Wydział Budowy Maszyn i Zarządzania
Politechniki Poznańskiej

Zmiennopozycyjna arytmetyka przedziałowa w algorytmach algebry liniowej

Mirosław Miedziarek

Praca magisterska napisana
pod kierunkiem
dr. inż. Karola Gajdy

Poznań 2007

Pracę dedykuję
moim najbliższym dziękując
im za nieustanną pomoc i wsparcie.

Spis treści

1 Wstęp

Z całą odpowiedzialnością można powiedzieć, że dzisiejszy świat jest całkowicie uzależniony od komputerów. *Cyfrowe maszyny liczące* występują praktycznie w każdej dziedzinie ludzkiego życia, które staje się powoli coraz bardziej od nich zależne. Wystarczy jednak zadać takiej *cyfrowej maszynie liczącej* prosty problem matematyczny do rozwiązania, a ta mimo swojej potężnej mocy obliczeniowej i zaawansowanych algorytmów numerycznych i tak zwróci niepoprawny wynik. Powodem takiego stanu rzeczy nie jest bynajmniej błąd w oprogramowaniu (co jednak często ma miejsce), tylko fakt, że komputery używają dyskretnej arytmetyki zmiennopozycyjnej. Znaczy to tyle, że potrafią dokładnie reprezentować tylko skończony zbiór liczb rzeczywistych, których, jak wiadomo, w samym przedziale od zera do jedności, jest nieskończenie wiele. Dla przykładu liczba 0.1 mająca nieskończone rozwinięcie w systemie dwójkowym (taki system używają rejestry i pamięć komputera) nie jest dokładnie reprezentowana w pamięci komputera, co powoduje, że wprowadzenie do komputera tej liczby powoduje zapamiętanie w pamięci całkiem innej wartości, nie wspominając o tym, że obliczenia na takiej wartości są błędne.

Podczas wojny w 1990 roku (ang. Gulf War)[1], Amerykanie prowadzili operację *Burza pustynna* (ang. Desert Storm), której główną częścią był system antyrakietowy relacji ziemia - powietrze o nazwie *Patriot* chroniący przed powietrznym atakiem Irackich pocisków klasy *SCUD* umiejscowiony w Dhahran w Arabii Saudyjskiej. Dnia 25 lutego system zawiódł, zginęło 28 amerykańskich żołnierzy i prawie stu zostało rannych.

Powodem tragedii był błąd obliczeń komputera systemu *Patriot*. Algorytm użyty do oszacowania następnego obszaru w przestrzeni, który ma być skanowany przez radar, wymagał aby prędkość pocisku oraz czas były wyrażone jako liczby rzeczywiste. Komputer posiadał jednak tylko 24 bitowe rejestry, a czas mierzony był w interwałach co dziesięć sekund, a następnie mnożony przez czynnik 0.1, który został obcięty w pamięci do 24 bitów. Błąd zaokrąglenia rósł tak szybko jak przyrastał czas pomiarów radaru, a niedokładność ostatecznych obliczeń rosła wprost proporcjonalnie do prędkości nadlatującego pocisku. W wyniku tego po około stu godzinach przewidywana przestrzeń, którą miał skanować radar oddalona była od faktycznego

[1]Informacje zaczerpnięte ze strony *http://www.cs.usyd.edu.au/~alum/patriot_bug.html*

położenia celu o ponad pół kilometra.

Oczywiście nie można powiedzieć na pewno, że lepsza arytmetyka zapobiegła by tej i innym podobnym katastrofom. Jakkolwiek chciało by się wierzyć w to, że komputery, od których tak wiele ludzkich istnień dzisiaj zależy, potrafiły wykonywać chociaż podstawowe operacje matematyczne poprawnie.

Jednym z rozwiązań problemu niedokładności obliczeniowej komputerów jest zastosowanie arytmetyki przedziałowej w implementacji podstawowych działań matematycznych. Arytmetyka przedziałowa nie zwiększa dokładności obliczanych wyników, jednak w każdym przypadku daje pewność zakresu dokładności, lub jej brak. Zakres dokładności znaczy tutaj ilość cyfr po przecinku (lub kropce dziesiętnej), które na pewno są dokładne.
W przypadku standardowych obliczeń na liczbach zmiennopozycyjnych, wynikiem jest pojedyncza liczba, o dokładności której często nie możemy być pewni. Natomiast obliczenia wykonane w arytmetyce przedziałowej dają w wyniku dwie liczby rzeczywiste, o których wiemy, że wyznaczają brzegi przedziału, w którym dokładny wynik obliczeń na pewno się znajduje. W najgorszym przypadku zatem, chociaż nie wiemy, gdzie leży rozwiązanie problemu, to jednak wiemy jak daleko od tego rozwiązania jest wynik naszych obliczeń.

2 Pojęcia podstawowe

W tym rozdziale wprowadzimy podstawowe pojęcia z zakresu algebry liniowej. Omówione zostaną wektory, macierze, działania na macierzach oraz ich własności, układy równań liniowych i metody ich rozwiązywania. Następnie przedstawiona zostanie teoria arytmetyki przedziałowej.

2.1 Wektory i macierze

Wprowadźmy pewne oznaczenia dla ujednolicenia zapisu w niniejszej pracy. Zbiór liczb rzeczywistych oznaczmy przez $\mathbb{R}$, natomiast liczby należące do tego zbioru małymi literami alfabetu, czyli $x, y, z, itd.$

Wektor

Wektorem rzeczywistym (inaczej elementem przestrzeni $\mathbb{R}^n$), nazywać będziemy każdy ciąg n liczb rzeczywistych zapisanych w postaci:

$$\vec{x} = [x_1, x_2, \ldots, x_n] \text{ gdzie } x_i \in \mathbb{R},\ i = 1 \ldots n.$$

Macierz

Macierzą $A = [a_{i,j}]$ stopnia $m \times n$ nazywamy tablicę prostokątną postaci:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,n} \\ a_{2,1} & a_{2,2} & \ldots & a_{2,n} \\ \ldots & & & \\ a_{m,1} & a_{m,2} & \ldots & a_{m,n} \end{bmatrix}.$$

gdzie $a_{i,j} \in \mathbb{R}$, dla $i = 1, \ldots, m$ oraz $j = 1, \ldots, n$ są elementami macierzy A. Macierz złożoną z samych zer nazywamy macierzą zerową.
Z każdej macierzy można wyciągnąć kolumnę lub wiersz w postaci wektora. Wiersz numer i macierzy A zapisujemy:

$$A_{i,\bullet} = [a_{i,1}, a_{i,2}, \ldots, a_{i,n}].$$

Z kolei kolumnę numer j wyciągniętą z macierzy A oznaczać będziemy następująco:

$$A_{\bullet,j} = \begin{bmatrix} a_{1,j} \\ a_{2,j} \\ \vdots \\ a_{m,j} \end{bmatrix}.$$

Działania na macierzach oraz postaci macierzy

Macierz nazywamy macierzą kwadratową stopnia n gdy ilość wierszy jest równa ilości kolumn macierzy, tzn. $m = n$.

Dwie macierze A i B są sobie równe jeśli mają takie same wymiary oraz takie same elementy. Jeśli macierz A jest wymiaru $m \times n$, a B jest wymiaru $p \times q$ to warunkiem równości macierzy $A = B$ jest:

1) $m = p$ oraz $n = q$,
2) $\forall\ i \in \{1, ..., m\}\ \ \forall\ j \in \{1, ..., n\}\ \ a_{i,j} = b_{i,j}$.

Pojęcie sumy i różnicy dwóch macierzy A i B dotyczy działania na elementach tych macierzy, to znaczy dodawania i odejmowania elementów o jednakowych indeksach. Przy czym spełnione musi być założenie 1) z powyższej definicji o równości wymiarów macierzy. Oznaczając macierz $C = A+B$ oraz $D = A - B$ mamy, że:

$$c_{i,j} = a_{i,j} + b_{i,j},$$

$$d_{i,j} = a_{i,j} - b_{i,j},$$

gdzie $i \in \{1, \ldots, m\}$ oraz $j \in \{1, \ldots, n\}$.

Macierz $C = [a_{i,j}]$ będąca iloczynem macierzy $A_{m\times n}$ oraz $B_{p\times q}$ jest zdefiniowana następująco:

$$c_{i,j} = a_{i,1}b_{1,j} + a_{i,2}b_{2,j} + \ldots + a_{i,n}b_{n,j}.$$

Zatem element $c_{i,j}$ macierzy $C = A \cdot B$ jest kombinacją liniową i-tego wiersza macierzy A oraz j-tej kolumny macierzy B. Warto jednocześnie zauważyć, że iloczyn macierzy istnieje wtedy i tylko wtedy gdy ilość kolumn macierzy A jest równa ilości wierszy macierzy B, tzn. $n = p$. Inaczej możemy zapisać:

$$c_{i,k} = \sum_{j=1}^{n} a_{i,j}b_{j,k},$$

$$\text{dla } i = 1, \ldots, m \text{ oraz } k = 1, \ldots, q.$$

Własności iloczynu macierzy:

1. łączność $(A \cdot B) \cdot C = A \cdot (B \cdot C)$ dla macierzy $A_{k\times m}, B_{m\times n}, C_{n\times p}$,

2. rozdzielność względem dodawania $(A + B) \cdot C = A \cdot C + B \cdot C$ dla macierzy $A_{m\times n}, B_{m\times n}, C_{n\times k}$.

Iloczyn macierzy zwykle nie jest przemienny, tzn. $A \cdot B \neq B \cdot A$.

Elementami diagonalnymi (diagonalą macierzy, przekątną główną) macierzy nazywamy elementy postaci:

$$a_{1,1}, a_{2,2}, \ldots, a_{l,l} \text{ gdzie } l = min(m, n).$$

Macierz nazywamy diagonalną, jeśli wszystkie jej elementy poza elementami diagonalnymi są równe zero, tzn.:

$$a_{i,j} = 0 \text{ dla } i \neq j.$$

Szczególną postacią macierzy diagonalnej jest kwadratowa macierz jednostkowa (lub jedynkowa) oznaczana zwykle przez I. Elementy na diagonali macierzy jednostkowej są wszystkie równe 1, tzn. macierz ta jest postaci:

$$I = \begin{bmatrix} 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 0 \\ 0 & 0 & \ldots & 0 & 1 \end{bmatrix}.$$

Macierz nazywamy górno-trójkątną (1), jeśli wszystkie jej elementy poniżej diagonali są równe zero oraz dolno-trójkątną (2) jeśli wszystkie jej elementy powyżej diagonali są równe zero:

$$A = \begin{bmatrix} a & a & \ldots & a & a \\ 0 & a & \ldots & a & a \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & a & a \\ 0 & 0 & \ldots & 0 & a \end{bmatrix}. \tag{1}$$

$$A = \begin{bmatrix} a & 0 & \dots & 0 & 0 \\ a & a & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a & a & \dots & a & 0 \\ a & a & \dots & a & a \end{bmatrix}. \tag{2}$$

Macierzą pasmową (taśmową) o szerokości $l = p + q + 1$ jeśli $a_{i,j} = 0$ dla $i > j + p$ oraz $j > i + q,\ p, q \geq 0$. Oznacza to że macierz A zawiera elementy niezerowe tylko w p dolnych przekątnych (tzn poniżej przekątnej głównej) oraz w q górnych przekątnych (powyżej przekątnej głównej) np.

$$A = \begin{bmatrix} a & a & 0 & 0 \\ a & a & a & 0 \\ 0 & a & a & a \\ 0 & 0 & a & a \end{bmatrix} ,B = \begin{bmatrix} b & b & b & 0 \\ 0 & b & b & b \\ 0 & 0 & b & b \\ 0 & 0 & 0 & b \end{bmatrix}.$$

Macierz transponowana A^T do macierzy A powstaje przez "odbicie" macierzy A względem przekątnej głównej. Formalnie zapiszemy to wyrażając elementy macierzy transponowanej następująco:

$$A^T_{i,j} = A_{j,i},$$

np.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix},\ A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}.$$

Zatem transpozycja macierzy jest zamianą wszystkich wierszy na kolumny oraz kolumn na wiersze. Transpozycja macierzy spełnia poniższe własności:

1. $(A^T)^T = A$,
2. $(A + B)^T = A^T + B^T$,
3. $(\lambda A)^T = \lambda A^T$,
4. $(AB)^T = B^T A^T$.

Z pojęciem transponowania związane jest pojęcie macierzy symetrycznej. Mówimy, że macierz A jest macierzą symetryczną jeśli $A^T = A$. Oznacza to dosłownie "symetryczność" elementów macierzy względem głównej przekątnej.

Rząd macierzy

Rzędem macierzy nazywamy maksymalny stopień kwadratowej nieosobliwej podmacierzy wyciągniętej z danej macierzy i oznaczamy $R(A)$. Dla macierzy w postaci schodkowej rząd macierzy jest równy ilości niezerowych wierszy.

Przekształcenia elementarne

Przekształceniami elementarnymi na danej macierzy $A = [a_{i,j}]_{m\times n}$ nazywamy następujące działania na wierszach lub kolumnach macierzy:

- T1 - polega na pomnożeniu wszystkich elementów wybranego wiersza lub kolumny przez liczbę $\alpha \neq 0$,
- T2 - polega na zamianie miejscami dwóch dowolnie wybranych wierszy lub kolumn,
- T3 - polega na dodaniu do wszystkich elementów wybranego wiersza lub kolumny odpowiadających im elementów innego wiersza lub kolumny pomnożonych przez liczbę $\alpha \neq 0$.

Wyznacznik

Dla każdej macierzy kwadratowej stopnia n $A_{n\times n}$ o elementach rzeczywistych ($a_{i,j} \in \mathbb{R}$) określone jest pojęcie wyznacznika macierzy (ang. determinant):

$$det_n : A_{n\times n} \to \mathbb{R},$$

który przyporządkowuje macierzy A liczbę rzeczywistą. Poniższa definicja wyznacznika nosi nazwę *wzoru Laplace'a dla 1-go wiersza*[2]

Definicja 2.1 *Wyznacznik stopnia n jest to funkcja* $det_n : A_{n\times n} \to \mathbb{R}$ *określona następująco*

- *Dla* $n = 1$ *niech* $A = (a)$ *wtedy*

$$det_1(A) = a,$$

[2]Definicja wyznacznika macierzy według książki nr [4].

- *dla* $n \geq 2$ *niech* $A = [a_{i,j}]_{i,j=1,\ldots,n}$ *wtedy*

$$det_n(A) = \sum_{j=1}^{n} (-1)^{1+j} a_{1,j} det_{n-1}(A^{(1,j)}), \tag{3}$$

gdzie $A^{(1,j)}$ *powstaje z macierzy* A *przez pominięcie* 1*-go wiersza i* j*-tej kolumny.*

W dalszym ciągu na ogół będziemy pomijać wskaźnik n, pisząc det zamiast det_n.

Własności wyznacznika:

1. Transpozycja macierzy nie powoduje zmiany wartości jej wyznacznika.
2. Zamiana miejscami dwóch dowolnych kolumn lub wierszy zmienia znak wyznacznika, nie zmieniając jego wartości bezwzględnej.
3. Jeśli dwa wiersze lub dwie kolumny macierzy są proporcjonalne (np. są równe), wyznacznik ma wartość zero.
4. Jeśli jakiś wiersz jest kombinacją liniową innych wierszy (np. wiersz składa się tylko z zer), wyznacznik ma wartość zero. To samo dotyczy kolumn.
5. Pomnożenie dowolnej kolumny lub dowolnego wiersza przez stałą mnoży przez tę samą stałą wartość wyznacznika.
6. Dodając lub odejmując od dowolnego wiersza/ kolumny inny wiersz/ kolumnę lub kombinacje liniowe innych wierszy/ kolumn nie zmieniamy wartości wyznacznika.
7. Wyznacznik iloczynu macierzy jest równy iloczynowi wyznaczników: $det(A \cdot B) = det(A) \cdot det(B)$.
8. Wyznacznik macierzy odwrotnej jest równy odwrotności wyznacznika: $det(A^{-1}) = (detA)^{-1}$.
9. Zachodzi $det(k \cdot A) = k^n \cdot det(A)$, gdzie k jest dowolną liczbą, n stopniem macierzy A.

Macierz A nazwiemy nieosobliwą wtedy i tylko wtedy gdy $det(A) \neq 0$

Minor

Definicja 2.2 [3] *Dla dowolnej macierzy A, każdą macierz kwadratową powstałą przez skreślenie pewnej liczby kolumn i wierszy nazywamy* minorem *macierzy A;*

Minor macierzy A powstały przez wykreślenie i-tego wiersza oraz j-tej kolumny będziemy oznaczać przez $M_{i,j}$

Macierz odwrotna

Macierz odwrotną do macierzy kwadratowej A oznaczamy przez A^{-1}. Macierz odwrotna spełnia następujące równanie:

$$A \cdot A^{-1} = A^{-1} \cdot A = I_n,$$

gdzie I_n oznacza macierz jednostkową stopnia n.

Każda kwadratowa macierz nieosobliwa posiada macierz odwrotną i mówimy wtedy, że macierz ta jest odwracalna. Jedną z metod wyznaczania macierzy odwrotnej jest metoda dopełnień algebraicznych przedstawiona poniższym wzorem:

$$A^{-1} = \frac{A_D^T}{det(A)}, \tag{4}$$

gdzie A_D oznacza macierz dopełnień algebraicznych, której elementy wyznaczone są następująco:

$$a_{ij} = (-1)^{i+j} \cdot det(M_{ij}),$$

Inną metodą wyznaczania macierzy odwrotnej jest metoda Gaussa, która będzie omówiona w części dotyczącej algorytmu eliminacji Gaussa.

Omówmy pewne własności macierzy odwrotnej.
Załóżmy, że macierze A i B są odwracalne, tzn. istnieją dla nich macierze odwrotne. Wówczas prawdziwe są następujące własności:

- $(A^{-1})^{-1} = A$,
- $det(A^{-1}) = \frac{1}{det(A)}$,
- $(A^T)^{-1} = (A^{-1})^T$,
- $(A \cdot B)^{-1} = B^{-1} \cdot A^{-1}$.

[3] Definicja minora macierzy według [4].

2.2 Układy równań liniowych

Układem m równań liniowych z n niewiadomymi nazywamy układ postaci:

$$\begin{cases} a_{1,1}x_1 & + & a_{1,2}x_2 & + & \dots & + & a_{1,n}x_n & = b_1 \\ a_{2,1}x_1 & + & a_{2,2}x_2 & + & \dots & + & a_{2,n}x_n & = b_2 \\ a_{3,1}x_1 & + & a_{3,2}x_2 & + & \dots & + & a_{3,n}x_n & = b_3 \\ \vdots & & \vdots & & \ddots & & \vdots & \vdots \\ a_{m,1}x_1 & + & a_{m,2}x_2 & + & \dots & + & a_{m,n}x_n & = b_m \end{cases}, \tag{5}$$

gdzie współczynniki $a_{i,j}$ definiują pewną macierz $A = [a_{i,j}]$ zwaną macierzą układu, stałe b_i tworzą wektor wyrazów wolnych $\vec{b} = [b_i]$, natomiast x_j są niewiadomymi szukanymi w układzie równań.
Powyższy układ można zatem zapisać macierzowo :

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \dots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \dots & a_{m,n} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_m \end{bmatrix}. \tag{6}$$

Traktując wektor $\vec{b}$ oraz $\vec{x}$ jako macierze jednokolumnowe, możemy zapisać prościej:

$$A \cdot X = B. \tag{7}$$

Przy czym jeśli macierz A jest rozmiaru $m \times n$ to macierz X musi być rozmiaru $n \times 1$, a macierz B rozmiaru $m \times 1$. Macierz A nazywamy macierzą główną układu równań liniowych, macierz X nazywamy macierzą niewiadomych, a macierz B macierzą wyrazów wolnych. Rozwiązaniem układu (5) jest ciąg liczb $(x_1, \dots, x_n)$ rzeczywistych, które spełniają ten układ. Układ równań liniowych, który nie posiada rozwiązania, nazywamy układem sprzecznym.

Układ nazywamy jednorodnym, jeśli macierz wyrazów wolnych B jest macierzą zerową, w przeciwnym wypadku układ nazywamy niejednorodnym. Macierzą rozszerzoną układu nazywamy macierz A z dopisaną na końcu kolumną B wyrazów wolnych.

Twierdzenie 2.1 ***Twierdzenie Kroneckera-Capellego***
Układ (5) posiada rozwiązanie wtedy i tylko wtedy gdy rząd macierzy A jest równy rzędowi macierzy rozszerzonej.

Oznaczmy ten rząd przez r oraz przez m liczbę niewiadomych, wtedy dodatkowo

- jeśli $r = m$ to układ ma jedno rozwiązanie i nazywa się oznaczonym,
- jeśli $r < m$ to układ ma nieskończenie wiele rozwiązań zależnych od $m - r$ parametrów i nazywa się nieoznaczonym.

Układ Cramera
Wprowadźmy następującą definicję [4]

Definicja 2.3 *Układem Cramera nazywamy układ (7), w którym macierz układu A jest macierzą kwadratową nieosobliwą.*

Rozwiązanie X układu Cramera jest określone następującym wzorem:

$$X = \frac{1}{det(A)} \begin{bmatrix} det(A_1) \\ det(A_2) \\ \vdots \\ det(A_n) \end{bmatrix} \tag{8}$$

nazywanym wzorem Cramera. Macierze A_j $1 \leq j \leq n$ powstają przez zastąpienie j-tej kolumny w macierzy A kolumną wyrazów wolnych B. Zatem każdy z elementów macierzy (kolumny) X można wyznaczyć następująco

$$x_j = \frac{det(A_j)}{det(A)} \text{ dla } 1 \leq j \leq n$$

Zapis (7) przypomina w istocie zwykłe równanie liniowe, którego rozwiązanie można by otrzymać z prostego przekształcenia

$$X = \frac{B}{A}.$$

Wiemy jednak, że jako takie dzielenie macierzy przez macierz nie istnieje [5]. Korzystamy tutaj z faktu, że iloraz jest równy iloczynowi dzielnej i elementu

[4] Definicja wg. książki [2] .

[5] Iloraz macierzy po elementach, tzn. $b_{1,1}/a_{1,1}$, $b_{1,2}/a_{1,2}\ldots$ nie prowadzi tutaj do rozwiązania, ponieważ macierze B i A musiałyby być jednakowych rozmiarów, a w omawianym przypadku macierz A jest macierzą kwadratową rozmiaru $n \times n$, natomiast B jest macierzą $n \times 1$, czyli kolumną.

odwrotnego do dzielnika. Otrzymujemy zatem równanie:

$$X = A^{-1} \cdot B, \tag{9}$$

które definiuje jednocześnie jedyne rozwiązanie nieosobliwego układu równań Cramera.
Zatem warunkiem koniecznym i dostatecznym na to, aby układ Cramera posiadał rozwiązanie jest nieosobliwość macierzy układu A.

Kolejności składników w iloczynie $A^{-1} \cdot B$ nie jest przypadkowa i wynika z następujących własności:

$$A^{-1} \cdot A = I,$$
$$I \cdot X = X.$$

Aby zatem otrzymać po lewej stronie $I \cdot X$ mnożymy lewostronnie obie strony równania przez macierz odwrotną A^{-1} otrzymując ostatecznie postać (9).

Metoda Eliminacji Gaussa-Jordana dla układów Cramera

Niech $A \cdot X = B$ będzie układem Cramera z nieosobliwą macierzą układu A stopnia n. Metoda eliminacji Gaussa Jordana rozwiązania tego układu polega na wykonaniu kolejnych kroków

1. z macierzy A i B budujemy macierz rozszerzoną układu postaci $[A|B]$

$$[A|B] = \left[\begin{array}{ccccc|c} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} & b_1 \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,n} & b_2 \\ a_{3,1} & a_{3,2} & a_{3,3} & \dots & a_{3,n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \dots & a_{n,n} & b_n \end{array}\right] \tag{10}$$

2. na macierzy rozszerzonej wykonujemy przekształcenia elementarne T1, T2, T3 w celu doprowadzenia jej do postaci

$$[I|X] = \left[\begin{array}{ccccc|c} 1 & 0 & 0 & \dots & 0 & x_1 \\ 0 & 1 & 0 & \dots & 0 & x_2 \\ 0 & 0 & 1 & \dots & 0 & x_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & x_n \end{array}\right]$$

gdzie I jest macierzą jednostkową, a X macierzą kolumnową zawierającą kolejne rozwiązania danego układu równań liniowych.

Podczas przekształcania macierzy rozszerzonej do postaci $[I|X]$ stosujemy algorytm eliminacji Gaussa Jordana sprowadzania macierzy do postaci jednostkowej.

Metoda Gaussa-Jordana wyznaczania macierzy odwrotnej

Przypomnijmy sobie definicję macierzy odwrotnej do kwadratowej macierzy A. Jest to taka macierz A^{-1}, która spełnia równanie :

$$A \cdot A^{-1} = A^{-1} \cdot A = I_n.$$

Metoda eliminacji Gaussa-Jordana wyznaczania macierzy odwrotnej polega na zastosowaniu, przedstawionego wcześniej, algorytmu Gaussa-Jordana na macierzy połączonej $[A|I]$. Stosujemy tutaj, tak jak w przypadku rozwiązywania układów równań liniowych, przekształcenia elementarne, aż do momentu uzyskania po lewej stronie macierzy jednostkowej I. Wtedy macierz po prawej stronie jest szukaną macierzą odwrotną:

$$[I|A^{-1}].$$

Jak widać algorytm Gaussa-Jordana ma bardzo szerokie zastosowanie, zarówno do rozwiązywania układów równań liniowych, znajdywania macierzy odwrotnej, ale także do innych operacji na macierzach.
Przy pomocy tego algorytmu można bowiem uzyskać również wyznacznik macierzy oraz jej rząd.

2.3 Algorytm Gaussa

Algorytm eliminacji Gaussa polega w każdym i-tym kroku na eliminacji i-tej niewiadomej z równań od $i+1$ do n. Wykorzystuje się tutaj przekształcenia elementarne $T1, T2, T3$ tak kombinując równania, aby z postaci $[A|B]$ otrzymać postać $[R|B']$ z macierzą trójkątną górną R oraz pewną zmienioną macierzą B'. Następnie wykonujemy postępowanie w przeciwnym kierunku, tzn. zaczynając od ostatniego równania w każdym kolejnym $(n-i)$-tym kroku eliminujemy zmienną numer $(n-i)$ z równań od $(n-i-1)$ do 1 otrzymując ostatecznie postać $[I|X]$, gdzie X jest szukanym rozwiązaniem układu równań.

W algorytmie Gaussa stosuje się postępowanie iteracyjne, tzn. ustalony układ działań powtarza się w każdym kolejnym kroku:

1. dzielimy cały pierwszy wiersz układu (10) przez element $a_{1,1}$,
2. dla $i = 2, \ldots, n$ od każdego i-tego wiersza odejmujemy wiersz pierwszy pomnożony przez element $a_{i,1}$ tak, aby wyzerować kolumnę pierwszą począwszy od wiersza 2-go,
3. dzielimy cały drugi wiersz układu (10) przez element $a_{2,2}$,
4. dla $i = 3, \ldots, n$ od każdego i-tego wiersza odejmujemy wiersz drugi pomnożony przez element $a_{i,2}$ tak, aby wyzerować kolumnę drugą począwszy od wiersza 3-go.

Powtarzamy powyższe kroki dla każdego wiersza, aż do uzyskania po lewej stronie macierzy trójkątnej górnej $R_{n\times n}$, która dodatkowo na diagonali będzie posiadać same jedynki. Macierz ta będzie postaci

$$R = \begin{bmatrix} 1 & r_{1,2} & r_{1,3} & \ldots & r_{1,n} \\ 0 & 1 & r_{2,3} & \ldots & r_{2,n} \\ 0 & 0 & 1 & \ldots & r_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \end{bmatrix}. \tag{11}$$

Tak otrzymana postać układu pozwala już w zasadzie na proste obliczenie rozwiązania. Wystarczy bowiem, zaczynając od ostatniej zmiennej, do każdego i-tego równania podstawić odczytane wcześniej wartości niewiadomych o indeksach od $i, \ldots n$ i obliczyć wartość i-tej niewiadomej.

Aby jednak doprowadzić układ (10) do postaci $[I|X]$ trzeba dodatkowo jeszcze wykonać powyższą eliminację zmiennych w odwrotnej kolejności, tzn.:

1. dla $i = n-1, \ldots, 1$ od każdego i-tego wiersza odejmujemy wiersz numer n pomnożony przez element $a_{i,n}$ tak, aby wyzerować n-tą kolumnę do wiersza $n-1$ włącznie,
2. dla $i = n-2, \ldots, 1$ od każdego i-tego wiersza odejmujemy wiersz numer $n-1$ pomnożony przez element $a_{i,n-1}$ tak, aby wyzerować kolumnę numer $n-1$ od wiersza 1-go do $(n-2)$-go włącznie.

Kroki te powtarzamy aż do wiersza pierwszego, w wyniku czego otrzymujemy rozwiązanie, które odczytujemy z kolumny X.

Algorytm Gaussa z wyborem elementu głównego

Algorytm przedstawiony powyżej nie zadziała poprawnie w sytuacji, gdy w i-tym kroku eliminacji element $a_{i,i}$ będzie równy zero. W każdym i-tym kroku eliminacji cały i-ty wiersz macierzy jest dzielony właśnie przez element diagonalny $a_{i,i}$, co w przypadku $a_{i,i} = 0$ doprowadzi do niedozwolonego dzielenia przez zero.

Rozwiązaniem tego problemu jest modyfikacja metody o dodanie wyboru elementu głównego (centralnego) w kolumnie. Oznacza to, że w każdym i-tym kroku eliminacji poszukujemy elementu $a_{k,i}$ dla $k = i+1, \ldots, n$, takiego, że:

$$|a_{k,i}| > a_{i,i} \text{ oraz } a_{k,i} \neq 0.$$

Jeśli elementu takiego nie znajdziemy oznacza to że macierz jest nieosobliwa[6], czyli układ nie jest cramerowski. Jeśli założymy, że znaleziony element maksymalny znajduje się w wierszu numer r, w następnym kroku zamieniamy wiersz i-ty z r-tym, i kontynuujemy obliczenia.

Metoda eliminacji Gaussa z wyborem elementu głównego daje korzystne wyniki nie tylko ze względu na możliwość uniknięcia dzielenia przez element zerowy. Również od strony obliczeń numerycznych jest zalecana do obliczania rozwiązań układów równań liniowych, aniżeli eliminacja bez wyboru elementu głównego. Nawet jeśli liczba $a_{i,i}$, przez którą dzielimy cały i-ty wiersz, nie jest równa zero, to jednak może być bardzo bliska zeru, co w przypadku operacji dzielenia w arytmetyce zmiennopozycyjnej może powodować bardzo duże, trudne do oszacowania błędy. Błędy te, zwielokrotniane w każdym kroku eliminacji mogą doprowadzić do całkowicie niepoprawnych wyników.

Inną modyfikacją omawianej metody jest algorytm eliminacji Gaussa z pełnym wyborem elementu głównego. Pełen wybór oznacza tutaj poszukiwanie maksymalnego, niezerowego elementu w całej macierzy A. Zauważmy, że każdy kolejny i-ty krok eliminacji Gaussa polega na wykonywaniu obliczeń na pod-macierzy A' wyrwanej z macierzy A i złożonej z wierszy (kolumn) o indeksach od i do n. Zatem w tym przypadku element główny będzie szukany w pod-macierzy A', tzn. szukamy takiego $a_{k,l} \neq 0$ dla $k, l = i, \ldots n$

[6] W przypadku, gdy $i = 1$, oznacza, że cała kolumna jest zerowa, czyli oczywiście $det(A) = 0$, natomiast dla pozostałych wartości i oznacza, że co najmniej wszystkie $a_{i,j}$ dla $j = i, \ldots, n$ są równe zero, co z kolei oznacza, że pod-macierz złożona z wierszy i kolumn o indeksach $k = i, \ldots, n$ jest osobliwa, zatem cały układ jest nieoznaczony.

który spełnia:

$$|a_{k,l}| > a_{i,i}.$$

Trzeba tutaj pamiętać jednak, że zamiana miejscami dwóch kolumn zamienia kolejność zmiennych w rozwiązaniu.
Takie postępowanie daje większą, niż wybór tylko w kolumnie, dokładność obliczonych wartości, jednak niestety kosztem czasu obliczeń. Dla przykładu dla macierzy rozmiaru 10×10 w każdym kroku trzeba dodatkowo wykonać $(11 - i) \cdot (11 - i) \approx 400$ operacji porównania. Z tego powodu częściej jest stosowana metoda z wyborem elementu głównego w kolumnie.

3 Arytmetyka przedziałowa

Przypuśćmy, że mamy dokonać pomiarów pewnego prostokątnego obszaru[7]. Nie ważne jakbyśmy uważnie odczytywali wartości z miary, nigdy nie możemy być pewni dokładnych rozmiarów obszaru, który mierzymy. Co najwyżej możemy powiedzieć z pewnością, że szerokość i długość leżą w pewnych granicach.
Załóżmy zatem, że wiemy tylko, że długość nie jest mniejsza niż 68 metrów i nie większa niż 72, a szerokość nie mniejsza niż 49 i nie większa niż 51 metrów. Poczynając takie założenia możemy teraz śmiało stwierdzić, że pole mierzonego obszaru jest wartością na pewno nie mniejszą niż 3332 metry kwadratowe i nie większą niż 3672 metry kwadratowe. Obliczając te wartości użyliśmy właśnie arytmetyki przedziałowej, ponieważ $[68, 72] \times [49, 51] = [3332, 3672]$. Arytmetyka przedziałowa daje nam zatem oszacowanie wyniku proporcjonalne do szerokości danych wejściowych.

3.1 Wprowadzenie

Przedziałem domkniętym (interwałem) w sensie arytmetyki przedziałowej będziemy nazywali każdy zbiór liczb rzeczywistych ograniczony z góry i z dołu i oznaczać będziemy w nawiasach kwadratowych. Formalnie przedział $[\underline{x}, \overline{x}]$ zapiszemy jako

$$[\underline{x}, \overline{x}] = \{x \in \mathbb{R} \colon \underline{x} \leq x \leq \overline{x}\},$$

przy czym $\underline{x}$ oznacza dolny kres przedziału, a $\overline{x}$ górny kres przedziału. Zwykle przedział będziemy zapisywać podając jego oba brzegi $[\underline{x}, \overline{x}]$ albo $[a, b]$ lub po prostu jako $[x]$, ale będzie to jasno wynikało z kontekstu.

Zbiór wszystkich przedziałów domkniętych będziemy oznaczać przez $\mathbb{IR}$. Dwa przedziały rzeczywiste $[x]$ i $[y]$ są równe wtedy i tylko wtedy gdy $\underline{x} = \underline{y}$ oraz $\overline{x} = \overline{y}$.

Dla przedziałów określone są szerokość (średnica), środek i promień odpowiednio:

$$\text{szerokość - } w([x]) = \overline{x} - \underline{x},$$

[7] Przykład zaczerpnięty z [5]

$$\text{środek - } m([x]) = \frac{\underline{x} + \overline{x}}{2},$$

$$\text{promień - } r([x]) = \frac{\overline{x} - \underline{x}}{2}.$$

Dodatkowo, zbiór przedziałów punktowych, czyli przedziałów o szerokości równej zero, będziemy oznaczać przez $\mathbb{IR}^*$

3.2 Działania arytmetyczne

Na zbiorze $\mathbb{IR}$ zdefiniowane są podstawowe działania arytmetyczne dodawania, odejmowania, mnożenia i dzielenia $(+, -, \cdot, /)$ następująco:

$$[x] + [y] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}],$$
$$[x] - [y] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}],$$
$$[x] \cdot [y] = [\min\{\underline{x}\overline{y}, \overline{x}\underline{y}, \underline{x}\underline{y}, \overline{x}\overline{y}\}, \max\{\underline{x}\overline{y}, \overline{x}\underline{y}, \underline{x}\underline{y}, \overline{x}\overline{y}\}],$$
$$[x]/[y] = [x] \cdot [1/\overline{y}, 1/\underline{y}] \; 0 \notin [y].$$

Wyrażenie $0 \notin [y]$ oznacza, że przedział $[y]$ nie zawiera zera, zatem albo $\underline{y} > 0$ albo $\overline{y} < 0$.
Dla przedziałów postaci $[a, a]$ tzn. $\underline{x} = a$ oraz $\overline{x} = a$ (tzw. przedział punktowy) mamy:

$$[a, a] + [b, b] = [a + b, a + b],$$
$$[a, a] - [b, b] = [a - b, a - b],$$
$$[a, a][b, b] = [ab, ab],$$
$$[a, a]/[b, b] = [a/b, a/b] \; b \neq 0.$$

Arytmetyka przedziałowa zawiera więc w sobie arytmetykę rzeczywistą traktując liczby rzeczywiste jako przedziały punktowe, czyli przedziały o szerokości równej zero. Powyższe operacje są rozszerzeniem elementarnych działań arytmetycznych w zbiorze liczb rzeczywistych $\mathbb{R}$. Oznacza to, że dla dowolnych przedziałów $[x]$ i $[y]$ oraz dla dowolnej liczby rzeczywistej $a \in [x]$ i dowolnej liczby rzeczywistej $b \in [y]$ zachodzi, że:

$$a \circ b \in [x] \circ [y] gdzie \circ \in \{+, -, \cdot, /\}.$$

Zatem przedział będący wynikiem obliczeń wykonanych na argumentach przedziałowych zawiera w sobie rozwiązanie dokładne w kontekście arytmetyki rzeczywistej. Można więc powiedzieć, że zbiór liczb rzeczywistych jest podzbiorem zbioru przedziałów rzeczywistych, a arytmetyka przedziałowa jest uogólnieniem arytmetyki rzeczywistej. Dla przykładu, jeśli

$$f(x) = x(x - 1), \tag{12}$$

wtedy wartość funkcji $f(x)$ dla argumentu $[0, 1]$ będzie równa

$$f([0, 1]) = [0, 1]([0, 1] - 1) = [0, 1][-1, 0] = [-1, 0]. \tag{13}$$

Otrzymane rozwiązanie obliczone przy pomocy zastosowania arytmetyki przedziałowej zawiera dokładny wynik rzeczywisty, równy w tym przypadku

$$[-1/4, 0].$$

Odległość $q([x],[y])$ między dwoma przedziałami jest zdefiniowana wzorem:

$$q([x],[y]) = max\{|\underline{x} - \underline{y}|, |\overline{x} - \overline{y}|\}. \tag{14}$$

Odległość jest nieujemna, nie zależy od kolejności argumentów, równa *zero* wtedy i tylko wtedy gdy $[x] = [y]$ oraz spełnia nierówność trójkąta. Jest zatem metryką, a przestrzeń $\mathbb{IR}$ (przedziałów rzeczywistych) z tak zdefiniowaną metryką jest przestrzenią metryczną. Dowód tego faktu przedstawiony jest poniżej.

Dowód.

Zwrotność:

$$\begin{aligned}
&\mathrm{q}([\mathrm{x}],[\mathrm{y}]) = 0 \Leftrightarrow \\
&\max(|\underline{\mathrm{x}} - \underline{y}|, |\overline{x} - \overline{y}|) = 0 \Leftrightarrow \\
&|\underline{\mathrm{x}} - \underline{y}| = 0 \wedge |\overline{x} - \overline{y}| = 0 \Leftrightarrow \\
&\underline{\mathrm{x}} = \underline{y} \wedge \overline{x} = \overline{y} \Leftrightarrow [x] = [y].
\end{aligned}$$

Symetryczność:

$$\begin{aligned}
&q([x],[y]) = \\
&\max(|\underline{\mathrm{x}} - \underline{y}|, |\overline{x} - \overline{y}|) = \\
&\max(|-(\underline{y} - \underline{\mathrm{x}})|, |-(\overline{y} - \overline{x})|) = \\
&\max(|\underline{y} - \underline{\mathrm{x}}|, |\overline{y} - \overline{x}|) = \\
&q([y],[x]).
\end{aligned}$$

Przechodniość:

$$\begin{aligned}
&\forall_{[x],[y],[z] \in \mathbb{IR}} \; q([x],[z]) = \\
&\max(|\underline{x} - \underline{z}|, |\overline{x} - \overline{z}|) = \\
&\max(|\underline{x} - \underline{z} - \underline{y} + \underline{y}|, |\overline{x} - \overline{z} - \underline{y} + \underline{y}|) = \\
&\max(|\underline{x} - \underline{y} + \underline{y} - \underline{z}|, |\overline{x} - \overline{y} + \overline{y} - \overline{z}|) \leq \\
&\max(|\underline{x} - \underline{y}| + |\underline{y} - \underline{z}|, |\overline{x} - \overline{y}| + |\overline{y} - \overline{z}|) \leq \\
&\max(|\underline{x} - \underline{y}|, |\overline{x} - \overline{y}|) + \max(|\underline{y} - \underline{z}|, |\overline{y} - \overline{z}|) = \\
&q([x],[y]) + q([y],[z]).
\end{aligned}$$

□

3.3 Własności działań w $\mathbb{IR}$

Zbadamy podstawowe własności działań na przedziałach rzeczywistych. Działania dodawania oraz mnożenia w przestrzeni $\mathbb{IR}$ są działaniami wewnętrznymi, tzn.

$$\otimes : \mathbb{IR} \times \mathbb{IR} \to \mathbb{IR} \text{ gdzie } \otimes \in \{+, \cdot\}. \tag{15}$$

W przestrzeni $\mathbb{IR}$ działanie dodawania jest łączne:

$$([x] + [y]) + [z] = ([\underline{x} + \underline{y}, \overline{x} + \overline{y}]) + [\underline{z}, \overline{z}] = [\underline{x} + \underline{y} + \underline{z}, \overline{x} + \overline{y} + \overline{z}] =$$

$$[\underline{x}, \overline{x}] + ([\underline{y} + \underline{z}, \overline{y} + \overline{z}]) = [x] + ([y] + [z]).$$

Jest również przemienne:

$$[x] + [y] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}] = [\underline{y} + \underline{x}, \overline{y} + \overline{x}] = [y] + [x]. \tag{16}$$

Własności te zachodzą również dla iloczynu przedziałów. Łatwo wskazać element neutralny dla tych działań. Dla dodawania elementem neutralnym jest przedział $[0, 0]$, natomiast dla mnożenia przedział $[1, 1]$:

$$[x] + [0, 0] = [\underline{x} + 0, \overline{x} + 0] = [\underline{x}, \overline{x}] = [x] \tag{17}$$

$$[x] \cdot [1, 1] = [\min\{\underline{x} \cdot 1, 1 \cdot \overline{x}\}, \max\{\underline{x} \cdot 1, 1 \cdot \overline{x}\}] = [\underline{x}, \overline{x}] = [x]. \tag{18}$$

Element przeciwny w ogólności nie istnieje w arytmetyce przedziałowej, ponieważ równość

$$[x] - [y] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}] = [0, 0] \tag{19}$$

zachodzi tylko dla $\underline{x} = \overline{y}$ oraz $\overline{x} = \underline{y}$. Dla dowolnego przedziału $[x]$ mamy, że $\underline{x} \leq \overline{x}$, zatem dla przedziału który by spełniał powyższą równość musiało by zachodzić, że $\overline{y} \leq \underline{y}$, co znaczyło by że $[y]$ nie jest przedziałem w ramach przyjętych założeń o interpretacji przedziałów rzeczywistych domkniętych. W innym przypadku, zakładając, że oba przedziały $[x]$ oraz $[y]$ spełniają wspomniane założenia, czyli $\underline{x} \leq \overline{x}$ oraz $\underline{y} \leq \overline{y}$, mamy, że:

$$[x] - [y] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}] = [0, 0] \tag{20}$$

zachodzi tylko dla

$$\underline{x} = \overline{x} = \underline{y} = \overline{y}. \tag{21}$$

Jedynymi przedziałami, które posiadają element przeciwny względem dodawania, są zatem przedziały punktowe, czyli liczby rzeczywiste. Stąd, pozornie trywialna, własność dla arytmetyki rzeczywistej nie jest spełniona w arytmetyce przedziałowej, ponieważ ogólnie, dla dowolnego przedziału domkniętego $[x] \in \mathbb{IR}$, zachodzi nierówność

$$[x] - [x] \neq [0, 0]. \tag{22}$$

Element odwrotny również nie istnieje dla wszystkich przedziałów, z wyjątkiem przedziałów punktowych, ponieważ

$$[x] \cdot [y] = [\min\{\underline{x}\overline{y}, \overline{x}\underline{y}, \underline{x}\underline{y}, \overline{x}\overline{y}\}, \max\{\underline{x}\overline{y}, \overline{x}\underline{y}, \underline{x}\underline{y}, \overline{x}\overline{y}\}] = [1, 1] \tag{23}$$

zachodzi tylko dla

$$\underline{x} = \overline{x} = \underline{y}^{-1} = \overline{y}^{-1}. \tag{24}$$

W arytmetyce przedziałowej również nie zachodzi prawo rozdzielności mnożenia względem dodawania.
Weźmy następujące przedziały $[a, b], [c, c], [-c, -c] \in \mathbb{IR}$. Mamy wtedy, że

$$[a, b]([c, c] + [-c, -c]) = [a, b][0, 0] = [0, 0],$$

natomiast, gdy najpierw pomnożymy

$$[a, b]([c, c] + [-c, -c]) = [a, b][c, c] + [a, b][-c, -c] =$$
$$[\min\{ac, bc\}, \max\{ac, bc\}] + [\min\{-ac, -bc\}, \max\{-ac, -bc\}] \neq [0, 0].$$

W ogólności prawo rozdzielności nie jest zatem spełnione. Zachodzi jednak w przypadku, gdy wspólnym czynnikiem mnożenia jest liczba ze zbioru $\mathbb{IR}^*$, czyli przedział punktowy. Wtedy dla $[a, a] \in \mathbb{IR}^*$ oraz $[b, c], [d, e] \in \mathbb{IR}$ spełnione jest prawo rozdzielności, a dowód tego faktu przebiega w trzech etapach, w zależności czy $a < 0$, $a = 0$, czy $a > 0$. Pokażemy tutaj tylko przypadek dla $a < 0$. Pozostałe dowody przebiegają analogicznie.

$$[a, a]([b, c] + [d, e]) = [a, a][b, c] + [a, a][d, e] =$$
$$[min\{ab, ac\}, max\{ab, ac\}] + [min\{ad, ae\}, max\{ad, ae\}] =$$

jeśli $b \leq c$ to $ab \geq ac$ i podobnie dla pozostałych

$$[ac, ab] + [ae, ad] = [a(c + e), a(b + d)],$$

z drugiej strony jednak

$$[a,a]([b,c]+[d,e]) = [a,a][b+d,c+e] =$$
$$[min\{a(b+d),a(c+e)\},max\{a(b+d),a(c+e)\}] = [a(c+e),a(b+d)].$$

Nie ma zatem prawa rozdzielności mnożenia względem dodawania. Zachodzi jednak jednostronna inkluzja. Otóż dla dowolnych $[x],[y],[z] \in \mathbb{IR}$ mamy, że

$$[x]\cdot([y]+[z]) \subset ([x]\cdot[y])+([x]\cdot[z]).$$

Stąd, z punktu widzenia struktury algebraicznej, zbiór przedziałów domkniętych $\mathbb{IR}$ z działaniem dodawania $(\mathbb{IR},+)$, jest półgrupą przemienną z jedynką. Udowodniliśmy bowiem, że operacja dodawania przedziałów jest wewnętrzna, łączna, przemienna oraz posiada element neutralny. Również struktura $(\mathbb{IR},\cdot)$ z działaniem mnożenia przedziałów jest półgrupą przemienną (abelową) z jedynką. Struktury te nie są grupami, ponieważ zarówno dla dodawania, jak i mnożenia nie istnieje element odwrotny. Implikuje to, że struktura $(\mathbb{IR},+,\cdot)$ nie jest również pierścieniem.

Rozważmy teraz własności dzielenia przedziałów i związane z nimi konsekwencje. Warto zauważyć, że kolejna pozornie trywialna własność nie istnieje w arytmetyce przedziałowej. Bowiem dla dowolnego przedziału domkniętego $[a,b] \in \mathbb{IR}$ w ogólności zachodzi nierówność:

$$\frac{[a,b]}{[a,b]} \neq [1,1]\ ,\ 0 \notin [a,b],$$

ponieważ

$$\frac{[a,b]}{[a,b]} = [a,b]\left[\frac{1}{b},\frac{1}{a}\right] = \left[\min\left(\frac{a}{b},1,\frac{b}{a}\right),\max\left(\frac{a}{b},1,\frac{b}{a}\right)\right] = [1,1]$$

zachodzi tylko w przypadku gdy $a=b$, zatem dla przedziału punktowego.

Ma to szczególne znaczenie podczas projektowania algorytmów komputerowych rozwiązujących zagadnienia matematyczne, w których wykorzystywane są wyrażenia ułamkowe oraz wielomiany. Biorąc pod uwagę pokazaną przed chwilą "anomalię" nie możemy już wykorzystać prostego przekształcenia sprowadzenia wyrażenia do wspólnego mianownika, bowiem

$$[a,b]+\frac{[1,1]}{[a,b]} = \frac{[a,b]^2+[1,1]}{[a,b]}$$

nie zachodzi dopóki a nie jest równe b.

3.4 Funkcje w $\mathbb{IR}$

Własności funkcji na zbiorze przedziałów domkniętych mają szczególne znaczenie przy implementacji przedziałowych odpowiedników funkcji rzeczywistych. Funkcje przedziałową definiujemy następująco:

Niech $f : D \subset \mathbb{R} \to \mathbb{R}$ będzie funkcją rzeczywistą, ciągłą na przedziale D, wtedy rozszerzenie funkcji f na argumenty przedziałowe $[x] \in \mathbb{IR}$ ma postać:

$$f([x]) = \{f(x) : x \in [x]\}. \tag{25}$$

Przy czym od razu widać inkluzję

$$\underset{x \in [x]}{\forall} f(x) \subset f([x]). \tag{26}$$

Trzeba jednak mieć świadomość, że jakkolwiek wartość funkcji $f(x)$ argumentu rzeczywistego zawsze zawiera się w przedziale będącym wynikiem odpowiedniej funkcji przedziałowej f([x]) to jednak implementacja funkcji na argumentach przedziałowych powinna być gruntownie przemyślana, aby uniknąć nieoczekiwanych wyników.

Dla większości monotonicznych funkcji rzeczywistych zdefiniowano ich przedziałowe odpowiedniki.

$$abs([x]) = \begin{cases} [\min(|\underline{x}|, |\overline{x}|), \max(|\underline{x}|, |\overline{x}|)] & \text{gdy } 0 \notin [x] \\ [0, \max(|\underline{x}|, |\overline{x}|)] & \text{gdy } 0 \in [x] \end{cases},$$

$$[x]^n (n \in \mathbb{N}) = \begin{cases} [\underline{x}^n, \overline{x}^{-n}] & \text{gdy } (n \text{ - nieparzyste) lub } (\underline{x} \geqslant 0) \\ [\underline{x}^{-n}, \overline{x}^n] & \text{gdy } (n \text{ - parzyste) i } (\overline{x} \leqslant 0) \\ [0, abs([x])^n] & \text{gdy } (n \text{ - parzyste) i } (0 \in [x]) \end{cases},$$

$$\sqrt[n]{[x]}(n \in \mathbb{N}) = [x]^{\frac{1}{n}} = [\sqrt[n]{\underline{x}}, \sqrt[n]{\overline{x}}] \quad \text{gdy } (n\text{-nieparzyste) lub } (\underline{x} \geq 0),$$

$$e^{[x]} = [e^{\underline{x}}, e^{\overline{x}}],$$

$$f([x]) = [f(\underline{x}), f(\overline{x})], \text{ gdzie } f \in \{tg, ctg, arctg, arcsinh, ln, sinh\},$$

$$f([x]) = [f(\overline{x}), f(\underline{x})], \text{ gdzie } f \in \{arcctg, arctgh\}.$$

4 Implementacja

Głównym celem niniejszej pracy jest porównanie oraz dokonanie analizy dokładności arytmetyki przedziałowej i arytmetyki rzeczywistej w komputerach klasy PC. Wynika z tego jednoznacznie potrzeba implementacji owych arytmetyk oraz wszystkich, opisanych w tej pracy, struktur i działań arytmetycznych.

Do zrealizowania tego zadania napisany został program implementujący wspomniane zagadnienia. Jako integralna część niniejszej pracy zostanie on omówiony w następnych rozdziałach. Program ten nazwany *IntComp* (z ang. Interval Computations) napisany został w języku C++ w postaci okienkowego interpretatora składni matematycznej. Język C++ wybrany został ze względu na swoją elastyczność oraz ogrom informacji i źródeł wiedzy na jego temat.

W następnych rozdziałach przedstawiona będzie implementacja struktur danych oraz działań w programie *IntComp*. Ze względu na charakter pracy, implementacje arytmetyki przedziałowej oraz rzeczywistej będą omawiane równolegle.

4.1 Struktury danych

Arytmetyka przedziałowa została zaimplementowana w języku C++ jako klasa *Interval*. Krańce przedziałów, początek i koniec, oznaczono odpowiednio jako $\_i$ oraz $\_s$

```
1 class Interval
2 {
3 public:
4   LDouble _i, _s;
5   ...
6 }
```

przy czym *LDouble* jest jednoznaczny z najszerszym typem języka C++ `long double`. Typ `long double` jest to typ zmiennopozycyjny, poszerzonej precyzji [8]. Jest odpowiednikiem typu `Extended` języka Turbo Pascal.

Klasa *Interval* zawiera metody konwersji z typu tekstowego, całkowitego oraz rzeczywistego. Ponadto implementacja obejmuje również operatory arytmetyczne: dodawania, odejmowania, mnożenia oraz dzielenia oraz podstawowe funkcje przedziałowe: szerokość przedziału, środek przedziału oraz promień.

Struktury danych reprezentujące macierze oraz wektory również zostały zaimplementowane jako klasy. Odpowiednio *IntMatrix* oraz *RealMatrix* reprezentują klasy implementujące dynamiczne tablice wskaźnikowe. Klasy te zawierają wbudowane mechanizmy kontroli zawartości oraz implementacje działań na macierzach. Szczegółową implementację można prześledzić w dołączonych źródłach programu.

[8] Typ poszerzonej precyzji zajmuje 80 bitów w wewnętrznej reprezentacji liczby (64 bity - mantysa, 15 bitów - cecha oraz 1 bit znaku)

4.2 Macierz i wektor

W programie *IntComp* wykorzystano dynamiczną tablicę wskaźników jako strukturę do reprezentacji macierzy, wektorów. Takie rozwiązanie daje większą kontrolę nad pamięcią komputera oraz zakresami indeksów elementów macierzy czy wektora. Unika się w ten sposób, między innymi, sytuacji kiedy obliczenia obejmują elementy tablicy, które nie należą do reprezentowanej przez tą tablicę macierzy czy wektora.

Tablica wskaźników składa się z wiersza (dynamiczna tablica jednowymiarowa), którego elementami są wskaźniki na kolumny (dynamiczne tablice jednowymiarowe) zawierające z kolei wskaźniki na elementy danego typu. Te ostatnie mogą być obiektami klasy *Interval* w przypadku danych przedziałowych, lub liczbami rzeczywistymi typu `long double` w przypadku struktur w arytmetyce rzeczywistej. Klasy *IntMatrix* oraz *RealMatrix* zawierają implementację podstawowych działań arytmetycznych na macierzach oraz metody rozwiązywania układów równań liniowych, o czym będzie mowa w następnym rozdziale.

Poniżej przedstawiono nagłówek klasy *IntMatrix* reprezentującej przedziałową macierz lub wektor.

```
1
2 class IntMatrix
3 {
4 private:
5   void allocElem(int rows,int cols);
6
7 public:
8   int RowCnt, ColCnt;
9   Interval** _el;
10
11  IntMatrix();
12  IntMatrix(int _rows,int _cols);
13  IntMatrix(Interval x);
14  ...
15 }
```

Prywatna metoda *allocElem* służy do rezerwowania pamięci potrzebnej na nowo tworzoną macierz. Zmienne *RowCnt* oraz *ColCnt* zawierają informację o ilości wierszy i kolumn macierzy. Ich wartości są ustalane w momencie

tworzenia macierzy w pamięci. Zmienna *_el* zawiera wskaźniki na elementy macierzy. Istotne tutaj jest to, że w języku C++ tablice dynamiczne zawsze są indeksowane od zera co ma znaczenie w późniejszej implementacji działań i algorytmów algebraicznych.

Wektory są reprezentowane jako tablice jednowymiarowe w postaci wiersza tablicy lub odpowiednio w postaci kolumny, przy czym łatwo określić czy wektor jest wierszem czy kolumną, sprawdzając która ze zmiennych *RowCnt* oraz *ColCnt* jest równa jeden.
Analogicznie wygląda implementacja struktury *RealMatrix* reprezentującej macierze i wektory w arytmetyce rzeczywistej.

Mnożenie macierzy A i B zrealizowane jest w następującej postaci

```
1 for (i=0;i<RowCnt;i++)
2  for (j=0;j<ColCnt;j++)
3   {
4    result->_el[i][j]=0.0;
5    for (k=0;k<RowCnt;k++)
6     {
7      result[i][j]=result[i][j]+A[i][k]*B[k][j];
8     }
9   }
10 return (result);
```

4.3 Układ równań liniowych

Układ równań liniowych postaci (7) w programie *IntComp* jest reprezentowany jako macierz A typu *IntMatrix* oraz wektor B tego samego typu. W programie zostały zaimplementowane metody Gaussa rozwiązywania układów równań liniowych. Dla potrzeb testów zakres tej implementacji objął trzy warianty metody Gaussa: z pełnym wyborem elementu głównego, z częściowym wyborem elementu głównego oraz bez wyboru elementu głównego. Implementacja tych metod jest identyczna zarówno dla macierzy przedziałowych, jak i rzeczywistych, różni się tylko operowanym typem danych.

Metoda Gaussa bez wyboru elementy głównego

Implementacja metody Gaussa bez wyboru elementu głównego (centralnego) w programie *IntComp* wykorzystuje algorytm Gaussa rozkładu macierzy do postaci trójkątnej górnej.

```
1  IntMatrix Gauss(IntMatrix& b)
2  {
3      if (ColCnt!=RowCnt) return;
4
5      int i,j,k;
6      Interval mul;
7      int n=RowCnt;
8
9      if ((b.ColCnt!=n) && (b.RowCnt==n))
10      b=b.Transpose();
11
12     for (k=0;k<n-1;k++)
13     {
14       for (i=k+1;i<n;i++)
15       {
16         if (_el[k][k].iszero()) return;
17         mul=_el[i][k]/_el[k][k];
18         for (j=k+1;j<n;j++)
19           _el[i][j]= _el[i][j] - mul*_el[k][j];
20
21         b._el[0][i]=b._el[0][i]-mul*b._el[0][k];
22       }
23     }
24
```

```
25      for (j=0;j<n-1;j++)
26        for (i=j+1;i<n;i++)
27          _el[i][j]=Interval();
28
29      return (*this);
30    }
```

Przy czym metoda ta jest wywoływana w kontekście obiektu typu *IntMatrix*, który reprezentuje macierz układu A. Stąd wszystkie zmienne, a w szczególności tablica elementów `_el[][]` odnoszą się do zawartości tego właśnie obiektu a tym samym do macierzy układu A.
Metoda Gaussa polega na sprowadzeniu macierzy układu A do postaci trójkątnej górnej, przy czym każde przekształcenie wiersza macierzy jest również wykonywane na wierszach kolumny wyrazów wolnych b. Zadanie to realizują w realizują pętle numerowane w powyższym kodzie od 12 do 23. Cała iteracja przebiega wzdłuż głównej przekątnej macierzy układu (indeks k) od lewej w dół do prawej. Na pierwszym kroku jest sprawdzane, czy element centralny nie jest przedziałem zawierającym zero [9], i jeżeli nie jest to obliczany jest mnożnik dla bieżącego wiersza `mul`. Wewnętrzna pętla (18-19) oblicza nowe wartości elementów bieżącego wiersza począwszy od elementu centralnego w prawo do końca wiersza. Kolejna instrukcja dokonuje tego samego przekształcenia na wyrazach kolumny b.

W tym momencie warto zauważyć że nie wszystkie elementy macierzy układu są poddawane przekształceniom. Metoda Gaussa opisana w rozdziale 2.3 wymaga wykonywanie przekształceń na całych wierszach macierzy, przy czym mnożnik dobierany jest zawsze tak aby cała kolumna pod główną przekątną została wyzerowana. W tym przypadku jednak mamy do czynienia z macierzą przedziałów, zatem mając na uwadze fakt, że w ogólności dla przedziałów nie zachodzi $[a,b]-[a,b]=[0,0]$ możemy pominąć przekształcanie elementów, które z definicji metody mają być równe zero.
Aby jednak otrzymać macierz trójkątną górną implementacja metody przypisuje przedziały zerowe do wszystkich elementów poniżej głównej diagonali macierzy układu (25-27).

Wynikiem obliczeń powyższej funkcji jest przekształcona macierz układu równań liniowych w postaci trójkątnej górnej oraz wektor - kolumna wyrazów wolnych, również po przekształceniach. Pozostałe czynności sprowadzają się jedynie do odczytania wartości z układu i są wykonywane przez następujące

[9] W przypadku macierzy rzeczywistych warunek ten sprawdza czy element centralny nie jest równy 0

linie kodu.

```
1  int n = RowCnt;
2  int i,j;
3  Interval sum;
4
5    for(i=n-1;i>=0;i--)
6    {
7      sum=0.0;
8      for(j=i+1;j<n;j++)
9       sum=sum+_el[i][j]*b._el[0][j];
10
11     b._el[0][i]= (b._el[0][i]-sum)/_el[i][i];
12   }
13
14   b=b.Transpose();
15   return(b);
```

Jak łatwo zauważyć iteracja na tym etapie przebiega w odwrotnym kierunku (5-12) to znaczy od ostatniej kolumny i ostatniego wiersza wzdłuż głównej przekątnej. Obliczone wartości są zapisywane w wektorze b w kolejności od ostatniej do pierwszej, przy czym, jak wiadomo z przebiegu metody Gaussa-Jordana obliczanie każdej kolejnej niewiadomej wymaga użycia wartości niewiadomych obliczonych wcześniej.

Metoda Gaussa z częściowym wyborem elementu głównego

Wybór elementu centralnego w metodzie Gaussa ma szczególne znaczenie w przypadku implementacji tej metody w języku programowania. Wartości elementu centralnego zbyt bliskie zera mogą powodować trudne do oszacowania błędy zaokrągleń, ponieważ operacja dzielenia przez liczby bardzo małe, ze względu na sposób reprezentacji liczb w pamięci komputera, generuje duże błędy obliczeniowe. Dlatego zmodyfikowana zmodyfikowany algorytm Gaussa rozwiązywania układów równań liniowych wprowadza dodatkowe czynności optymalizujące metodę.
Omawiany wariant metody Gaussa dotyczy wyboru elementu głównego w kolumnie. W każdym i-tym kroku iteracji na początku są wykonywane następujące linie kodu.

```
1 ...
2       max=k;
3       for (i=k;i<n;i++)
4       {
5         if (_el[i][k].iabs()._s > _el[max][k].iabs()._s)
6          max=i;
7       }
8
9       if (_el[max][k]==0.0) return;
10
11      if (k!=max)
12      {
13        for (j=k;j<n;j++)
14        {
15          mul=_el[k][j];
16          _el[k][j]=_el[max][j];
17          _el[max][j]=mul;
18        }
19
20        mul=b._el[0][k];
21        b._el[0][k]=b._el[0][max];
22        b._el[0][max]=mul;
23      }
24 ...
```

Na początku metoda wyszukuje największy co do bezwzględnej wartości element w bieżącej kolumnie (3-7), to znaczy w kolumnie w której znajduje się element centralny, przy czym poszukiwania obejmują tylko elementy poniżej diagonali. Następnie jest sprawdzany warunek osobliwości macierzy (9). Po znalezieniu maksymalnego elementu centralnego w kolumnie następuje operacja zamiany wierszy, przy czym wiadomo, że w momencie zamiany wierszy w macierzy układu równań liniowych zmienia się kolejność owych równań, należy zatem również zamienić kolejność elementów w kolumnie wyrazów wolnych b, za co odpowiedzialne ostatnie linie przedstawionego kodu.

Metoda Gaussa z pełnym wyborem elementu głównego

Metoda Gaussa z pełnym wyborem elementu centralnego polega na wybieraniu elementu głównego w całej podmacierzy rozpatrywanego układu. Pozwala to na uniknięcie błędów zaokrągleń omówionych w poprzednim paragrafie, aczkolwiek kosztem tego zwiększa ilość wykonywanych operacji, a tym samym czas obliczeń całej metody.

```
1 ...
2         maxc=k;
3         maxr=k;
4         for (i=k;i<n;i++)
5           for (j=k;j<n;j++)
6           {
7             if (_el[i][j].iabs()._s > _el[maxr][maxc].iabs()._s)
8             { maxr=i; maxc=j; }
9           }
10
11          if (_el[maxr][maxc]==0.0) return;
12
13          if (k!=maxr)
14          {
15            for (j=k;j<n;j++)
16            {
17              mul=_el[k][j];
18              _el[k][j]=_el[maxr][j];
19              _el[maxr][j]=mul;
20            }
21
22            mul=b._el[0][k];
23            b._el[0][k]=b._el[0][maxr];
24            b._el[0][maxr]=mul;
25          }
26
27          if (k!=maxc)
28          {
29            for (i=0;i<n;i++)
30            {
31              mul=this->_el[i][k];
32              _el[i][k]=_el[i][maxc];
33              _el[i][maxc]=mul;
34            }
35 ...
36          }
```

W każdym i-tym kroku iteracji element centralny jest poszukiwany w dolnej lewej podmacierzy układu (linie 4-9). Następnie zamieniane są miejscami wiersze macierzy oraz kolumny wyrazów wolnych (13-25) oraz kolumny macierzy (27-34). Po tych operacjach wykonywane są opisane wcześniej przekształcenia układu równań.

4.4 Działania arytmetyczne

W klasie *Interval* zostały zaimplementowane działania arytmetyczne w postaci przeładowanych operatorów arytmetycznych. Podobnie jak poprzednio zmienne, które nie mają przypisanego obiektu (tzn. $_i$ oraz $_s$) należą do domyślnego obiektu reprezentowanego przez *this* względem którego wykonywane są obliczenia.

```
1  /*DODAWANIE*/
2  Interval operator +(Interval y)
3    {
4      Interval* result = new Interval();
5      Rdown();
6      result->_i = _i + y._i;
7      Rup();
8      result->_s = _s + y._s;
9      Rnear();
10     return(*result);
11   };
12
13 /*ODEJMOWANIE*/
14   Interval operator -( Interval y)
15   {
16     Interval* result = new Interval();
17     Rdown();
18     result->_i = _i - y._s;
19     Rup();
20     result->_s = _s - y._i;
21     Rnear();
22     return(*result);
23   };
24
25 /*MNOZENIE*/
26   Interval operator *( Interval y)
27   {
28     Interval* result = new Interval();
29     Rdown();
30     result->_i = minimum(_i*y._i , _i*y._s ,
31                          _s*y._i , _s*y._s);
32     Rup();
33     result->_s = maximum(_i*y._i , _i*y._s ,
34                          _s*y._i , _s*y._s);
```

```
35      Rnear();
36      return(*result);
37    };
38
39  /*DZIELENIE*/
40    Interval operator /(Interval y)
41    {
42      if (y.iszero())
43        return(Interval(0.0/0.0));
44
45        Rdown();
46        result->_i=minimum(_i/y._i,_i/y._s,_s/y._i,_s/y._s);
47
48        Rup();
49        result->_s=maximum(_i/y._i,_i/y._s,_s/y._i,_s/y._s);
50
51        Rnear();
52        return (*result);
53    }
```

Funkcje *Rdown()*, *Rup()* oraz *Rnear()* są odpowiedzialne za sterowanie zaokrąglaniami koprocesora, odpowiednio w dół, w górę i do najbliższej. Omówione zostaną w następnym rozdziale. Funkcje minimum i maximum obliczają najmniejszą i największą wartość spośród czterech wprowadzonych typu *LDouble*;

4.5 Sterowanie zaokrągleniami

Sterowanie zaokrągleniami to kwestia ściśle związana z implementacją arytmetyki przedziałowej. Fakt ten wynika z samej definicji przedziału, który ma obejmować swoim zakresem dokładne rozwiązanie obliczanych wyrażeń. Zatem gdy obliczenia w arytmetyce rzeczywistej obarczone są błędami zaokrągleń, wynik takich obliczeń może być większy lub mniejszy od wartości dokładnej. Mniejszy gdy obliczenia zostały zaokrąglone w dół i większy gdy w górę. Otrzymując taki wynik, nie ma możliwości oszacowania jak wielkie były błędy zaokrągleń, ani tym bardziej w którą stronę wynik został zaokrąglony.

Przedział natomiast, z samej definicji, zawiera w sobie cały zakres możliwych liczb powstałych w wyniku obliczeń danego wyrażenia z uwzględnieniem ewentualnych błędów zaokrągleń. Pozwala zatem oszacować wynik dokładny z pewną liczbą cyfr znaczących, co nie ma miejsca w przypadku zwykłej arytmetyki zmiennopozycyjnej. Aby osiągnąć ten cel należy wymuszać tryby zaokrągleń w koprocesorze. Dolny kraniec przedziału należy obliczać przy włączonym zaokrągleniu w dół, natomiast górny kraniec przedziału przy zaokrągleniu w górę.

W programie *IntComp* zadanie to spełniają funkcje *Rdown()* oraz *Rup()*, które włączają zaokrąglenie odpowiednio w dól i w górę ustawiając słowo kontrolne koprocesora. Stosowanie tych funkcji przedstawiono w poprzednim rozdziale na przykładzie implementacji operatorów arytmetycznych.

Sterowanie zaokrągleniami ma też zastosowanie w przypadku konwersji liczb rzeczywistych na krańce przedziałów. Należy bowiem wiedzieć, że liczba, którą wprowadzamy do programu nie jest dokładnie zapisana w pamięci komputera. Jest natomiast konwertowana do swojego odpowiednika maszynowego. Zakładając, że użytkownik wprowadził liczbę rzeczywistą x, jej reprezentację maszynową $rd(x)$ możemy zapisać jako

$$x \neq rd(x) = x(1 + \varepsilon),$$

gdzie $|\varepsilon| \leqslant eps = 2^{-t}$, natomiast t oznacza ilość bitów przeznaczonych do zapisu mantysy (w typie `long double` 64 bity).

Rozpatrzymy przypadek, kiedy podana została pojedyncza liczba rzeczywista typu `long double`, którą należy skonwertować na krańce przedziału. Jest to ważne zagadnienie, ponieważ wprowadzanie danych do programu nie

jest jednoznaczne z przypisaniem tej samej wartości do zmiennej. Wpisana przez użytkownika wartość zostaje wczytana do zmiennej, jednak jeśli wartość ta nie może być reprezentowana dokładnie, wtedy wartość zmiennej jest obarczona błędem zaokrągleń, w rezultacie czego zmienna reprezentująca liczbę może mieć wartość inną niż ta, którą wpisał użytkownik.

Jedno rozwiązanie tego problemu nasuwa się na myśl od razu. Można bowiem od wartości liczby wczytanej do pamięci komputera odpowiednio odjąć i dodać dokładność maszynową *eps*, otrzymując dolny i górny kres przedziału. Jednak to rozwiązanie w wielu przypadkach deje nam zbyt duże oszacowanie. Przedział tak wygenerowany jest zbyt szeroki, co po wielokrotnych obliczeniach może doprowadzić do całkowicie nieprzydatnych wyników.

Inne rozwiązanie powyższego problemu polega na odczytaniu z wprowadzonej liczby jej postaci ułamkowej. Postaci ułamkowej, to znaczy licznika i mianownika, które po podzieleniu dają pierwotną liczbę. Ze względu na to, że wprowadzana liczba zapisana jest w postaci ciągu znaków, rozdzielenie jej na licznik i mianownik nie przysparza większych problemów. Tak wyznaczony ułamek można zamienić na dwie liczby całkowite, przy czym konwersja taka nie powoduje już błędów zaokrągleń. Mając licznik i mianownik w postaci zmiennych typu całkowitego można je przez siebie podzielić otrzymując pierwotnie wprowadzoną liczbę oraz przypisać do dolnego i górnego krańca przedziału z włączonymi zaokrągleniami odpowiednio w dół i w górę.

Dla przykładu załóżmy, że użytkownik wprowadził liczbę 0.1, która nie jest dokładnie reprezentowana w procesorze. Rozdzielamy ją na licznik i mianownik i otrzymujemy licznik równy 1, a mianownik równy 10. Teraz dzieląc te dwie liczby przez siebie przypisujemy wynik do krańców przedziału, pamiętając o odpowiednim sterowaniu zaokrągleniami.

```
1 Rdown();
2 przedzial._i = licznik/mianownik;
3 Rup();
4 przedzial._s = licznik/mianownik;
5 Rnear();
```

Wynikiem tej operacji są krańce przedziałów, które zawierają liczby reprezentowane dokładnie w pamięci komputera. Przedział ten zawiera w sobie liczbę, która została wprowadzona do programu.

Pewną metodę rozwiązania problemu konwertowania wprowadzonej liczby na przedział zaprezentował dr hab. Andrzej Marciniak w szeregu swoich publikacji na temat arytmetyki przedziałowej. Metoda ta opiera się na stwier-

dzeniu, że jeżeli liczba nie jest liczbą maszynową [10], to może być reprezentowana w postaci przedziału o końcach będących dwiema sąsiednimi liczbami maszynowymi.

Realizacja tego algorytmu jest trudniejsza i wymaga większej liczby operacji oraz operowania na długich łańcuchach znaków a także reprezentacji wewnętrznej liczb. Zakładając, że użytkownik wprowadził liczbę a typu `long double`, niech sa oznacza ciąg znaków wprowadzonych przez użytkownika, natomiast zmienna x zawiera wartość maszynową liczby a odczytaną z wprowadzonego ciągu znaków. Zmienna sx typu łańcucha długiego zawiera stałoprzecinkową, dziesiętną wartość reprezentacji maszynowej liczby a, czyli wartość zmiennej x odczytaną z jej wewnętrznej postaci.

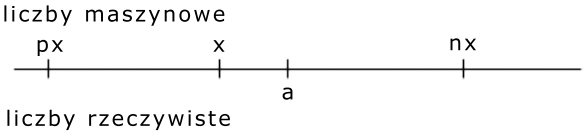
Mając te zmienne można przystąpić do porównywania ciągów znaków. Jeżeli $sa = sx$ to oznacza, że liczba a jest reprezentowana dokładnie w pamięci komputera, zatem za oba końce przedziału można przyjąć wartość zmiennej x. Otrzymujemy przedział $[x, x]$, czyli przedział o szerokości równej zero.

Jeżeli $sa \neq sx$ wtedy trzeba określić dwie kolejne liczby, odpowiednio następną po x liczbę maszynową oznaczając przez nx oraz poprzednią liczbę maszynową oznaczając px. Otrzymać je można poprzez odpowiednio dodanie i odjęcie od x zmienną *eps*, której wartość wyznacza wewnętrzna (znormalizowana) postać mantysy równa 1 i cecha mniejsza o 63 od wartości cechy w wewnętrznej reprezentacji wartości zmiennej x.

Teraz już wystarczy tylko sprawdzić, która z następujących relacji jest prawdziwa relacje $sa < sx$ lub $sx < sa$. Jeśli prawdziwa jest pierwsza para nierówności, to przedziałem zmiennopozycyjnym reprezentującym liczbę a jest przedział $[px, x]$, a w drugim przypadku – przedział $[x, nx]$. W przypadku liczby ujemnej, gdy prawdziwa jest pierwsza nierówność, to poszukiwanym przedziałem jest $[x, nx]$, a gdy druga – przedział $[px, x]$.

Jeden z przypadków przedstawiony jest na poniższym rysunku. Liczba a to liczba wprowadzona przez użytkownika. Widać, że nie odpowiada jej żadna liczba maszynowa, zatem wynikowy przedział będzie zawierał najbliższe liczbie a liczby maszynowe. W tym przypadku będzie to $[x, nx]$.

[10] To znaczy nie może być dokładnie reprezentowana w pamięci komputera.



Zagadnienie konwertowania liczb na ich odpowiednie przedziały maszynowe ma duże znaczenie. Wpływa ono w dużym stopniu na dokładność obliczeń, ponieważ w przypadku, gdy źle reprezentowana liczba może zostać skonwertowana na przedział, który nie obejmuje jej swoim zakresem, prowadząc w ciągu dalszych obliczeń do potęgowania się tej niedokładności i do nieprzewidywalnych wyników.

Ze względu na trudność implementacji metody przedstawionej przez Pana dr. hab. Marciniaka, w programie *IntComp* zaimplementowana została druga metoda konwersji liczb. Generuje ona nieco szersze przedziały, jednak dla potrzeb tej pracy oraz badań nad arytmetyką przedziałową w zupełności wystarcza. Fragment kodu realizującego to zadanie przedstawiony został poniżej.

```
1 ...
2 LDouble licz,mian;
3 ToFrac(sx,licz,mian);
4 Rdown();
5 _i=(long double)(licz/mian);
6 ...
7 LDouble licz,mian;
8 ToFrac(sy,licz,mian);
9 Rup();
10 _s=(long double)(licz/mian);
11 ...
```

Metoda *ToFrac* zamienia ciąg znaków reprezentujący wprowadzoną liczbę na dwie zmienne zawierające odpowiednio licznik oraz mianownik w postaci liczb całkowitych.

5 Badania

5.1 Przykłady

Rozważmy kilka układów równań liniowych. Przykłady zaczerpnięte zostały z książki [3].

1. Dany jest układ równań

$$\begin{array}{rcrcrcl} 3x_1 & + & x_2 & + & 6x_3 & = & 2 \\ 2x_1 & + & x_2 & + & 3x_3 & = & 7 \\ x_1 & + & x_2 & + & x_3 & = & 4 \end{array}.$$

Spróbujmy rozwiązać ten układ stosując arytmetykę rzeczywistą

```
> <3,1,6;2,1,3;1,1,1>solvef<2;7;4>

  Result =
          +1.90000000000000000E+0001
          -7.00000000000000000E+0000
          -8.00000000000000000E+0000

  dimension: 3x1
```

Podobne wyniki otrzymujemy przy wykorzystaniu arytmetyki przedziałowej

```
> <[3][1][6];[2][1][3];[1][1][1]>solvef<[2];[7];[4]>

  Result =
  [+1.90000000000000000E+0001,+1.90000000000000000E+0001]
  [-7.00000000000000000E+0000,-7.00000000000000000E+0000]
  [-8.00000000000000000E+0000,-8.00000000000000000E+0000]

  dimension: 3x1
```

Współczynnik uwarunkowania macierzy A w normie nieskończonej wynosi 86.6666666666666667.

2. Dany jest układ z macierzą trójdiagonalną.

$$A = \begin{bmatrix} -1.276 & 2.5473 & 0 & 0 \\ 0 & 2.756 & 0.827 & 0 \\ 0 & 4.72 & 6.98376 & 0.0003 \\ 0 & 0 & 8.9174 & 0 \end{bmatrix},$$

oraz wektorem wyrazów wolnych

$$b = \begin{bmatrix} 1.2 \\ 7 \\ 3.333 \\ 0 \end{bmatrix}.$$

Przy użyciu arytmetyki rzeczywistej oraz metody Gaussa-Jordana bez wyboru elementu głównego otrzymujemy następujące wyniki

```
> <-1.276,2.5 ... 0,8.9174,0>solve<1.2;7;3.333;0>

 Result =

         +4.13003148445568745E+0000
         +2.53991291727140784E+0000
         -1.55792473250909726E-0019
         -2.88512965650701500E+0004

 dimension: 4x1
```

Wskaźnik uwarunkowania macierzy układu jest duży zatem można się spodziewać, że obliczone wyniki nie są dokładne. Zobaczmy jakie wyniki otrzymamy z wykonując obliczenia z wykorzystaniem arytmetyki przedziałowej

```
Result =
  [+4.13003148445568744E+0000,+4.13003148445568745E+0000]
  [+2.53991291727140784E+0000,+2.53991291727140784E+0000]
  [+0.00000000000000000E+0000,+0.00000000000000000E+0000]
  [-2.88512965650701500E+0004,-2.88512965650701500E+0004]
```

```
width%
  +6.0715321659188248300E-0018
  +1.9515639104739079800E-0018
  +0.0000000000000000000E+0000
  +1.7763568394002504600E-0014
```

Punkty środkowe przedziałów wynikowych wynoszą

```
> mid%

 Result =
 +4.13003148445568745E+0000
 +2.53991291727140784E+0000
 +0.00000000000000000E+0000
 -2.88512965650701500E+0004
```

Zatem można przypuszczać, że otrzymane wyniki wyznaczają wartość dokładną. Spróbujmy zastosować algorytm Gaussa-Jordana z pełnym wyborem elementu centralnego w arytmetyce rzeczywistej.

```
> <-1.276,...,8.9174,0>solvef<1.2;7;3.333;0>

Result =
      +4.13003148445568745E+0000
      +2.53991291727140784E+0000
      +0.00000000000000000E+0000
      -2.88512965650701500E+0004

      dimension: 4x1
```

3. Układ równań z macierzą trójkątną górną.

$$\begin{bmatrix} 2 & -1 & 3 \\ 0 & 1 & -1 \\ 0 & 0 & 3 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1.5 \\ 1 \end{bmatrix}.$$

W arytmetyce rzeczywistej mamy

```
> <2,-1,3;0,1,-1;0,0,3>solvef<2;1.5;1>

Result =
+1.41666666666666667E+0000
+1.83333333333333333E+0000
+3.33333333333333333E-0001

dimension: 3x1
```

natomiast w arytmetyce przedziałowej

```
> <[2][-1][3];[0][1][-1];[0][0][3]>solvef<[2];[1.5];[1]>

Result =
  [+1.41666666666666667E+0000,+1.41666666666666667E+0000]
  [+1.83333333333333333E+0000,+1.83333333333333333E+0000]
  [+3.33333333333333333E-0001,+3.33333333333333334E-0001]

dimension: 3x1

width%
  +4.33680868994201774 00E-0019
  +6.50521303491302660 00E-0019
  +5.14996031930614606 00E-0019
```

4. Rozważmy układ postaci

$$\begin{bmatrix} 1 & 4.5 & 1.2 & 2.1 & 2.52 \\ 5.5 & 3.3 & 9.9 & 1.848 & 2.31 \\ 2.2 & 1.485 & 4.752 & 9.24 & 1.188 \\ 7.15 & 5.148 & 1.716 & 3.432 & 4.5045 \\ 2.002 & 1.5015 & 5.148 & 1.05105 & 1.4014 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 2.1 \\ 1.98 \\ 1.0395 \\ 4.004 \\ 1.26126 \end{bmatrix}.$$

Obliczenia w arytmetyce rzeczywistej dają następujące wyniki

```
Result =

   -7.71054612831234935E-0003
   -1.00755886806752288E-0001
   +8.15052000454983006E-0004
   -5.70634656152984222E-0004
   +1.01640170918098132E+0000

 dimension: 5x1
```

Natomiast po zastosowaniu arytmetyki przedziałowej mamy

```
Result =

[-7.71054612831235283E-0003,-7.71054612831234581E-0003]
[-1.00755886806752290E-0001,-1.00755886806752286E-0001]
[+8.15052000454979611E-0004,+8.15052000454986346E-0004]
[-5.70634656152984473E-0004,-5.70634656152983969E-0004]
[+1.01640170918098132E+0000,+1.01640170918098133E+0000]

width%
  1.28554190404785162000E-0017
  6.80336863234654032000E-0018
  1.23424876513568340000E-0017
  1.01665129494197398000E-0018
  1.14925430283463470000E-0017
```

5.2 Macierz źle uwarunkowana

Rozważmy układ dwóch równań liniowych z dwiema niewiadomymi o macierzy:

$$A = \begin{bmatrix} 25.01 & -35.99 \\ 15.98 & -23.01 \end{bmatrix},$$

oraz kolumnie wyrazów wolnych

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$

Rozwiązanie tego układu równań obliczone za pomocą systemu algebry komputerowej *Maple* z dokładnością do 32 cyfr po przecinku wynosi:

$$\begin{bmatrix} 163.93442622950819672131147541012 \\ 113.89274798555154209502639622137 \end{bmatrix}.$$

Spróbujemy rozwiązać ten układ równań przy użyciu standardowej arytmetyki rzeczywistej oraz arytmetyki przedziałowej. Przy pomocy programu *IntComp* sprawdźmy jaki jest wskaźnik uwarunkowania [11] macierzy A.

```
> cond<25.01,-35.99;15.98,-23.01>
  Result =
           +2.77799391497638221E+0004

  dimension: 1x1
```

Wskaźnik uwarunkowania jest duży, można się zatem spodziewać, że wyniki rozwiązania układu równań z taką macierzą będą obarczone dużym błędem.
Spróbujmy zatem rozwiązać powyższy układ równań. Zastosujemy do tego celu metody *solve*, *solvep* oraz *solvef*, które oznaczają odpowiednio metody Gaussa-Jordana rozwiązywania układów równań liniowych bez wyboru elementu centralnego, z wyborem w kolumnie oraz z pełnym wyborem. Obliczenia wykonamy w arytmetyce rzeczywistej.

```
> <25.01,-35.99;15.98,-23.01>solve<1;-1>
  Result =
```

[11] Polecenie *cond* oblicza wskaźnik uwarunkowania macierzy korzystając z normy $\|\cdot\|_\infty$, czyli maksimum z sum wierszy macierzy.

```
           +1.63934426229508209E+0002
           +1.13892747985551551E+0002
  dimension: 2x1

> <25.01,-35.99;15.98,-23.01>solvep<1;-1>
  Result =
          +1.63934426229508209E+0002
          +1.13892747985551551E+0002
  dimension: 2x1

> <25.01,-35.99;15.98,-23.01>solvef<1;-1>
  Result =
          +1.63934426229508193E+0002
          +1.13892747985551540E+0002
  dimension: 2x1
```

Mając takie wyniki, co możemy o nich powiedzieć? Wiemy tylko, że macierz układu jest źle uwarunkowana, co może być przyczyną niedokładności wyników. Czy możemy na podstawie tych obliczeń podać ilość cyfr dokładnych wyniku, to znaczy ilość cyfr które są identyczne z rozwiązaniem dokładnym?

Zobaczmy teraz na rozwiązania tego samego układu równań w arytmetyce przedziałowej:

```
> <[25.01][-35.99];[15.98][-23.01]>solve<[1][-1]>

  Result =
    [+1.63934426229508170E+0002,+1.63934426229508209E+0002]
    [+1.13892747985551523E+0002,+1.13892747985551551E+0002]

 dimension: 2x1

> <[25.01][-35.99];[15.98][-23.01]>solvep<[1][-1]>

  Result =
    [+1.63934426229508170E+0002,+1.63934426229508209E+0002]
    [+1.13892747985551523E+0002,+1.13892747985551551E+0002]

  dimension: 2x1
```

```
> <[25.01][-35.99];[15.98][-23.01]>solvef<[1][-1]>

  Result =
    [+163.934426229508164,+163.934426229508279]
    [+113.892747985551519,+113.892747985551600]

  dimension: 2x1
```

Otrzymaliśmy przedziały, które zawierają w sobie rozwiązania dokładne. Zatem z jaką dokładnością możemy podać wynik dokładny? Zobaczmy na szerokość przedziałów wynikowych.

```
> width(%)

   +1.13783982236270731E-0013
   +7.91103293984463107E-0014

 dimension: 2x1
```

Otrzymaliśmy rozwiązanie z dokładnością do czternastu miejsc po przecinku. W przeciwieństwie do obliczeń przy pomocy arytmetyki rzeczywistej, teraz możemy być pewni, że dokładne rozwiązanie zadanego układu równań leży w granicach otrzymanego przedziału.

5.3 Macierz Hilberta

Rozpatrzmy teraz układ równań liniowych, w którym macierzą układu jest macierz Hilberta stopnia $n = 4$.
Elementy macierzy Hilberta wyznacza się ze wzoru:

$$a_{ij} = \frac{1}{i + j - 1}.$$

Wskaźnik uwarunkowania dla macierzy Hilberta stopnia 4 wynosi 28375.
Rozwiązanie dokładne tego układu równań wynosi odpowiednio

$$-64, 900, -2520, 1820.$$

W arytmetyce rzeczywistej z zastosowaniem metod *solve*,*solvep* oraz *solvef* oraz wektorem wyrazów wolnych $b = [1, 2, 3, 4]$ otrzymujemy następujące rozwiązanie

```
  -6.40000000000003487E+0001
  +9.00000000000003939E+0002
  -2.52000000000000954E+0003
  +1.82000000000000623E+0003
```

natomiast przy tych samych parametrach w arytmetyce przedziałowej otrzymujemy

```
Result =
  [-6.40000000000428386E+0001,-6.39999999999566563E+0001]
  [+8.99999999999971710E+0002,+9.00000000000022487E+0002]
  [-2.52000000000005418E+0003,-2.51999999999993164E+0003]
  [+1.81999999999995543E+0003,+1.82000000000003524E+0003]

width%
   +8.61822002651280172E-0011
   +5.07756614531729155E-0011
   +1.22524212997632275E-0010
   +7.98012766978217769E-0011
```

Jak widać, wyniki nie różnią się specjalnie od rozwiązań uzyskanych w arytmetyce rzeczywistej. Jednak zastosowanie arytmetyki przedziałowej umożliwia oszacowanie wyników dokładnych z bardzo dużą dokładnością, czego nie można osiągnąć mając tylko do dyspozycji wyniki obliczeń rzeczywistych. Jest to szczególnie ważne w przypadku zadań co do których wiemy, że dane wejściowe powodują duże błędy zaokrągleń.

5.4 Układ Boothroyda-Dekkera

Rozpatrzmy układ równań liniowych

$$Ax = b,$$

gdzie A jest macierzą kwadratową stopnia n, której współczynniki obliczamy ze wzoru:

$$a_{ij} = \begin{pmatrix} n+i-1 \\ i-1 \end{pmatrix} \cdot \begin{pmatrix} n-1 \\ n-j \end{pmatrix} \cdot \frac{n}{i+j-1},$$

gdzie $i, j = 1, 2, \ldots, n$ oznaczają odpowiednio numer wiersza oraz kolumny w której znajduje się obliczany element. Wektor wyrazów wolnych jest postaci $b_i = i$
Dokładne rozwiązanie tego układu równań wynosi:

$$x_i = (-1)^i \cdot (i-1)\,.$$

Spróbujmy rozwiązać układ Boothroyda-Dekkera stopnia $n = 10$. Aby zmniejszyć ilość potrzebnych obliczeń przekształcamy wzór na elementy macierzy A do postaci

$$a_{ij} = \frac{(n+i-1)!}{(i-1)!(j-1)!(n-1)!(i+j-1)}.$$

W arytmetyce rzeczywistej dla metod Gaussa bez wybory elementu centralnego, z wyborem w kolumnie oraz z pełnym wyborem otrzymujemy odpowiednio:

```
> <...>solve<...>

  Result =
+8.00713479895409641E-0012
+9.99999999923896326E-0001
+1.99999999959709273E+0000
+2.99999999843556754E+0000
-3.99999999503540218E+0000
+4.99999998636858708E+0000
-5.99999996647478066E+0000
+6.99999992442874729E+0000
-7.99999984131976395E+0000
+8.99999968592229004E+0000
```

```
> <...>solvep<...>

  Result =
-1.36255892676783844E-0011
+1.00000000012953378E+0000
-2.00000000068433323E+0000
+3.00000000264914538E+0000
-4.00000000837900964E+0000
+5.00000002293014742E+0000
-6.00000005621465129E+0000
+7.00000012633545008E+0000
-8.00000026452642051E+0000
+9.00000052221326970E+0000

 > <...>solvef<...>

   Result =
-4.54594910909048832E-0012
+1.00000000004164493E+0000
-2.00000000021290685E+0000
+3.00000000080081331E+0000
-4.00000000246979090E+0000
+5.00000000661035395E+0000
-6.00000001588997636E+0000
+7.00000003509055063E+0000
-8.00000007233029646E+0000
+9.00000014078778974E+0000
```

Jak widać wyniki są obarczone dość dużym błędem. Po zastosowaniu arytmetyki przedziałowej mamy

```
> <...>solve<...>

Result =
[-3.27207783561308336E+0004,+3.27207783489015953E+0004]
[-4.48462560571335603E+0003,+4.48662567747363546E+0003]
[-7.26560121348878347E+0002,+7.22559729025903210E+0002]
[-1.33472227430271885E+0002,+1.39473788953086407E+0002]
[-3.43294377031307015E+0001,+2.63243839152195931E+0001]
[-3.13806339353960176E+0000,+1.31521636123837586E+0001]
```

```
[-8.7659429677678739E+0000,-3.2691991280077956E+0000]
[+5.8016808864436944E+0000,+8.2784282212613489E+0000]
[-8.8952067819809156E+0000,-7.2746226274186575E+0000]
[+7.5522882018074055E+0000,+1.0786658180643958E+0001]


> <...>solvep<...>

 Result =
[-3.2720778356130834E+0004,+3.2720778348901595E+0004]
[-4.4846256057133560E+0003,+4.4866256774736355E+0003]
[-7.2656012134887835E+0002,+7.2255972902590321E+0002]
[-1.3347222743027189E+0002,+1.3947378895308641E+0002]
[-3.4329437703130702E+0001,+2.6324383915219593E+0001]
[-3.1380633935396018E+0000,+1.3152163612383759E+0001]
[-8.7659429677678739E+0000,-3.2691991280077956E+0000]
[+5.8016808864436944E+0000,+8.2784282212613489E+0000]
[-8.8952067819809156E+0000,-7.2746226274186575E+0000]
[+7.5522882018074055E+0000,+1.0786658180643958E+0001]

> <...>solvef<...>

 Result =
[-7.6613406904252796E-0004,+7.6613401629849860E-0004]
[+9.9156667571488446E-0001,+1.0084333247258035E+0000]
[-2.0090367526478555E+0000,-1.9909632494631581E+0000]
[+2.9970219451429947E+0000,+3.0029780624166725E+0000]
[-4.0230968753480276E+0000,-3.9769031470832602E+0000]
[+4.9939182158652588E+0000,+5.0060818423211048E+0000]
[-6.0190047266035052E+0000,-5.9809954096804212E+0000]
[+6.9989023820944201E+0000,+7.0010979123419069E+0000]
[-8.0022520592437654E+0000,-7.9977485363686244E+0000]
[+8.9956338389008035E+0000,+9.0043673016956880E+0000]

width%
 +3.2040683394019542E-0003
 +3.5269101412284766E-0002
 +3.7792718736351955E-0002
 +1.2454617459977211E-0002
 +9.6593601726406544E-0002
 +2.5434810862636318E-0002
```

```
+7.94795339403736724E-0002
+4.59072873703953886E-0003
+9.41718522533204058E-0003
+1.82632746296559545E-0002
```

6 Wnioski

Główną zaletą stosowania arytmetyki przedziałowej jest fakt że dzięki niej otrzymujemy dość wąskie oszacowanie dokładnego wyniku. Szczególnie ważne jest to, gdy wiemy, że wprowadzane dane mogą być obarczone błędem, są to dane pomiarowe z urządzeń o określonej dokładności, bądź odczytywane, przez jakże podatne na pomyłki, ludzkie oko. Używając w takich przypadkach standardowej arytmetyki rzeczywistej nie możemy być pewni że otrzymamy dokładny wynik. Okazuje się jednak, że wielu ludzi, często nawet inżynierów, bezgranicznie zawierza niezawodności i precyzji komputerów i uważa takie wyniki za absolutnie dokładne. O tragicznych skutkach takich błędów ludzi (bo komputerów nie można obwiniać za błędy, w które je sam człowiek zaopatrzył) słyszeliśmy niejednokrotnie.

Co zatem daje zastosowanie arytmetyki przedziałowej? Wyobraźmy sobie praktyczny przypadek. Urządzenia pomiarowe pewnego samolotu odczytują jego wysokość z pewną dokładnością. Na podstawie tych danych komputer steruje autopilotem, który z kolei utrzymuje samolot w wyznaczonym dla siebie tunelu powietrznym. Jednak awaria urządzenia pomiarowego sprawia, że odczytany wynik, po wielokrotnych obliczeniach zostaje drastycznie zniekształcony. Komputer nie wiedząc o tym koryguje współrzędne dla autopilota. Samolot zmienia kurs i leci w nieprzewidywalnym kierunku. Więcej nie trzeba sobie wyobrażać. Błędy zaokrągleń mogą spowodować, że błąd wyniku operacji arytmetycznych będzie rósł bardzo szybko.

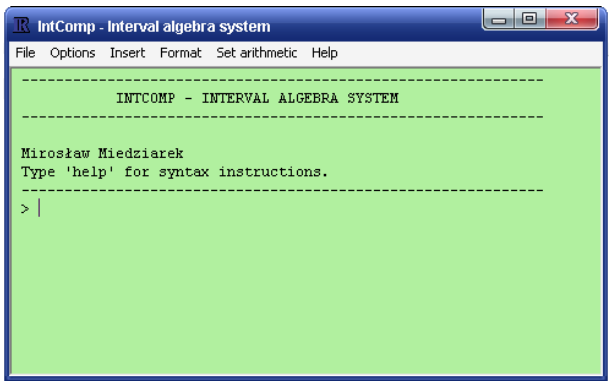
Co może tutaj pomóc zastosowanie arytmetyki przedziałowej? Otóż trzeba wiedzieć, o czym przekonaliśmy się rozwiązując układy równań liniowych, że szerokość przedziału będącego wynikiem operacji arytmetycznych rośnie wraz ze wzrostem błędów zaokrągleń. Znaczy to, że im więcej błędów zaokrągleń powodują dane wprowadzanie do systemu, tym szerszy jest ostateczny wynik obliczeń. W przypadku arytmetyki rzeczywistej wynik mógł rosnąć w górę lub w dół, przy czym nie wiadomo było jak daleko jest od wartości dokładnej. W przypadku arytmetyki przedziałowej kresy przedziału wynikowego rosną proporcjonalnie w przeciwnych kierunkach, co wynika z implementacji samej arytmetyki i jej działań. Zatem aby uniknąć katastrofy w omówionym wyżej przykładzie, można by zaimplementować mechanizm, który na zbyt szeroki przedział wynikowy spowodowany niedokładnymi danymi oraz błędami zaokrągleń reagowałby odpowiednio (np. informował pilota o błędzie aby ten wyłączył autopilota).

Podobnych przykładów można by znaleźć wiele. W każdym wypadku trzeba mieć świadomość, że komputer jest tylko maszyną stworzoną przez człowieka i jego skuteczność jest ograniczona przez wyobraźnię swojego stwórcy. W przypadku obliczeń matematycznych na liczbach rzeczywistych niebagatelne znaczenie ma implementacja samej arytmetyki a także całego oprogramowania, bo od tego głównie zależy dokładność obliczeń wykonywanych przez komputer.

7 Instrukcja obsługi programu

Program *IntComp* został zaprojektowany i zaimplementowany od podstaw dla celów niniejszej pracy. Od podstaw znaczy, że wszystkie typy danych, klasy, implementacja działań arytmetycznych oraz algorytmów rozwiązywania układów równań liniowych zostały napisane w całości przez autora niniejszej pracy.

Program *Intcomp* zawiera zaawansowane parsery składni matematycznej, co pozwala na wykonywanie obliczeń dowolnych ciągów wyrażeń matematycznych. Wzorem dla programu były znane systemy algebry komputerowej takie jak *Matlab*, *Derive* czy *Maple*. Stąd wprowadzanie danych w programie *IntComp* odbywa się w lini komend.



Rysunek 1: Główne okno programu *IntComp*

Wprowadzanie danych

W programie *IntComp* dane wprowadzamy w linii komend, która zaczyna się zawsze znakiem zachęty ». W celu obliczenia wyrażenia matematycznego wciskamy klawisz *ENTER*.

Liczby możemy wprowadzać zarówno w postaci stałopozycyjnej jak i zmiennopozycyjnej, przy czym obowiązującym separatorem dziesiętnym jest kropka. Przedziały w prowadzamy w nawiasach kwadratowych, podając jedną liczbą, bądź oba końce przedziału, przy czym w drugim przypadku liczby oddzielamy przecinkiem:

$$[1.2, 3.15e-1],$$
$$[52.6e-1].$$

Macierze i wektory wprowadzamy wpisując elementy wierszami i oznaczając koniec wiersza średnikiem. Na przykład wprowadzenie ciągu:

$$< [1.3, 2][3, 4.4]; [5, 6][6, 7] >$$

oznacza macierz przedziałów

$$\begin{bmatrix} [1.3, 2] & [3, 4.4] \\ [5, 6] & [6, 7] \end{bmatrix}.$$

W arytmetyce rzeczywistej elementy każdego wiersza oddzielamy dodatkowo przecinkiem.
Na przykład macierz:

$$\begin{bmatrix} 1.3 & 2 & 4.4 \\ 5 & 6.3E-2 & 7 \end{bmatrix}$$

wprowadzimy ciągiem znaków

$$< 1.3, 2, 4.4; 5, 6.3E-2, 7 >.$$

Przy czym wszelkie nieprawidłowości we wprowadzonym wyrażeniu zostaną wykryte przez parser.

Funkcje

Oprócz standardowych działań arytmetycznych (+,−,∗,/) następujące funkcje są wspólne w przypadku stosowania obu arytmetyk:

- $abs()$ - wartość bezwzględna,
- $x \hat{} n$ - potęga naturalna,
- $sqrt()$ - pierwiastek kwadratowy,
- $exp()$ - eksponent,
- $log()$ - logarytm naturalny,
- $sin()$ - sinus,
- $cos()$ - kosinus,
- $tg()$ - tangens,

- $ctg()$ - kotangens.

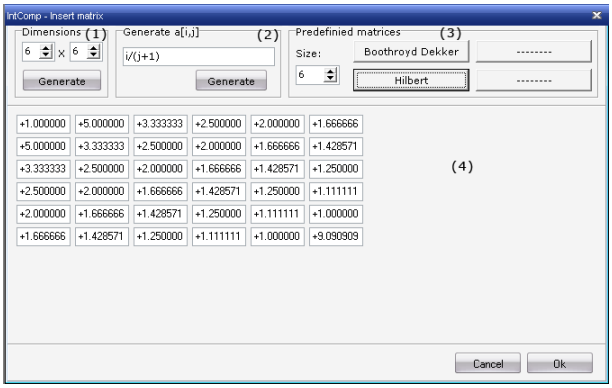
Dla przedziałów mamy dodatkowo

- $width[,]$ - szerokość przedziału,
- $rad[,]$ - promień przedziału,
- $mid[,]$ - środek przedziału,
- $[,]dist[,]$ - odległość dwóch przedziałów.

W przypadku zastosowania powyższych funkcji dla macierzy lub wektorów (oprócz czterech podstawowych działań arytmetycznych) obliczenia są wykonywana na każdym elemencie osobno.

Wprowadzania macierzy

Liczne ułatwienia usprawniają pracę z programem. Do nich należy między innymi narzędzie do wprowadzania macierzy i wektorów. Ułatwia ono wprowadzanie dużych macierzy i wektorów a także umożliwia generowanie niestandardowych macierzy, których elementy są określone wzorem.



Rysunek 2: Okno wprowadzania macierzy

Kolejne sekcje narzędzia opisano poniżej

1. Ustawianie rozmiarów macierzy - w widocznych okienkach należy ustawić (wpisać, bądź ustawić przyciskami) wartość odpowiednio wierszy i kolumn macierzy. Po naciśnięciu przycisku *Generate* na głównym ekranie (4) pojawi się tablica krotek, w które należy wpisać żądane wartości macierzy, przy czym pusta wartość zostaje automatycznie zamieniona na wartość zero.

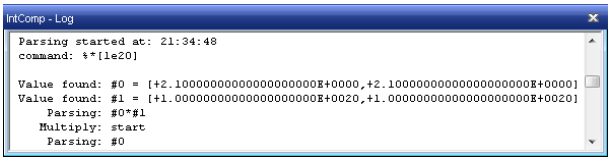
2. Generator macierzy ze wzoru - w okienku należy wpisać wyrażenie matematyczne, które ma wyznaczać elementy macierzy, przy czym generator ten rozpoznaje zmienne: i - numer bieżącego wiersza, j - numer bieżącej kolumny, m - ilość wierszy, n - ilość kolumn. Poza podstawowymi działaniami (+,−,/,∗) dostępne są tutaj również funkcje trygonometryczne: *sin*(), *cos*(), *ctg*(), *tg*() oraz inne: *abs*(), *sqrt*(), *log*(), *exp*(), a także potęgowanie a ^ b i silnia $x!$. O wszelkich nieprawidłowościach podczas obliczania wartości elementów macierzy poinformuje okienko informacyjne. Aby wygenerować macierz należy nacisnąć na przycisk *Generate*.

3. Gotowe macierze kwadratowe - w okienku należy ustawić stopień macierzy kwadratowej, po czy nacisnąć na przycisk *Generate*.

4. Główny panel wprowadzania macierzy - służy do wprowadzania elementów lub podglądu generowanej macierzy. Przyciski *Cancel* oraz *Ok* służą odpowiednio do odrzucenia i zatwierdzenia wprowadzonej macierzy.

Dodatkowe funkcje

Ciekawym dodatkiem jest okienko informacyjne, które wyświetla informacje z każdej sekcji, która aktualnie pracuje nad wyrażeniem.



Rysunek 3: Okno wprowadzania macierzy

Okienko to otwieramy poprzez wybranie pozycji *Pokaż wyniki pośrednie* w zakładce *Opcje* menu głównego lub naciśniecie kombinacji klawiszy *Ctrl+Alt+T*. Dzięki niemu użytkownik ma wgląd na wyniki poszczególnych części obliczanego wyrażenia. Również w przypadku błędu w wyrażeniu informacje na temat miejsca wystąpienia tego błędu są wypisywane w okienku informacyjnym.

Zmiana arytmetyki w programie *IntComp* następuje po wybraniu odpowiedniej pozycji w menu *Arytmetyka*. Pomocne tutaj są polecenia z menu

Format, które służą do konwertowania wyrażenia pomiędzy formatem arytmetyki przedziałowej i rzeczywistej. To znaczy, że na przykład wprowadzona w arytmetyce rzeczywistej macierz może przy pomocy tych poleceń zostać skonwertowana tak aby była rozumiana przez parser arytmetyki przedziałowej.

W linii komend można również używać znaku %, w którego miejsce zostaje wstawiona wartość poprzedniego wyniku, przy czyw wartość nie jest zczytywana z ekranu tylko z pamięci wewnętrznej programu.

Literatura

[1] A. Kiełbasiński, H. Schwetlick, *Numeryczna Algebra Liniowa*, Wydawnictwa Naukowo-Techniczne, Warszawa, **1992**

[2] T. Jurlewicz, Z. Skoczylas, *Algebra liniowa 1*, Państwowe Wydawnictwo Naukowe, Warszawa, **2004**

[3] A. Marciniak, D. Gregulec, J. Kaczmarek, *Podstawowe procedury numeryczne w języku Turbo Pascal*, Wydawnictwo Nakom, Poznań, **1997**

[4] M. Moszyńska, J. Święcicka, *Geometria z algebrą liniową*, Państwowe Wydawnictwo Naukowe, Warszawa, **1987**

[5] B. Hayes, *A lucid interval*, American Scientist magazine, Vol. 91, **2003**

[6] R. E. Moore, *Interval arithmetic and automatic error analysis in digital computing*, Stanford University California, **1962**

[7] S. Shayer, *Interval arithmetic with some applications for digital computers*, Lockheed Palo Alto Research Laoratory Palo Alto, California, Vol. 7, **1965**

[8] A. Marciniak, A. Marlewski, *Interwałowe reprezentacje liczb niemaszynowych w języku Object Pascal*, Wydawnictwo NAKOM Poznań, ProDalog 7, **1998**

[9] Andrzej Marciniak, $0,1$, Wydawnictwo NAKOM Poznań, ProDalog 5, **1997**