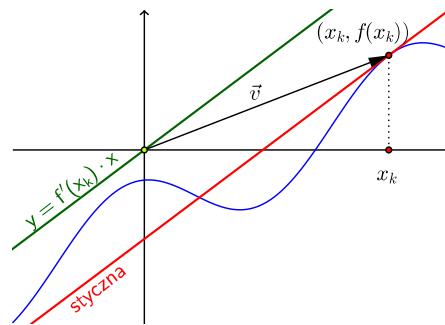
32.2 Metoda stycznych

Metoda ta nazywana jest również *metodą Newtona*. Jej działanie jest następujące. Wybieramy startowy punkt x_0 . W kolejnych krokach mając obliczone przybliżenie x_k generujemy następne przybliżenie x_{k+1} . W tym celu wyznaczamy styczną do wykresu funkcji f w punkcie $(x_k, f(x_k))$. Następnie obliczamy jej miejsce zerowe i ono staje się liczbą x_{k+1} .



Równanie stycznej można wyznaczyć następująco. Wiemy, że współczynnikiem kierunkowym stycznej do wykresu funkcji f w punkcie $(x_k, f(x_k))$ jest $f'(x_k)$. Szukana styczna jest więc równoległa do prostej

$$y = f'(x_k) \cdot x.$$



Aby otrzymać styczną trzeba prostą $y = f'(x_k)x$ przesunąć o wektor $\vec{v} = [x_k, f(x_k)]$. Zatem równaniem stycznej jest

$$y = f'(x_k)(x - x_k) + f(x_k)$$

Obliczmy miejsce zerowe stycznej (przy założeniu, że ono istnieje).

$$\begin{aligned} f'(x_k)(x - x_k) + f(x_k) &= 0 \\ f'(x_k)(x - x_k) &= -f(x_k) \\ x - x_k &= -\frac{f(x_k)}{f'(x_k)} \\ x &= x_k - \frac{f(x_k)}{f'(x_k)} \end{aligned}$$

Zauważmy, że obliczone miejsce zerowe istnieje wtedy i tylko wtedy, gdy $f'(x_k) \neq 0$. Punkt x_{k+1} jest z definicji tym miejscem zerowym, więc

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}.$$

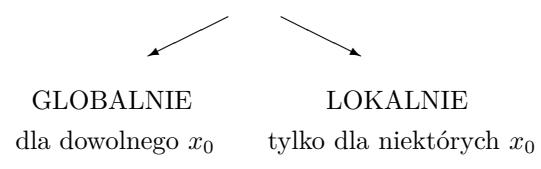
Do obliczenia x_{k+1} potrzeba znać wartość funkcji i pochodnej w punkcie x_k .

Metoda ma zastosowanie jedynie dla funkcji, dla których potrafimy wyznaczyć wartość pochodnej. Dodatkowo musi być zagwarantowane, że w każdym kroku x_k należy do dziedziny funkcji f oraz $f'(x_k) \neq 0$. Metoda ta nie zawsze jest zbieżna — tzn. nie dla każdego przybliżenia początkowego.

32.3 Kula i promień zbieżności

Okazuje się, że tylko nieliczne metody mają własność, że startując z dowolnego przybliżenia początkowego x_0 generują ciąg x_k zbieżny do rozwiązania α . Jeśli metoda posiada taką cechę to mówimy, że jest *zbieżna globalnie*. W innych przypadkach określa się warunki, jakie powinno spełniać przybliżenie początkowe i mówimy wówczas o metodach *zbieżnych lokalnie*.

METODA ZBIEŻNA



metoda generuje ciąg x_k zbieżny do rozwiązania α

Definicja 2. Niech funkcja f należy do pewnej klasy funkcji F , a α oznacza rozwiązanie równania

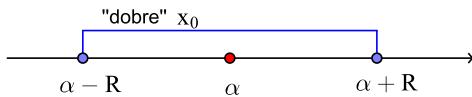
$$f(x) = 0.$$

Dla tej funkcji rozważmy wszystkie liczby $r \geq 0$ takie, że dla dowolnego przybliżenia początkowego

$$x_0 \in [\alpha - r, \alpha + r],$$

ciąg kolejnych przybliżeń x_k generowany daną metodą jest zbieżny do α . Wybierzmy największe takie r i oznaczmy przez R .

Przedział $[\alpha - R, \alpha + R]$ nazywamy *kulą zbieżności* danej metody dla funkcji f , a R *promieniem zbieżności* dla f .



Jeśli promień zbieżności wynosi R i $x_0 \in [\alpha - R, \alpha + R]$, to mamy gwarancję, że metoda znajdzie rozwiązanie α .

Klasę F ustala się osobno dla każdej metody. Na przykład F może być klasą funkcji posiadających dwie ciągle pochodne. Ustalenie klasy jest równoznaczne z podaniem założeń o funkcji f takich, by można było zastosować daną metodę.

Wyznaczenie promienia zbieżności jest trudne dla większości metod. Dlatego zadowalamy się jego oszacowaniami.

32.4 Wykładnik zbieżności

Wykładnik zbieżności metody opisuje szybkość zbieżności ciągów generowanych przez daną metodę.

Definicja 3. Wykładnikiem zbieżności metody nazywamy największą liczbę p spełniającą nierówność

$$|x_{k+1} - \alpha| \leq A|x_k - \alpha|^p,$$

dla pewnej stałej A (być może zależnej od f i α) i wszystkich dostatecznie dużych k .

Jeśli $p = 1$ to

$$|x_k - \alpha| \leq A|x_{k-1} - \alpha| \leq A^2|x_{k-2} - \alpha| \leq \dots \leq A^k|x_0 - \alpha|.$$

Gdy $A < 1$ to

$$\lim_{k \rightarrow \infty} A^k = 0,$$

zatem

$$\lim_{k \rightarrow \infty} x_k = \alpha.$$

Definicja 4. Mówimy, że metoda jest *zbieżna liniowo z ilorazem A* jeśli $p = 1$ i A jest stałą występującą w definicji wykładnika zbieżności mniejszą od 1.

Zauważmy, że jeśli A nie zależy od f to metoda jest zbieżna globalnie.

Jeśli $p > 1$, to otrzymujemy

$$\begin{aligned} |x_k - \alpha| &\leq A|x_{k-1} - \alpha|^p \leq A(A|x_{k-2} - \alpha|^p)^p \\ &= A^{1+p}|x_{k-2} - \alpha|^{p^2} \\ &\dots \\ &\leq A^{1+p+p^2+\dots+p^{k-1}}|x_0 - \alpha|^{p^k} = A^{\frac{p^k-1}{p-1}}|x_0 - \alpha|^{p^k} \\ &= \left(A^{\frac{1}{p-1}}|x_0 - \alpha|\right)^{p^k-1}|x_0 - \alpha|. \end{aligned}$$

Gdy $A^{\frac{1}{p-1}}|x_0 - \alpha| < 1$, to ciąg x_k jest zbieżny do α . Zatem promień zbieżności wynosi co najmniej $A^{-\frac{1}{p-1}}$.

Wykładnik zbieżności bardzo często zależy od krotności wyznaczanego miejsca zerowego.

Definicja 5. Mówimy, że α jest *miejscem zerowym f o krotności m*, jeśli f można przedstawić w postaci

$$f(x) = (x - \alpha)^m g(x),$$

gdzie $g(\alpha) \neq 0$ i g jest ograniczona w pewnym otoczeniu punktu α .

Można udowodnić poniższe twierdzenie.

Twierdzenie 1. Niech f posiada m ciągłych pochodnych. Liczba α jest miejscem zerowym f o krotności m wtedy i tylko wtedy, gdy

$$f(\alpha) = f'(\alpha) = \dots = f^{(m-1)}(\alpha) = 0$$

oraz

$$f^{(m)}(\alpha) \neq 0.$$

32.5 Wykładnik zbieżności dla metody stycznych

Poniższe twierdzenie opisuje szybkość zbieżności metody Newtona w przypadku, gdy szukane miejsce zerowe jest jednokrotne.

Twierdzenie 2. Niech funkcja f posiada dwie ciągle pochodne w przedziale P zawierającym miejsce zerowe α funkcji f oraz

$$\inf_{x \in P} |f'(x)| > 0.$$

Niech r oznacza wielkość

$$r = 2 \frac{\inf_{x \in P} |f'(x)|}{\sup_{x \in P} |f''(x)|}.$$

Wtedy jeśli $|x_0 - \alpha| < r$ to ciąg x_k generowany przez metodę Newtona jest zbieżny do α i

$$|x_{k+1} - \alpha| \leq \frac{1}{r} |x_k - \alpha|^2.$$

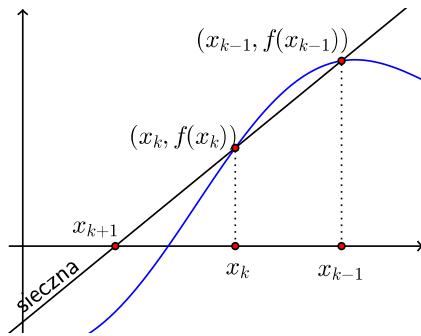
Wykładnik zbieżności metody Newtona wynosi zatem 2, a promień zbieżności co najmniej r . Z nierówności

$$|x_{k+1} - \alpha| \leq \frac{1}{r} |x_k - \alpha|^2$$

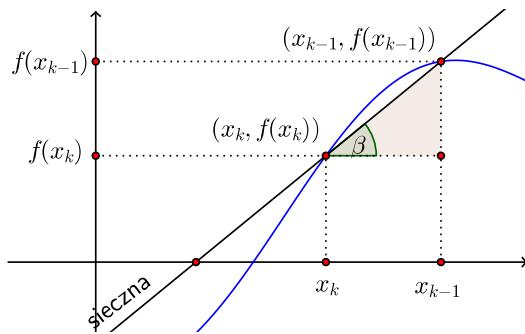
wynika, że jeśli x_k jest przybliżeniem α , które ma n cyfr znaczących dokładnych, to x_{k+1} ma mniej więcej $2n$ cyfr znaczących dokładnych. Gdy więc x_0 ma 1 cyfrę znaczącą dokładną, to po wykonaniu czterech kroków możemy się spodziewać, że x_4 będzie miało 16 cyfr znaczących dokładnych. Jest to bardzo szybka zbieżność.

32.6 Metoda siecznych

Metoda siecznych pozwala uniknąć obliczania pochodnych funkcji. Przybliżenie x_{k+1} konstruuje się na podstawie przybliżeń x_k oraz x_{k-1} , jest ono miejscem zerowym siecznej przechodzącej przez punkty $(x_k, f(x_k))$ i $(x_{k-1}, f(x_{k-1}))$.



Równanie siecznej wyznaczmy następująco. Z zaznaczonego poniżej trójkąta odczytujemy współczynnik kierunkowy siecznej, który wynosi $\tan(\beta)$.

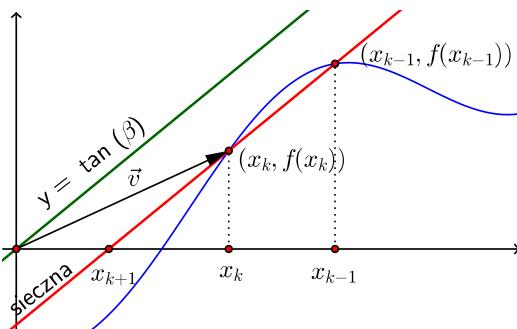


$$\tan(\beta) = \frac{f(x_{k-1}) - f(x_k)}{x_{k-1} - x_k} = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

Aby otrzymać sieczną trzeba prostą

$$y = \tan(\beta) \cdot x = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}} \cdot x$$

przesunąć o wektor $\vec{v} = [x_k, f(x_k)]$.



Zatem równaniem siecznej jest

$$y = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}(x - x_k) + f(x_k).$$

Obliczmy miejsce zerowe siecznej (przy założeniu, że ono istnieje).

$$\begin{aligned} \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}(x - x_k) + f(x_k) &= 0 \\ \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}(x - x_k) &= -f(x_k) \\ x - x_k &= -\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \cdot f(x_k) \\ x &= x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \cdot f(x_k) \end{aligned}$$

Otrzymujemy więc,

$$\begin{aligned} x_{k+1} &= x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k) \\ &= \frac{x_{k-1}f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})}. \end{aligned}$$

Można udowodnić, że wykładnik zbieżności wynosi

$$p = \frac{1 + \sqrt{5}}{2} \approx 1.618,$$

przy założeniu, że miejsce zerowe jest jednokrotne.

Twierdzenie 3. Niech funkcja f posiada dwie ciągłe pochodne w przedziale P zawierającym miejsce zerowe α funkcji f oraz

$$\inf_{x \in P} |f'(x)| > 0.$$

Niech r oznacza wielkość

$$r = 2 \frac{\inf_{x \in P} |f'(x)|}{\sup_{x \in P} |f''(x)|}.$$

Wtedy jeśli $|x_0 - \alpha| < r$ i $|x_1 - \alpha| < r$ to ciąg x_k generowany przez metodę siecznych jest zbieżny do α i istnieje $A > 0$ takie, że

$$|x_{k+1} - \alpha| \leq A|x_k - \alpha|^{\frac{1+\sqrt{5}}{2}}.$$

Z twierdzenia wynika, że jeśli x_k jest przybliżeniem α , które ma n cyfr znaczących dokładnych, to x_{k+1} ma mniej więcej $1.618n$ cyfr znaczących dokładnych. Gdy więc x_0 ma 1 cyfrę znaczącą dokładną, to po wykonaniu czterech kroków możemy się spodziewać, że x_4 będzie miało 6 lub 7 cyfr znaczących dokładnych, a żeby uzyskać 15 cyfr dokładnych trzeba wykonać około 6 kroków. Zbieżność jest wolniejsza niż w metodzie Newtona, ale wykonanie pojedynczego kroku jest szybsze, bo nie trzeba obliczać pochodnych. Szacuje się, że jeśli koszt policzenia pochodnej przewyższa 0.44 kosztu obliczenia wartości funkcji, to bardziej opłacalna czasowo będzie metoda siecznych.

32.7 Metoda bisekcji

Jeśli funkcja f jest ciągła i umiemy znaleźć przedział $[a, b]$ taki, że f na jego krańcach ma różne znaki, to do wyznaczenia

miejsca zerowego α leżącego wewnątrz $[a, b]$ możemy zastosować metodę bisekcji. Korzysta ona z twierdzenia o przyjmowaniu wartości pośrednich.

Twierdzenie 4. Niech f będzie funkcją ciągłą określona na przedziale $[a, b]$.

1. Jeśli $f(a) < c < f(b)$, to istnieje $x \in (a, b)$ taki, że

$$f(x) = c.$$

2. Jeśli $f(b) < c < f(a)$, to istnieje $x \in (a, b)$ taki, że

$$f(x) = c.$$

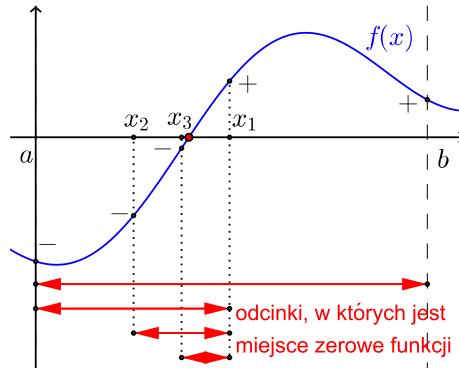
Jeśli więc f na krańcach $[a, b]$ ma różne znaki, to biorąc w twierdzeniu $c = 0$ otrzymujemy, że istnieje $\alpha \in (a, b)$ takie, że

$$f(\alpha) = 0.$$

Zauważmy, że f może mieć kilka miejsc zerowych w (a, b) .

Metoda bisekcji konstruuje ciąg x_k następująco.

lewy = a ;
 prawy = b ;
 dla $k = 0, 1, 2, \dots$
 {
 $x_k = (\text{lewy} + \text{prawy})/2$;
 jeśli ($\text{znak}(f(x_k)) == \text{znak}(f(\text{lewy}))$) to
 $\text{lewy} = x_k$
 w przeciwnym przypadku
 $\text{prawy} = x_k$
 }
 }



Zauważmy, że w każdym kroku $\alpha \in [\text{lewy}, \text{prawy}]$, przy czym za każdym razem długość przedziału $[\text{lewy}, \text{prawy}]$ zmniejsza się o połowę. Zatem

$$|x_{k+1} - \alpha| \leq \frac{1}{2}|x_k - \alpha|.$$

Stąd wynika poniższe twierdzenie.

Twierdzenie 5. Metoda bisekcji jest zbieżna liniowo z ilorazem $A = \frac{1}{2}$.

32.8 Metody Dekkera oraz Brenta

Łącząc metodę siecznych i bisekcji. Dzięki temu są ona zbieżna globalnie, a jednocześnie istotnie szybciej niż liniowo.

Tak jak w metodzie bisekcji trzeba podać przedział $[a, b]$ tak, że $f(a)$ i $f(b)$ mają przeciwnie znaki. Metoda Dekkera pochodzi z 1969 roku. Metoda Brenta jest cztery lata młodsza i jest modyfikacją metody Dekkera.

Funkcja Octave'a `fzero` implementuje metodę Brenta¹.

32.9 Metody interpolacyjne

Definicja 6. Metoda nazywana jest interpolacyjną, jeśli kolejne przybliżenie x_k jest miejscem zerowym pewnego wielomianu interpolującego funkcję f .

Stosowana jest zarówno interpolacja Lagrange'a, jak i Hermite'a. Wielomiany powinny być niskich stopni, tak aby łatwo można było wyznaczyć ich pierwiastki — stąd praktycznie ograniczamy się do wielomianów pierwszego i drugiego stopnia. Zauważmy, że metody siecznych i stycznych są metodami interpolacyjnymi:

- metoda stycznych — x_k jest zerem wielomianu Hermite'a interpolującego funkcję f w podwójnym węźle x_{k-1} .
- metoda siecznych — x_k wyznaczamy jako miejsce zerowe wielomianu Lagrange'a interpolującego f w węzłach x_{k-1} oraz x_{k-2} .

Inną metodą interpolacyjną jest np. *metoda Mullera*. Punkt x_k oblicza się jako miejsce zerowe wielomianu Lagrange'a opartego na węzłach x_k, x_{k-1}, x_{k-2} interpolującego f . Jest to wielomian drugiego stopnia, ma zatem dwa pierwiastki, być może zespolone. Należy wybrać ten pierwiastek, który jest bliższy punktowi x_k . Należy zwrócić uwagę na realizację tego algorytmu przy obliczaniu pierwiastka równania kwadratowego.

Metoda Mullera ma wykładnik zbieżności co najmniej 1.84 dla zer jednokrotnych.

32.10 Uwagi o obliczaniu zer wielokrotnych

Jeśli szukane miejsce zerowe jest zerem wielokrotnym, to opisane metody nie muszą być zbieżne, a jeśli są zbieżne, to zazwyczaj mają mniejszy wykładnik zbieżności. Niech dalej m oznacza krotność pierwiastka α funkcji f .

Przykład Metoda stycznych. W klasie funkcji mających $(m+2)$ pochodną ciągłą wykładnik zbieżności wynosi co najwyżej 1, iloraz zbieżności $A = 1 - \frac{1}{m} \rightarrow 1$, gdy $m \rightarrow \infty$.

Przykład Metoda siecznych. W klasie funkcji mających $(m+2)$ pochodną ciągłą zbieżność nie jest zagwarantowana, nawet dla bliskich punktów startowych. Wykładnik zbieżności co najwyżej 1.

Aby uniknąć osłabienia zbieżności wynikającego z krotności zera stosuje się znane metody nie do funkcji f , lecz do innej, dla której α jest zerem jednokrotnym. Przypomnijmy, że jeśli α jest zerem m krotnym, to f można zapisać w postaci

$$f(x) = (x - \alpha)^m g(x), \quad g(\alpha) \neq 0.$$

¹Opis metody jest na stronie [en.wikipedia.org/wiki/Brent's_method](https://en.wikipedia.org/wiki/Brent%27s_method).

Zatem funkcja

$$u(x) = \sqrt[m]{f(x)},$$

o ile m jest nieparzyste, ma jednokrotne zero α . Zamiast stosować metody dla f , stosujemy je dla funkcji u . Poważną niedogodnością tego podejścia jest fakt, że musimy znać krotność miejsca zerowego α . Jeśli taką informacją nie dysponujemy, możemy zamiast f użyć funkcji

$$v(x) = \frac{f(x)}{f'(x)}.$$

Z definicji krotności wynika bowiem, że f' ma zero ($m - 1$) krotne i stąd v ma zero jednokrotne. Ta druga metoda jest droższa.

32.11 Własności numeryczne metod rozwiązywania równań nieliniowych

Dla omawianego zadania precyzuje się pojęcia uwarunkowania zadania, numerycznej poprawności i stabilności algorytmów. Okazuje się m.in., że w przypadku pojedynczego równania nieliniowego pojęcia numerycznej stabilności i poprawności są równoważne, tzn. jeśli algorytm jest numerycznie stabilny to można udowodnić, że jest numerycznie poprawny. Właściwość taka jest rzadko spotykana. Można udowodnić, że algorytmy realizujące metodę stycznych i siecznych zgodnie z przedstawionym zapisem są numerycznie poprawne.

Z metodami iteracyjnymi (tzn. takimi, które generują ciągi kolejnych przybliżeń rozwiązania) wiąże się pojęcie *maksymalnej granicznej dokładności*. Przypomnijmy, że wykonując obliczenia w arytmetyce zmiennopozycyjnej nie uzyskujemy wyników dokładnych, lecz zaburzone. Zatem każdy algorytm realizujący pewną metodę daje zamiast x_1 wielkość \tilde{x}_1 nieco zaburzoną. Następny krok startuje zatem już z innego punktu niż powinien i znów obarcza go błędami obliczeń. Otrzymujemy zatem ciąg \tilde{x}_k zamiast x_k . Błędy się kumulują i ciąg \tilde{x}_k zazwyczaj nie jest zbieżny do α . Tak na prawdę w ogóle nie musi być zbieżny — może „błądzić” w okolicy rozwiązania α . Na szczęście często daje się ustalić najmniejszą liczbę ε taką, że dla dużych n zachodzi $|\tilde{x}_k - \alpha| \leq \varepsilon$. Tą liczbę nazywamy *maksymalną graniczną dokładnością*.

Żądanie osiągnięcia wyższej dokładności na ogół kończy się niepowodzeniem. Programując metodę iteracyjną należy przerwać obliczenia, jeśli ta dokładność osiągniemy. Ponieważ nie znamy α (często wielkość ε też nie jest wyznaczona dokładnie), sytuację osiągnięcia maksymalnej granicznej dokładności wykrywa się w inny sposób — obserwując zachowanie ciągu kolejnych przybliżeń. Np. jeśli zaczyna „błądzić”, to jest to najczęściej sygnał do przerwania obliczeń.

32.12 Kryteria stopu

W omawianych metodach został podany sposób generowania ciągu x_k przybliżeń rozwiązania α . Nie zostało jednak wskazane kryterium zatrzymania procesu. W pewnym momencie należy stwierdzić, że albo rozwiązanie zostało uzyskane, albo, że nie da się obliczyć miejsca zerowego. Kryterium stopu zwykle formułuje się nie przy pomocy jednego, lecz kilku warunków.

Pierwszym z nich może być *kryterium liczby kroków*. Określa się wtedy z góry, że poszukiwanie rozwiązania jest przerwane po pewnej liczbie kroków. Uznajemy, że metoda nie znalazła rozwiązania (przyczyną może być np. złe przybliżenie początkowe, zły dobór metody dla funkcji, brak rozwiązania równania). Znając szybkość zbieżności metody możemy mieć pewne przesłanki do ustalenia maksymalnej liczby kroków. Przykładowo szukając rozwiązań w arytmetyce, w której reprezentowane jest dokładnie 16 cyfr dziesiętnych i stosując metodę Newtona spodziewamy się, że o ile startowaliśmy z punktu początkowego posiadającego jedną cyfrę dokładną, to rozwiązanie powinniśmy uzyskać w 4–5 krokach. Rozsądny wydaje się więc ustalenie maksymalnej liczby kroków na 7–8.

Drugim z kryteriów stopu jest *kryterium residualne*. Sprawdzamy tu nierówność

$$|f(x_k)| \leq \varepsilon,$$

gdzie $\varepsilon > 0$ jest małą liczbą. Jeśli warunek jest spełniony, to uznajemy x_k za rozwiązanie i przerywamy obliczenia.

Kolejne kryterium, *przyrostowe*, powoduje zakończenie obliczeń, gdy zostanie spełniony warunek

$$|x_{k+1} - x_k| \leq \delta,$$

dla pewnej małej liczby $\delta > 0$.

Oprócz wymienionych kryteriów czasami należy przerwać generowanie ciągu ze względu na to, że nie jest możliwe zdefiniowanie następnego przybliżenia. Dzieje się tak np. w metodzie stycznych, gdy pochodna funkcji w punkcie x_k jest równa zeru lub bliska零. Odpowiednia styczna nie przecina wówczas osi OX, albo przecina w miejscu tak odległym, że nie ma sensu dalszego stosowania metody.

32.13 Uwagi o metodach zbieżnych globalnie

Wszystkie twierdzenia o zbieżności omawianych metod interpolacyjnych zakładały znajomość dostatecznie dobrego przybliżenia początkowego. Otrzymaliśmy zatem lokalną zbieżność generowanych ciągów. Nie jest to przypadkowe, gdyż można udowodnić, że dla klasy funkcji regularnych (czyli posiadających pewną liczbę ciągłych pochodnych) i mających pojedyncze miejsca zerowe, nie istnieje zbieżna globalnie metoda, która do wyznaczenia kolejnego przybliżenia wykorzystuje jedynie wartości funkcji i jej pochodnych w punktach poprzednich przybliżeń.

Poszukiwania metod zbieżnych globalnie mogą iść w kierunku zmniejszenia klasy zadań lub wzbogacenia informacji niezbędnej do wyznaczenia kolejnego przybliżenia.

Przykładem klasy zadań, dla której istnieje zbieżna globalnie metoda jest klasa funkcji ciągłych f określonych na odcinku $[a, b]$ takich, że $f(a)$ oraz $f(b)$ mają przeciwnie znaki. Metoda zbieżna globalnie jest tu np. metoda bisekcji. Metoda ta jest zbieżna jedynie liniowo. Stosuje się ją zatem do otrzymania dobrego przybliżenia początkowego dla metod lokalnie, ale szybciej zbieżnych.

33 Wyznaczanie zer wielomianów

33.1 Uwarunkowanie zadania wyznaczania zer wielomianów

Rozważmy zadanie wyznaczenia zer $\alpha_1, \dots, \alpha_n$ wielomianu w danego w postaci naturalnej

$$w(x) = a_0 + a_1x + \dots + a_nx^n, \quad a_n \neq 0.$$

Przyjmijmy za dane współczynniki wielomianu a_i , liczby rzeczywiste.

Można udowodnić, że zera o większej krotności są bardziej wrażliwe na zmiany współczynników a_i wielomianu niż zera o krotnościach mniejszych. Jednak w każdym przypadku zdają się zadania bardzo źle uwarunkowane.

Przykład (Wilkinson) Zerami wielomianu:

$$\begin{aligned} w(x) &= a_{20}x^{20} + \dots + a_1x + a_0 \\ &= (x-1)(x-2)(x-3)\dots(x-19)(x-20) \end{aligned}$$

są liczby $1, 2, \dots, 20$. Jeśli zaburzymy tylko jeden ze współczynników: zamiast $a_{19} = 210$ weźmiemy $a_{19}' = a_{19}(1-\varepsilon)$ przy $\varepsilon = \frac{1}{2^{23} \cdot 210}$, to nowy wielomian ma już pierwiastki zespolone i najbliższymi pierwiastkami 15 wielomianu $w(x)$ są $13.992358137 \pm 2.518830070i$. Błąd wzgledny wyniku jest tu większy niż 0.05, choć błąd wzgledny danych wynosi jedynie $\frac{1}{2^{23} \cdot 210} < 0.06 \cdot 10^{-8}$.

Pokreślmy, że uwarunkowanie zależy od tego, jakie wielkości uznamy za dane. Zadanie obliczania zer wielomianu zapisanego w różnych bazach bądź postaciach może być różnie uwarunkowane.

33.2 Lokalizacja zer wielomianu

33.2.1 Liczba pierwiastków rzeczywistych

Ponieważ większość metod wymaga dobrego przybliżenia początkowego przydatne są twierdzenia umożliwiające w sposób przybliżony określenie lokalizacji zer. W przypadku wielomianów takie twierdzenia istnieją.

Przypomnę, że nawet jeśli wielomian ma wszystkie współczynniki rzeczywiste, to może mieć pierwiastki zespolone, np. $w(x) = x^2 + 1$. Pierwsza grupa twierdzeń służy do określenia liczby zer rzeczywistych wielomianu. Zero m -krotne liczymy m razy.

Twierdzenie 6 (Fouriera).

Niech w będzie wielomianem stopnia n , a przedział (a, b) będzie taki, że $w(a), w(b) \neq 0$. Dopuszczamy tu przypadek przedziałów nieograniczonych i wtedy $w(a)$ i $w(b)$ są rozumiane jako odpowiednie granice.

Określmy wielkość $M(x)$ jako liczbę zmian znaku w ciągu^a

$$w(x), w'(x), w''(x), \dots, w^{(n)}(x).$$

Wtedy liczba zer wielomianu w w przedziale (a, b) wynosi $M(a) - M(b)$ lub jest od tej liczby mniejsza o liczbę parzystą.

^aJeśli w ciągu występują zera, to je pomijamy.

Przykład Niech $w(x) = x^3 - x^2 - 4x + 3$. Wyznaczamy pochodne

$$\begin{aligned} w'(x) &= 3x^2 - 2x - 4, \\ w''(x) &= 6x - 2, \\ w'''(x) &= 6. \end{aligned}$$

Obliczmy teraz $M(0)$, $M(1)$, $M(3)$, $M(\infty)$, $M(-\infty)$. Utwórzmy w tym celu poniższą tabelę.

	$x = -\infty$	$x = \infty$	$x = 0$	$x = 1$	$x = 3$
$w(x)$	–	+	+	–	+
$w'(x)$	+	+	–	–	+
$w''(x)$	–	+	–	+	+
$w'''(x)$	+	+	+	+	+
$M(x)$	3	0	2	1	0

- Ponieważ $M(-\infty) - M(\infty) = 3$, to w ma w przedziale $(-\infty, \infty)$ jeden lub trzy pierwiastki rzeczywiste. To wiadomo również bez twierdzenia.
- Ponieważ $M(-\infty) - M(0) = 1$, to w ma jeden pierwiastek ujemny.
- Ponieważ $M(0) - M(1) = 1$, to w ma jeden pierwiastek w przedziale $(0, 1)$.
- Ponieważ $M(1) - M(3) = 1$, to w ma jeden pierwiastek w przedziale $(1, 3)$.

Z twierdzenia Fouriera możemy określić więc, że w ma trzy pierwiastki: jeden ujemny, jeden w przedziale $(0, 1)$ oraz jeden w przedziale $(1, 3)$.

Inną metodę określenia liczby zer rzeczywistych opisuje poniższe twierdzenie Laguerre'a.

Twierdzenie 7 (Laguerre'a).

Niech w będzie wielomianem stopnia n , a przedział (a, b) będzie taki, że $w(a), w(b) \neq 0$. Dopuszczamy tu przypadek przedziałów nieograniczonych i wtedy $w(a)$ i $w(b)$ są rozumiane jako odpowiednie granice.

Dla wielomianu $w(x) = a_0 + a_1x + \dots + a_nx^n$ i liczby x utworzymy ciąg $w_k(x)$ następująco:

$$\begin{aligned} w_0(x) &= a_n, \\ w_1(x) &= a_nx + a_{n-1}, \\ w_2(x) &= a_nx^2 + a_{n-1}x + a_{n-2}, \\ &\vdots \\ w_n(x) &= w(x). \end{aligned}$$

Niech $L(x)$ oznacza liczbę zmian znaku w powyższym ciągu.^a Wtedy liczba zer wielomianu w w przedziale (a, b) wynosi $L(a) - L(b)$ lub jest od tej liczby mniejsza o liczbę parzystą.

^aJeśli w ciągu występują zera, to je pomijamy.

Powyzsze twierdzenie zastosowane dla przedziału $(0, \infty)$ daje regułę Kartezjusza.

Wniosek 1 (reguła Kartezjusza).

Liczba zer dodatnich wielomianu $w(x) = a_0 + a_1x + \dots + a_nx^n$ jest równa liczbie zmian znaków w ciągu współczynników $a_n, a_{n-1}, \dots, a_1, a_0$ lub jest od tej liczby mniejsza o liczbę parzystą.

Z powyższej reguły można też skorzystać w celu znalezienia liczby ujemnych zer wielomianu $w(x)$. Należy wówczas zastosować ją do wielomianu $w(-x)$.

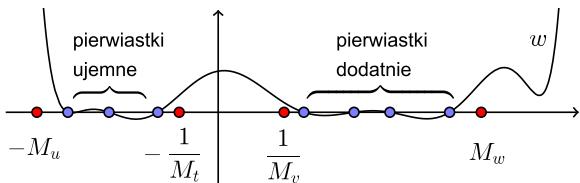
Przykład Wielomian $w(x) = x^3 - x^2 - 4x + 3$ ma dwa lub zero pierwiastków dodatnich oraz jeden pierwiastek ujemny.

Znane jest też twierdzenie Sturma pozwalające dokładnie wyznaczyć liczbę zer wielomianu w pewnym przedziale. Wymaga ono jednak bardziej uciążliwych obliczeń.

33.2.2 Lokalizacja zer rzeczywistych

Aby wyznaczyć przedział, w którym mieścią się wszystkie pierwiastki rzeczywiste wielomianu w wystarczy umieć znaleźć ograniczenie górne M_w pierwiastków dodatnich. Za uważmy bowiem, że ograniczenie

1. dolne ujemnych pierwiastków wielomianu $w(x)$ wynosi $-M_u$, gdzie M_u jest ograniczeniem górnym dodatnich pierwiastków wielomianu $u(x) = w(-x)$,
2. dolne dodatnich pierwiastków wielomianu $w(x)$ wynosi $\frac{1}{M_v}$, gdzie M_v jest ograniczeniem górnym dodatnich pierwiastków wielomianu $v(x) = x^n w(\frac{1}{x})$ (można też wziąć $v(x) = (-x)^n w(\frac{1}{x})$),
3. górne ujemnych pierwiastków wielomianu $w(x)$ wynosi $-\frac{1}{M_t}$, gdzie M_t jest ograniczeniem górnym dodatnich pierwiastków wielomianu $t(x) = (-x)^n w(-\frac{1}{x})$ (można też wziąć $t(x) = x^n w(-\frac{1}{x})$).



Udowodnimy, że rzeczywiście tak jest.

1. Niech

$$0 < x_0, x_1, \dots, x_k \leq M_u$$

będą wszystkimi dodatnimi miejscami zerowymi wielomianu $u(x) = w(-x)$. Wtedy dla $i = 0, 1, \dots, k$

$$\begin{aligned} u(x_i) = 0 &\iff \\ w(-x_i) = 0. & \end{aligned}$$

Liczby

$$0 > -x_0, -x_1, \dots, -x_k \geq -M_u$$

są wszystkimi ujemnymi miejscami zerowymi wielomianu w . Liczba $-M_u$ jest szukanym ograniczeniem dolnym pierwiastków ujemnych wielomianu w .

2. Niech

$$0 < x_0, x_1, \dots, x_k \leq M_v$$

będą wszystkimi dodatnimi miejscami zerowymi wielomianu $v(x) = x^n w(\frac{1}{x})$. Wtedy dla $i = 0, 1, \dots, k$

$$\begin{aligned} v(x_i) = 0 &\iff \\ x_i^n w\left(\frac{1}{x_i}\right) = 0 &\iff \\ w\left(\frac{1}{x_i}\right) = 0. & \end{aligned}$$

Liczby

$$\frac{1}{x_0}, \frac{1}{x_1}, \dots, \frac{1}{x_k} \geq \frac{1}{M_v} > 0$$

są wszystkimi dodatnimi miejscami zerowymi wielomianu w . Liczba $\frac{1}{M_v}$ jest szukanym ograniczeniem dolnym pierwiastków dodatnich wielomianu w . Analogicznie dowód przeprowadza się dla drugiej postaci funkcji v .

3. Niech

$$0 < x_0, x_1, \dots, x_k \leq M_t$$

będą wszystkimi dodatnimi miejscami zerowymi wielomianu $t(x) = (-x)^n w(-\frac{1}{x})$. Wtedy dla $i = 0, 1, \dots, k$

$$\begin{aligned} t(x_i) = 0 &\iff \\ (-x_i)^n w\left(-\frac{1}{x_i}\right) = 0 &\iff \\ w\left(-\frac{1}{x_i}\right) = 0. & \end{aligned}$$

Liczby

$$-\frac{1}{x_0}, -\frac{1}{x_1}, \dots, -\frac{1}{x_k} \leq -\frac{1}{M_t} < 0$$

są wszystkimi ujemnymi miejscami zerowymi wielomianu w . Liczba $-\frac{1}{M_t}$ jest szukanym ograniczeniem górnym pierwiastków ujemnych wielomianu w . Analogicznie dowód przeprowadza się dla drugiej postaci funkcji t .

Wobec tego dalej zajmiemy się jedynie zadaniem wyznaczania ograniczenia górnego dodatnich zer wielomianu.

Twierdzenie 8 (Lagrange'a).

Niech $w(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$ będzie wielomianem stopnia n . Niech k będzie takie, że

$$a_i \geq 0 \quad \text{dla } i = n-1, \dots, k+1, \quad \text{oraz } a_k < 0.$$

Ponadto niech A oznacza maksimum modulu ujemnych współczynników wielomianu. Wtedy wszystkie dodatnie pierwiastki wielomianu są mniejsze niż

$$M_w = 1 + \sqrt[n-k]{\frac{A}{|a_n|}}.$$

Jeżeli wszystkie współczynniki wielomianu są nieujemne, to nie ma on zer dodatnich.

Twierdzenie nie gwarantuje istnienia zer rzeczywistych. Określa jedynie przedziały, gdzie ich na pewno nie ma.

Przykład Z twierdzenia Lagrange'a wyznaczmy przedziały zawierające wszystkie zera rzeczywiste wielomianu

$$w(x) = 2x^9 + x^7 - x^5 + 6x^3 - 12x^2 + 9.$$

1. Liczba k z twierdzenia wynosi 5, zaś $A = 12$. Zatem pierwiastki dodatnie są nie większe niż

$$M_w = 1 + \sqrt[9-5]{6} \approx 2.565.$$

2. Niech $u(x) = w(-x)$, czyli

$$u(x) = -2x^9 - x^7 + x^5 - 6x^3 - 12x^2 + 9.$$

Teraz $k = 7$, $A = 12$. Czyli ograniczenie dolne ujemnych pierwiastków w wynosi

$$-M_u = -\left(1 + \sqrt{\frac{12}{2}}\right) \approx -3.449.$$

3. Rozpatrzmy wielomian $v(x) = x^9 w(\frac{1}{x})$. Zatem

$$v(x) = 9x^9 - 12x^7 + 6x^6 - x^4 + x^2 + 2.$$

Znów $k = 7$ oraz $A = 12$. Czyli ograniczenie dolne dodatnich pierwiastków w wynosi

$$\frac{1}{M_v} = \frac{1}{1 + \sqrt{\frac{12}{9}}} \approx \frac{1}{2.155} \approx 0.464.$$

4. Przyjmijmy $t(x) = x^9 w(-\frac{1}{x}) = -v(-x)$. Zatem

$$t(x) = 9x^9 - 12x^7 - 6x^6 + x^4 - x^2 - 2.$$

Znów $k = 7$ oraz $A = 12$. Czyli ograniczenie górne ujemnych pierwiastków w wynosi

$$-\frac{1}{M_t} = -\frac{1}{1 + \sqrt{\frac{12}{9}}} \approx -\frac{1}{2.155} \approx -0.464.$$

Podsumowując wszystkie przypadki otrzymujemy, że pierwiastki rzeczywiste w , o ile istnieją, leżą w zbiorze $(-3.449, -0.464) \cup (0.464, 2.565)$.

33.2.3 Lokalizacja zer zespolonych

Istnieje szereg twierdzeń pozwalających oszacować moduł zer zespolonych zer wielomianów lub stwierdzających, w których ćwiartkach płaszczyzny mogą się znajdować. Oto niektóre z nich.

Twierdzenie 9. Niech

$$w(z) = z^n + a_{n-1}z^{n-1} + a_{n-2}z^{n-2} + \dots + a_0,$$

gdzie $a_i \in \mathbb{C}$. Wybierzmy dowolną liczbę dodatnią β . Wtedy wszystkie pierwiastki wielomianu w (rzeczywiste i zespolone) są do modułu nie większe niż

$$M = \max \left\{ \frac{1}{\beta}, \sum_{k=0}^{n-1} |a_k| \beta^{n-k-1} \right\}.$$

Przykład Niech $w(z) = z^4 + 2z^3 + 3z^2 + 5z + 1$. Zatem oszacowanie modułu pierwiastków wynosi

$$M = \max \left\{ \frac{1}{\beta}, \beta^3 + 5\beta^2 + 3\beta + 2 \right\}.$$

Ponieważ β jest dowolne to można przyjąć $\beta = 0.3$. Wtedy uzyskujemy

$$M = 3.375.$$

Twierdzenie 10. Niech

$$w(z) = a_n z^n + a_{n-1} z^{n-1} + \dots + a_0,$$

gdzie $a_i \in \mathbb{C}$ i $a_n \neq 0$. Wtedy wszystkie pierwiastki wielomianu w (rzeczywiste i zespolone) są do modułu nie większe niż

$$M = 1 + \max_{0 \leq k \leq n} \frac{|a_k|}{|a_n|}.$$

Przykład Niech $w(z) = z^4 + 2z^3 + 3z^2 + 5z + 1$. Zatem oszacowanie modułu pierwiastków wynosi

$$M = 1 + \max \{2, 3, 5, 1\} = 6.$$

Następne twierdzenie dotyczy wielomianów o dodatnich współczynnikach rzeczywistych.

Twierdzenie 11. Niech

$$w(z) = z^n + a_{n-1} z^{n-1} + \dots + a_0,$$

gdzie $a_i \in \mathbb{R}$ i $a_i > 0$. Wtedy wszystkie pierwiastki wielomianu w (rzeczywiste i zespolone) są do modułu nie mniejsze niż

$$m = \min \left\{ \frac{a_{n-1}}{a_n}, \frac{a_{n-2}}{a_{n-1}}, \dots, \frac{a_0}{a_1} \right\},$$

oraz nie większe niż

$$M = \min \left\{ \frac{a_{n-1}}{a_n}, \frac{a_{n-2}}{a_{n-1}}, \dots, \frac{a_0}{a_1} \right\}.$$

Przykład Niech $w(z) = z^4 + 2z^3 + 3z^2 + 5z + 1$. Zatem oszacowanie dolne modułu pierwiastków wynosi

$$m = \min \left\{ \frac{2}{1}, \frac{3}{2}, \frac{5}{3}, \frac{1}{5} \right\} = \frac{1}{5},$$

a górne wynosi

$$M = \max \left\{ \frac{2}{1}, \frac{3}{2}, \frac{5}{3}, \frac{1}{5} \right\} = 2.$$

33.3 Metody przybliżonego wyznaczania zer wielomianów

Do wyznaczania zer można zastosować znane metody — np. metodę stycznych lub siecznych. Obliczenia wartości wielomianu lub jego pochodnej w punktach x_k należy przeprowadzić stosując schemat Hornera.

Szczególnie dobrą okazuje się metoda Mullera. W klasie zadań o zerze prostym metoda ta ma wykładownik zbieżności równy 3. Konstruuje ona ciąg x_k zgodnie z regułą:

$$x_{k+1} = x_k - \frac{2w(x_k)}{f'(x_k) + s_k \sqrt{(f'(x_k))^2 - 2f(x_k)f''(x_k)}},$$

gdzie $s_k = \text{sign}(f'(x_k))$.

Jeszcze wyżej ocenianą metodą jest *metoda Laguerre'a*. Jest ona zdefiniowana wzorem

$$x_{k+1} = x_k - \frac{nf(x_k)}{f'(x_k) + s_k \sqrt{H(x_k)}},$$

gdzie jak poprzednio $s_k = \text{sign}(f'(x_k))$, n jest stopniem wielomianu, a

$$H(x) = [(n-1)f'(x)]^2 - n(n-1)f(x)f''(x).$$

Udowodniono, że w przypadku wielomianów o zerach rzeczywistych metoda ta jest zbieżna globalnie. Przy czym jeśli zerami wielomianu są

$$\alpha_1 \leq \dots \leq \alpha_n,$$

to jeśli punkt startowy x_0 znajduje się w przedziale (α_k, α_{k-1}) to ciąg x_k zbiega do jednego z krańców przedziału. Jeśli zaś $x_0 > \alpha_n$ to x_k zbiega do α_n . Analogicznie, dla $x_0 < \alpha_1$ ciąg x_k zbiega do α_1 .

Obie powyższe metody mogą być użyte również do znajdowania zer zespolonych — wtedy są zbieżne lokalnie.

33.4 Deflacja

W praktyce często musimy wyznaczać nie jedno zero wielomianu, ale kilka lub wszystkie (tzn. szukamy rozkładu wielomianu na czynniki liniowe). Wydaje się celowe, aby po obliczeniu każdego zera obniżyć stopnieć wielomianu dzieląc go przez odpowiedni czynnik liniowy (bo jeśli α jest zerem wielomianu w , to $w(x) = (x - \alpha)v(x)$, gdzie v jest wielomianem stopnia o jeden mniejszym niż w). Nazywamy to *deflacją*. Kolejne zera wyznaczamy wówczas z równań wielomianowych coraz to niższych stopni. Jest to oczywiście tańsze niż obliczanie tych zer zawsze z oryginalnego równania. Ponadto każdorazowo likwidujemy obliczone zera i nie musimy się obawiać, że wyznaczmy je powtórnie.

O tym, czy otrzymany w procesie deflacyjnym iloraz może być zaakceptowany ze względów numerycznych decyduje jakość obliczonego zera, które służy do deflacji oraz dokładność samego procesu deflacyjnego.

Jeśli zastosujemy do procesu deflacyjnego algorytm Hornera, to proces deflacji może być niestabilny — daje on dobre wyniki jedynie jeśli zera wyznaczane są w kolejności wzrostających modułów. W 1971 r. opracowano numerycznie stabilny algorytm deflacyjny. Poniżej przedstawiam jego zarys.

Jeśli α jest dokładnym zera wielomianu w oraz

$$w(x) = (x - \alpha)v(x),$$

to współczynniki w_i $i = 1, \dots, n$ wielomianu

$$v(x) = w_1 + w_2x + \dots + w_nx^{n-1}$$

możemy wyznaczyć algorytmem Hornera:

$$w_n = a_n,$$

$$w_i = a_i + w_{i+1}\alpha, \quad i = n-1, n-2, \dots, 0.$$

Wtedy $w_0 = w(\alpha) = 0$. Możemy to wykorzystać, o ile $\alpha \neq 0$.

Zauważmy mianowicie, że

$$w_0 = 0,$$

$$w_{i+1} = (w_i - a_i)/\alpha, \quad i = 0, 1, \dots, n-1.$$

Algorytm zrealizowany wg powyższego zapisu nazywany jest *odwrotnym algorytmem Hornera*.

Okazuje się, że dopiero „sklejenie” algorytmu Hornera i odwrotnego algorytmu Hornera w odpowiednim miejscu daje dobre rezultaty. Należy współczynniki $w_n, w_{n-1}, \dots, w_{k+1}$ obliczyć algorytmem Hornera, zaś w_1, w_2, \dots, w_k odwrotnym algorytmem Hornera. Liczba k jest ścisłe określona i można ją obliczyć.

34 Układy równań nieliniowych

34.1 Sformułowanie zadania

Dany jest układ równań

$$\begin{cases} f_1(x_1, x_2, \dots, x_n) = 0 \\ f_2(x_1, x_2, \dots, x_n) = 0 \\ \dots \\ f_n(x_1, x_2, \dots, x_n) = 0, \end{cases}$$

gdzie przynajmniej jedna z funkcji f_1, f_2, \dots, f_n nie jest liniowa. Wszystkie te funkcje muszą mieć wspólną dziedzinę. Oznaczmy ją przez D . Zakładamy, że $D \in \mathbb{R}^n$ jest zbiorem wypukłym. Przeciwczdziedziną tych funkcji jest zbiór \mathbb{R} .

Układ taki można zapisać również w postaci:

$$\begin{cases} f_1(\vec{x}) = 0 \\ f_2(\vec{x}) = 0 \\ \dots \\ f_n(\vec{x}) = 0, \end{cases}$$

gdzie $\vec{x} \in \mathbb{R}^n$. Czasami stosuje się jeszcze bardziej zwięzły zapis:

$$\vec{f}(\vec{x}) = \vec{0},$$

gdzie

$$f(\vec{x}) = \begin{bmatrix} f_1(\vec{x}) \\ f_2(\vec{x}) \\ \vdots \\ f_n(\vec{x}) \end{bmatrix}.$$

W dalszym ciągu przez wektor

$$\vec{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix}.$$

będę oznaczać rozwiązanie tego układu.

34.2 Macierz Jacobiego

Opisana poniżej metoda Newtona wymaga umiejętności obliczenia pochodnej funkcji \vec{f} dla pewnego argumentu \vec{x} . Prawież $\vec{f} : \mathbb{R}^n \rightarrow \mathbb{R}^n$, to pochodna jest macierzą. Macierz ta nazywa się *macierzą Jacobiego* funkcji \vec{f} dla argumentu \vec{x} . Będę ją oznaczać przez $\vec{f}'(\vec{x})$. Aby wyznaczyć elementy tej macierzy, trzeba policzyć pochodne cząstkowe funkcji:

$$\vec{f}'(\vec{x}) = \begin{bmatrix} \frac{\partial f_1(\vec{x})}{\partial x_1} & \frac{\partial f_1(\vec{x})}{\partial x_2} & \cdots & \frac{\partial f_1(\vec{x})}{\partial x_n} \\ \frac{\partial f_2(\vec{x})}{\partial x_1} & \frac{\partial f_2(\vec{x})}{\partial x_2} & \cdots & \frac{\partial f_2(\vec{x})}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n(\vec{x})}{\partial x_1} & \frac{\partial f_n(\vec{x})}{\partial x_2} & \cdots & \frac{\partial f_n(\vec{x})}{\partial x_n} \end{bmatrix}.$$

Przypominam, że pochodną cząstkową $\frac{\partial f_i(\vec{x})}{\partial x_j}$ obliczamy traktując wszystkie zmienne poza x_j tak, jakby były stałymi, czyli na chwilę traktujemy funkcję f_i jako funkcję jednej zmiennej x_j i względem tej zmiennej liczymy pochodną.

Przykład Rozważmy funkcję

$$\begin{aligned} \vec{f} : \mathbb{R}^2 &\rightarrow \mathbb{R}^2 \\ \vec{f}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) &= \begin{bmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{bmatrix}, \end{aligned}$$

gdzie

$$\begin{aligned} f_1(x_1, x_2) &= -2x_1^2 + 3x_1x_2 + 4\sin(x_2) - 6, \\ f_2(x_1, x_2) &= 3x_1^2 - 2x_1x_2^2 + 3\cos(x_1) + 4. \end{aligned}$$

Aby obliczyć pochodną cząstkową $\frac{\partial f_1(\vec{x})}{\partial x_1}$ traktujemy na chwilę funkcję f_1 tak, jakby była funkcją tylko zmiennej x_1 , a zmienną x_2 traktujemy jak stałą i liczymy „zwykłą” pochodną.

$$\begin{aligned} f_1(\textcolor{red}{x}_1, x_2) &= -2\textcolor{red}{x}_1^2 + 3\textcolor{red}{x}_1x_2 + 4\sin(x_2) - 6, \\ \frac{\partial f_1(\vec{x})}{\partial \textcolor{red}{x}_1} &= -4\textcolor{red}{x}_1 + 3x_2. \end{aligned}$$

Podobnie, aby obliczyć pochodną cząstkową $\frac{\partial f_1(\vec{x})}{\partial x_2}$ traktujemy na chwilę funkcję f_1 tak, jakby była funkcją tylko zmiennej x_2 , a zmienną x_1 traktujemy jak stałą i liczymy „zwykłą” pochodną.

$$\begin{aligned} f_1(x_1, \textcolor{red}{x}_2) &= -2x_1^2 + 3x_1\textcolor{red}{x}_2 + 4\sin(\textcolor{red}{x}_2) - 6, \\ \frac{\partial f_1(\vec{x})}{\partial \textcolor{red}{x}_2} &= 3x_1 + 4\cos(\textcolor{red}{x}_2). \end{aligned}$$

Analogicznie obliczamy pochodną $\frac{\partial f_2(\vec{x})}{\partial x_1}$

$$\begin{aligned} f_2(\textcolor{red}{x}_1, x_2) &= 3\textcolor{red}{x}_1^2 - 2\textcolor{red}{x}_1x_2^2 + 3\cos(\textcolor{red}{x}_1) + 4, \\ \frac{\partial f_2(\vec{x})}{\partial \textcolor{red}{x}_1} &= 6\textcolor{red}{x}_1 - 2x_2^2 - 3\sin(\textcolor{red}{x}_1), \end{aligned}$$

oraz $\frac{\partial f_2(\vec{x})}{\partial x_2}$

$$\begin{aligned} f_2(x_1, \textcolor{red}{x}_2) &= 3x_1^2 - 2x_1\textcolor{red}{x}_2^2 + 3\cos(x_1) + 4, \\ \frac{\partial f_2(\vec{x})}{\partial \textcolor{red}{x}_2} &= -4x_1^2\textcolor{red}{x}_2. \end{aligned}$$

Macierzą Jacobiego funkcji \vec{f} dla argumentu \vec{x} jest zatem

$$\begin{aligned} \vec{f}'(\vec{x}) &= \begin{bmatrix} \frac{\partial f_1(\vec{x})}{\partial x_1} & \frac{\partial f_1(\vec{x})}{\partial x_2} \\ \frac{\partial f_2(\vec{x})}{\partial x_1} & \frac{\partial f_2(\vec{x})}{\partial x_2} \end{bmatrix} \\ &= \begin{bmatrix} -4x_1 + 3x_2 & 3x_1 + 4\cos(x_2) \\ 6x_1 - 2x_2^2 - 3\sin(x_1) & -4x_1^2x_2 \end{bmatrix}. \end{aligned}$$

Jeśli wezmę konkretny argument, np. $\vec{x} = [0, \pi]^T$, to Macierzą Jacobiego funkcji \vec{f} dla tego argumentu wynosi

$$\begin{aligned} \vec{f}'\left(\begin{bmatrix} 0 \\ \pi \end{bmatrix}\right) &= \begin{bmatrix} -4 \cdot 0 + 3\pi & 3 \cdot 0 + 4\cos(\pi) \\ 6 \cdot 0 - 2\pi^2 - 3\sin(0) & -4 \cdot 0^2 \cdot \pi \end{bmatrix} \\ &= \begin{bmatrix} 3\pi & -4 \\ -2\pi^2 & 0 \end{bmatrix}. \end{aligned}$$

34.3 Metoda Newtona

Metoda ta polega na generowaniu ciągu wektorów $\vec{x}^{(0)}, \vec{x}^{(1)}, \vec{x}^{(2)}, \vec{x}^{(3)} \dots$ w sposób rekurencyjny, tzn. wektor $\vec{x}^{(k+1)}$ wyznaczamy na podstawie funkcji \vec{f} i wektora $\vec{x}^{(k)}$. Jeśli wystartujemy dostatecznie blisko rozwiązania \vec{a} i funkcja \vec{f} posiada odpowiednie własności, to ciąg ten będzie do niego zbiegał.

Metoda Newtona definiuje wektor $\vec{x}^{(k+1)}$ następującym wzorem

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} - [\vec{f}'(\vec{x}^{(k)})]^{-1} \vec{f}(\vec{x}^{(k)}).$$

Macierz $[\vec{f}'(\vec{x}^{(k)})]^{-1}$ jest macierzą odwrotną do macierzy Jacobiego $\vec{f}'(\vec{x}^{(k)})$, czyli wartości pochodnej policzonej dla argumentu $\vec{x}^{(k)}$.

Aby metoda była poprawnie zdefiniowana należy założyć, że dla dowolnego k macierz $\vec{f}'(\vec{x}^{(k)})$ istnieje i jest nieosobliwa.

Realizacja komputerowa jednego kroku tej metody wygląda następująco.

1. Wyznacz wektor $\vec{f}(\vec{x}^{(k)})$.
2. Wyznacz macierz $\mathbf{A}^{(k)} = \vec{f}'(\vec{x}^{(k)})$.
3. Rozwiąż układ równań liniowych

$$\mathbf{A}^{(k)} \vec{x} = -\vec{f}(\vec{x}^{(k)}),$$

zapisz rozwiązanie w wektorze $\vec{z}^{(k)}$.

4. Oblicz $\vec{x}^{(k+1)} = \vec{x}^{(k)} + \vec{z}^{(k)}$.

Kryterium zakończenia algorytmu formułuje się na podstawie norm wektorów $\vec{x}^{(k)}, \vec{f}(\vec{x}^{(k)})$ i $\vec{z}^{(k)}$.

Podobnie jak dla równań nieliniowych, dla układów równań nieliniowych zdefiniuje się wykładnik zbieżności. Można udowodnić, że dla metody Newtona wynosi on 2.

34.4 Metoda siecznych

Metoda ta polega na generowaniu ciągu wektorów $\vec{x}^{(0)}, \vec{x}^{(1)}, \vec{x}^{(2)}, \vec{x}^{(3)} \dots$ w sposób rekurencyjny. Tym razem nie wystarczy pamiętać ostatni wektor, aby wyznaczyć

następny. Do wyznaczenia wektora $\vec{x}^{(k+1)}$ używamy bowiem $n+1$ wektorów $\vec{x}^{(k)}, \vec{x}^{(k-1)}, \vec{x}^{(k-2)}, \dots, \vec{x}^{(k-n)}$. Potrzebujemy zatem $n+1$ wektorów początkowych. Jeśli wystartujemy dostatecznie blisko rozwiązania \vec{a} i funkcja \vec{f} posiada odpowiednie własności, to ciąg ten będzie do niego zbiegał.

Metoda ta nie wymaga pochodnej funkcji \vec{f} . Używa natomiast w każdym kroku dwóch macierzy \mathbf{X}_k oraz \mathbf{F}_k , obie rozmiaru $n \times n$. Kolumnami macierzy \mathbf{X}_k są różnice sąsiednich wektorów $\vec{x}^{(k)}, \vec{x}^{(k-1)}, \vec{x}^{(k-2)}, \dots, \vec{x}^{(k-n+2)}, \vec{x}^{(k-n+1)}, \vec{x}^{(k-n)}$, dokładniej

$$\mathbf{X}_k = [\vec{x}^{(k-n+1)} - \vec{x}^{(k-n)} \quad \vec{x}^{(k-n+2)} - \vec{x}^{(k-n+1)} \quad \dots \quad \vec{x}^{(k)} - \vec{x}^{(k-1)}]$$

$$= \begin{bmatrix} x_1^{(k-n+1)} - x_1^{(k-n)} & x_1^{(k-n+2)} - x_1^{(k-n+1)} & \dots & x_1^{(k)} - x_1^{(k-1)} \\ x_2^{(k-n+1)} - x_2^{(k-n)} & x_2^{(k-n+2)} - x_2^{(k-n+1)} & \dots & x_2^{(k)} - x_2^{(k-1)} \\ \vdots & \vdots & & \vdots \\ x_n^{(k-n+1)} - x_n^{(k-n)} & x_n^{(k-n+2)} - x_n^{(k-n+1)} & \dots & x_n^{(k)} - x_n^{(k-1)} \end{bmatrix}.$$

Kolumnami macierzy \mathbf{F}_k są różnice sąsiednich wektorów $\vec{f}(\vec{x}^{(k)}), \vec{f}(\vec{x}^{(k-1)}), \vec{f}(\vec{x}^{(k-2)}), \dots, \vec{f}(\vec{x}^{(k-n+2)}), \vec{f}(\vec{x}^{(k-n+1)}), \vec{f}(\vec{x}^{(k-n)})$, dokładniej

$$\mathbf{F}_k = [\vec{f}(\vec{x}^{(k-n+1)}) - \vec{f}(\vec{x}^{(k-n)}) \quad \dots \quad \vec{f}(\vec{x}^{(k)}) - \vec{f}(\vec{x}^{(k-1)})] \text{ gdzie}$$

$$= \begin{bmatrix} f_1(\vec{x}^{(k-n+1)}) - f_1(\vec{x}^{(k-n)}) & \dots & f_1(\vec{x}^{(k)}) - f_1(\vec{x}^{(k-1)}) \\ f_2(\vec{x}^{(k-n+1)}) - f_2(\vec{x}^{(k-n)}) & \dots & f_2(\vec{x}^{(k)}) - f_2(\vec{x}^{(k-1)}) \\ \vdots & & \vdots \\ f_n(\vec{x}^{(k-n+1)}) - f_n(\vec{x}^{(k-n)}) & \dots & f_n(\vec{x}^{(k)}) - f_n(\vec{x}^{(k-1)}) \end{bmatrix}$$

Wektor $\vec{x}^{(k+1)}$ zdefiniowany jest wzorem

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} - \mathbf{X}_k \mathbf{F}_k^{-1} \vec{f}(\vec{x}^{(k)}).$$

Aby metoda była poprawnie zdefiniowana należy założyć, że dla dowolnego k macierze \mathbf{X}_k i \mathbf{F}_k są nieosobliwe.

Realizacja komputerowa jednego kroku tej metody wygląda następująco.

1. Wyznacz wektor $\vec{f}(\vec{x}^{(k)})$.
2. Wyznacz macierz \mathbf{X}_k .
3. Wyznacz macierz \mathbf{F}_k .
4. Rozwiąż układ równań liniowych

$$\mathbf{F}^{(k)} \vec{x} = -\vec{f}(\vec{x}^{(k)}),$$

zapisz rozwiązanie w wektorze $\vec{z}^{(k)}$.

5. Oblicz $\vec{x}^{(k+1)} = \vec{x}^{(k)} + \mathbf{X}_k \vec{z}^{(k)}$.

Można udowodnić, że dla metody siecznych wykładnik zbieżności zależy od funkcji \vec{f} i należy do przedziału $(1, 2)$.

34.5 Rozwiązywanie układów równań nieliniowych w Octave

Do rozwiązywania układów równań nieliniowych w Octave służy funkcja `fsolve`. Należy podać dwa argumenty — funkcję \vec{f} i wektor startowy. Trzeci argument nie jest obowiązkowy i służy do ustawienia opcji.

34.5.1 Implementacja funkcji \vec{f} w Octave

Aby móc podać argument pierwszy, czyli nazwę funkcji, najpierw trzeba tę funkcję napisać. Jeśli chcielibyśmy rozwiązać układ równań

$$\begin{cases} -2x^2 + 3xy + 4 \sin(y) = 6 \\ 3x^2 + 3 \cos(x) + 4 = 2xy^2, \end{cases}$$

to najpierw przekształcamy go do postaci, w której wszystkie prawe strony są równe 0.

$$\begin{cases} -2x^2 + 3xy + 4 \sin(y) - 6 = 0 \\ 3x^2 + 3 \cos(x) + 4 - 2xy^2 = 0. \end{cases}$$

Funkcja \vec{f} , która opisuje ten układ ma dwa argumenty i dwa wyniki, czyli argumentem jest wektor o dwóch elementach oraz wynikiem jest wektor o dwóch elementach,

$$\begin{aligned} \vec{f} &: \mathbb{R}^2 \rightarrow \mathbb{R}^2 \\ \vec{f}\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) &= \begin{bmatrix} f_1(x, y) \\ f_2(x, y) \end{bmatrix}, \end{aligned}$$

$$\begin{aligned} f_1(x, y) &= -2x^2 + 3xy + 4 \sin(y) - 6, \\ f_2(x, y) &= 3x^2 + 3 \cos(x) + 4. \end{aligned}$$

Ponieważ w Octave będziemy używać wektorów do zapisania argumentów i wyników, to lepiej zamiast nazw zmiennych x, y używać x_1, x_2

$$\begin{aligned} \vec{f} &: \mathbb{R}^2 \rightarrow \mathbb{R}^2 \\ \vec{f}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) &= \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \end{aligned}$$

gdzie

$$\begin{aligned} y_1 &= f_1(x_1, x_2) = -2x_1^2 + 3x_1x_2 + 4 \sin(x_2) - 6, \\ y_2 &= f_2(x_1, x_2) = 3x_1^2 - 2x_1x_2^2 + 3 \cos(x_1) + 4. \end{aligned}$$

Realizacja funkcji \vec{f} w Octave wygląda następująco.

```
function y = f(x)
y= zeros(2,1);
y(1)= -2*x(1)^2 + 3*x(1)*x(2) + 4*sin(x(2)) - 6;
y(2)= 3*x(1)^2 - 2*x(1)*x(2)^2 + 3*cos(x(1)) + 4;
endfunction
```

Polecenie `zeros(2,1)` tworzy macierz o 2 wierszach i jednej kolumnie (czyli wektor o dwóch współrzędnych) wypełnioną zerami.

34.5.2 Rozwiązywanie układu równań nieliniowych bez podawania macierzy Jacobiego

Aby rozwiązać układ równań nieliniowych wystarczy teraz napisać poniższe polecenia, metoda wymaga podania wektora startowego.

```
x0=[1;2];
[x, fvec, info, output, fjac] = fsolve ("f",x0)
```

Funkcja `fsolve` zwraca następujące wartości

1. `x` — wynik, który jest wektorem, jego współrzędne ozna- czają wartości kolejnych zmiennych,
2. `fvec` — wartość funkcji \vec{f} dla obliczonego wyniku,
3. `info` — informacja o zbieżności procesu:
 - 1 — ciąg przybliżeń jest zbieżny do uzyska- nego rozwiązania uzyskanego z zadaną dokładno- ścią, dokładność jest zapisana w opcjach "TolX", "TolFun", które można zmieniać,
 - 2 — osiągnięto pewien warunek stopu regulowany opcją "TolX",
 - 3 — osiągnięto pewien warunek stopu regulowany opcją "TolFun",
 - 0 — został przekroczony limit iteracji,
 - -3 — metoda zbytnio zawężała obszar poszukiwań nowego przybliżenia (stał się on zbyt mały).
4. `output` — struktura zawierająca informacje o liczbie ite- racji oraz obliczeń wartości funkcji,
5. `fjac` — ostatnio obliczona macierz Jacobiego.

Jeśli chcemy zobaczyć wyniki z większą dokładnością należy przed wywołaniem `fsolve` wykonać

```
format long;
```

34.5.3 Rozwiązywanie układu równań nieliniowych z podaniem macierzy Jacobiego

Jeśli umiemy obliczać macierz Jacobiego, to przekazując ją do funkcji `fsolve` możemy zmniejszyć liczbę iteracji lub obliczeń wartości funkcji \vec{f} . Przypomnijmy, że działanie funkcji `fsolve`, bowiem gdy macierz Jacobiego nie jest znana, to zastępowana jest przez jej przybliżenie.

Aby przekazać macierz Jacobiego do funkcji `fsolve` trzeba po pierwsze zmienić implementację funkcji \vec{f} , a po drugie ustawić odpowiednią opcję w `fsolve`.

Macierz Jacobiego dla rozpatrywanego układu została już wcześniej obliczona, jest nią

$$\vec{f}'(\vec{x}) = \begin{bmatrix} -4x_1 + 3x_2 & 3x_1 + 4\cos(x_2) \\ 6x_1 - 2x_2^2 - 3\sin(x_1) & -4x_1^2 x_2 \end{bmatrix}.$$

Implementacja funkcji \vec{f}' powinna obecnie nie tylko obliczać wartość funkcji \vec{f}' dla zadanego wektora \vec{x} , ale również macierz Jacobiego funkcji \vec{f}' dla argumentu \vec{x} , tak jak poniżej.

```
function [y, jac] = f (x)
y= zeros (2, 1);
y(1)= -2*x(1)^2 + 3*x(1)*x(2) + 4*sin(x(2)) - 6;
y(2)= 3*x(1)^2 - 2*x(1)*x(2)^2 + 3*cos(x(1)) + 4;
if (nargout == 2)
  jac = zeros (2, 2);
  jac(1,1) = 3*x(2) - 4*x(1);
```

```
    jac(1,2) = 4*cos(x(2)) + 3*x(1);
    jac(2,1) = -2*x(2)^2 - 3*sin(x(1)) + 6*x(1);
    jac(2,2) = -4*x(1)*x(2);
endif
endfunction
```

Funkcja zwraca dwa wyniki: `y` oraz `jac`. Instrukcja warunkowa `if (nargout == 2)` sprawdza, czy podczas wywołania `f` zostały zdefiniowane dwie zmienne na przechowanie wyników. Jeśli wywołanie `f` wygląda tak

```
[w,J]=f(x)
```

to `nargout` będzie miało wartość 2, jeśli zaś wywołamy ją na jeden z poniższych sposobów

```
w=f(x)
f(x)
```

to `nargout` będzie miało wartość 1. Sprawdzenie `if (nargout == 2)` pozwala skrócić działanie `f`, gdybyśmy nie potrzebowali dwóch wyników.

Po zmodyfikowaniu funkcji `f` ustawiamy opcje i wywołujemy `fsolve` z wektorem startowym `x0`.

```
op=optimset ("Jacobian", "on");
x0=[1;2];
[x, fvec, info, output, fjac] = fsolve ("f",x0,op)
```

34.5.4 Ustawianie dokładności i maksymalnej liczby iteracji

Funkcja `optimset` pozwala ustalić szereg opcji i parametrów, między innymi dokładność obliczonego wyniku oraz maksymalną liczbę iteracji. Można to zrobić jak poniżej.

```
op=optimset("TolFun",1e-9,"TolX",1e-9,"MaxIter",10)
```

Parametr "TolX" dotyczy dokładności argumentów, a "TolFun" dokładności wartości funkcji.

Gdy chcemy, by przy tych ustawieniach macierz Jacobiego była przekazana lub nie, ustawiamy dodatkową opcję.

```
op=optimset("Jacobian", "on",
            "TolFun",1e-9,"TolX",1e-9,"MaxIter",10)
op=optimset("Jacobian", "off",
            "TolFun",1e-9,"TolX",1e-9,"MaxIter",10)
```