UNIWERSYTET GDAŃSKI
Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki

Witold Bołt
nr albumu: 140530

Zastosowanie układów falkowych do aproksymacji rozwiązań równań różniczkowych

Praca magisterska na kierunku:
INFORMATYKA
Promotor:
dr P. Arłukowicz

Gdańsk 2008

Słowa kluczowe

falki, funkcja skalująca, analiza wielorozdzielcza, równania Naviera–Stokesa, aproksymacja, pola wektorowe o zerowej dywergencji, metody numeryczne

Spis treści

Wstęp

Niniejsza praca poświęcona jest zastosowaniom fragmentów teorii falek (ang. *wavelet theory*) w algorytmach numerycznych, a konkretnie do aproksymacji rozwiązania wybranego równania różniczkowego. Podstawy teorii falek, potrzebne do zrozumienia dalszego tekstu zawarto w rozdziale pierwszym. Równaniem różniczkowym którego rozwiązanie będziemy aproksymować jest dwuwymiarowe, stacjonarne równanie Naviera–Stokesa rozważane na kwadracie $[0,1]^2$.

Równanie Naviera–Stokesa, wywodzi się z fizycznych modeli zachowania cieczy (abstrahujemy w tej pracy od fizycznego sensu równania) i ma liczne praktyczne zastosowania w różnych gałęziach przemysłu i nauki. Jest ono również bardzo znane z powodu problemów jakie niesie poszukiwanie rozwiązania lub jego aproksymacji. Dlatego też jest tematem wielu badań i publikacji naukowych z zakresu metod i algorytmów numerycznych.

Nasze postępowanie prowadzące do sformułowania i zaimplementowania algorytmu wyliczającego aproksymację powyższego zagadnienia przebiegać będzie w następujący sposób. Najpierw wybierzemy pewną funkcję skalującą, która będzie podstawą do zbudowania tzw. analizy wielorozdzielczej, czyli ciągu podprzestrzeni domkniętych przestrzeni wszystkich funkcji całkowalnych z kwadratem określonych na odcinku $[0,1]$ zerujących się na jego brzegu wraz z pochodną. W celu budowy wspomnianej analizy wielorozdzielczej zajmiemy się konstrukcją zmodyfikowanych bazowych funkcji brzegowych – opis tej konstrukcji znajduje się w pierwszej części rozdziału 2. Mając zdefiniowaną analizę wielorozdzielczą na odcinku, konstruujemy na jej podstawie analizę wielorozdzielczą funkcji określonych na $[0,1]^2$ spełniających warunek brzegowy i warunek zerowej dywergencji zawarte w układzie Naviera–Stokesa. Wśród tych funkcji poszukujemy rozwiązania równia. Druga część rozdziału drugiego zawiera sformułowanie układów równań liniowych na współczynniki bazowe rozwiązania przybliżonego. Proponowany przez nas schemat rozwiązania zagadnienia wejściowego polega na sformułowaniu i rozwiązaniu ciągu układów liniowych. Granica tego ciągu stanowi aproksymację poszukiwaną dla zadanego poziomu rozdzielczości. Pod koniec drugiego rozdziału dowodzimy twierdzenie mówiące o tym, że wraz z podnoszeniem rozdzielczości (sterowanej przez parametr zwyczajowo oznaczany przez j) aproksymacja zbliża się do rozwiązania właściwego.

Znaczna większość tekstu rozdziału drugiego opiera się o artykuł [16]. W stosunku do tekstu oryginalnego poczyniono jednak kilka poprawek. W szczególności w uwadze 2.3 pokazano w jaki sposób stosować algorytm dla różnych wartości parametru liczbowego ν. Ujednolicono również niejasne oznaczenia pochodnych. Znacznej przeróbce uległ

również dowód twierdzenia 2.4, który stał się bardziej zrozumiały (i kompletny!) dzięki wprowadzeniu lematu 3.

Rozdział trzeci poświęcony jest implementacji prezentowanego algorytmu. W jego pierwszej części prezentowana jest modyfikacja, cytowanego w rozdziale drugim, algorytmu, która pozwala w znacznym stopniu ograniczyć liczbę całek potrzebnych do wyliczenia współczynników układu równań. Zastosowanie tych modyfikacji pozwala uzyskać stałą, niezależną od poziomu rozdzielczości, liczbę całek potrzebnych do zbudowania macierzy głównej układu i dodatkowo ułatwia budowanie tych macierzy dla wyższych poziomów rozdzielczości w oparciu o poziomy niższe. Prezentowana implementacja, dzięki zastosowanym optymalizacjom może skorzystać z dokładniejszych typów danych (liczb wymiernych o długości ograniczonej jedynie rozmiarem pamięci komputera) utrzymując przy tym akceptowalny czas działania. Przykładowe wyniki wraz ze zmierzonym czasem wykonania zaprezentowano w rozdziale 4.

Wyniki zawarte w rozdziale trzecim są przedmiotem dalszych badań prowadzonych przez autora. Ich rozszerzona i uogólniona wersja zostanie zgłoszona do publikacji jako artykuł naukowy.

ROZDZIAŁ 1

Podstawy teorii falek

W tym rozdziale przedstawimy podstawowe definicje i fakty teorii falek używane w dalszej części pracy. Znaczna większość informacji została zaczerpnięta z [15]. Dla ustalenia uwagi i uproszczenia cała teoria została zaprezentowana w kontekście przestrzeni funkcyjnej $L_2(\mathbb{R})$. Wszystkie pojęcia dają się jednak uogólnić na niektóre inne przestrzenie funkcyjne. W dalszej części pracy korzystamy z takich uogólnień, konstruując na przykład analizę wielorozdzielczą dla pewnej podprzestrzeni przestrzeni $L_2([0,1])$.

1.1. Analiza wielorozdzielcza i funkcja skalująca

Definicja 1.1 (analiza wielorozdzielcza)**.** Analizą wielorozdzielczą nazywamy ciąg $(V_j)_{j\in\mathbb{Z}}$ domkniętych podprzestrzeni $L_2(\mathbb{R})$ spełniający następujące warunki:

a) $\ldots \subset V_{-1} \subset V_0 \subset V_1 \subset \ldots$,

b) $\operatorname{span} \bigcup_{j\in\mathbb{Z}} V_j = L_2(\mathbb{R})$,

c) $\bigcap_{j\in\mathbb{Z}} V_j = \{0\}$,

d) $f \in V_j$ wtedy i tylko wtedy, gdy $f(2^{-j}\cdot) \in V_0$,

e) $f \in V_0$ wtedy i tylko wtedy, gdy $f(\cdot - m) \in V_0$ dla wszystkich $m \in \mathbb{Z}$,

f) istnieje funkcja $\Phi \in V_0$, taka, że układ $\{\Phi(\cdot - m)\}_{m\in\mathbb{Z}}$ jest bazą ortonormalną w V_0.

Definicja 1.2 (funkcja skalująca)**.** Funkcja Φ z ostatniego punkt definicji analizy wielorozdzielczej nazywana jest funkcją skalującą (lub „falką ojcem" od angielskiego *„father wavelet"*).

Uwaga 1.3. Oczywiście dla każdej analizy wielorozdzielczej istnieje nieskończenie wiele funkcji skalujących. Jeśli bowiem Φ jest pewną funkcją skalującą danej analizy, to również $\Phi(\cdot - m)$ dla $m \in \mathbb{Z}$ jest funkcją skalującą. Nie oznacza to jednak, że samo poszukiwanie funkcji skalującej jest proste. Okazuje się jednak, że w definicji analizy wielorozdzielczej można zamienić punkt f) na równoważny warunek f') postaci:

f') istnieje funkcja $\Phi \in V_0$ taka, że układ $\{\Phi(\cdot - m)\}_{m\in\mathbb{Z}}$ jest bazą Riesza w V_0.

Równoważność warunków f) i f') wynika z faktu, że baza ortonormalna jest również bazą Riesza dla $c = C = 1$ oraz z następującego twierdzenia.

Twierdzenie 1.4. *Załóżmy, że $\Phi \in L_2(\mathbb{R})$ jest taką funkcją, że $\{\Phi(\cdot - m)\}_{m\in\mathbb{Z}}$ jest bazą Riesza w $V_0 := \operatorname{span}\{\Phi(\cdot - m)\}_{m\in\mathbb{Z}}$. Istnieje wtedy funkcja $\Phi_1 \in V_0$ taka, że $\{\Phi_1(\cdot - m)\}_{m\in\mathbb{Z}}$ jest bazą ortonormalną w V_0.*

Podsumowując, aby sprawdzić, że ciąg $\{V_j\}_{j\in Z}$ podprzestrzeni domkniętych przestrzeni $L_2(\mathbb{R})$ tworzy analizę wielorozdzielczą wystarczy sprawdzić warunki a)-e) oraz f'). Mając funkcję Φ spełniającą warunek f') można uzyskać wzór na Φ_1, która spełnia f). W praktyce jednak często wygodniej jest pracować z funkcją Φ (mówimy wtedy o analizie wielorozdzielczej Riesza), zwłaszcza gdy postać funkcji Φ jest dużo „prostsza" niż Φ_1 (tak jest na przykład w przypadku bazowych funkcji giętych, których używać będziemy w dalszych rozdziałach).

Z powyższych definicji widać wyraźnie, że funkcja skalująca w pełni określa analizę wielorozdzielczą (ortonormalną lub Riesza). Znając bowiem funkcję skalującą, znamy również bazę V_0 więc w rezultacie znamy całą tą przestrzeń. Z warunku $d)$ definicji analizy wielorozdzielczej wiemy więc dokładnie jaka jest postać wszystkich innych V_j.

Poniższy fakt, będący prostym wnioskiem z definicji, okaże się podstawą konstrukcji przedstawionej w następnym rozdziale.

Fakt 1.5. *Jeśli Φ jest funkcją skalującą pewnej analizy wielorozdzielczej, to istnieją liczby h_k dla $k \in \mathbb{Z}$, takie, że:*

$$\Phi(x) = \sum_{k\in\mathbb{Z}} h_k \Phi(2x - k). \tag{1.1}$$

Dowód. Na mocy definicji $\Phi \in V_1$, a stąd $\Phi(x/2) \in V_0$. Z warunku $f)$ lub $f')$ mamy więc, że istnieją h_k dla $k \in \mathbb{Z}$ takie, że:

$$\Phi(x/2) = \sum_{k\in\mathbb{Z}} h_k \Phi(x - k),$$

a stąd (podstawiając $x := 2x$):

$$\Phi(x) = \sum_{k\in\mathbb{Z}} h_k \Phi(2x - k).$$

□

Równanie (1.1) nazywamy równaniem skalującym. W przypadku, gdy funkcja skalująca Φ ma zwarty nośnik, prawie wszystkie (wszystkie poza skończoną liczbą) współczynniki h_k są równe zero.

1.2. Falki

Definicja 1.6 (falka i baza falkowa)**.** Falką (lub „falką matką" od ang. „*mother wavelet*") nazywamy funkcję $\Psi \in L_2(\mathbb{R})$ taką, że rodzina funkcji:

$$\Psi_{j,k} = 2^{j/2}\Psi(2^j \cdot -k),$$

gdzie $j, k \in \mathbb{Z}$, jest bazą ortonormalną przestrzeni $L_2(\mathbb{R})$. Zbiór $\{\Psi_{j,k}\}_{j,k\in\mathbb{Z}}$ nazywać będziemy bazą falkową.

Uwaga 1.7. W dalszym tekście przyjmujemy konwencję, że symbol $f_{j,k}$, gdzie f jest dowolną funkcją, ma takie znaczenie jak w definicji falki.

Termin falka (ang. wavelet) początkowo związany z podanym wyżej konkretnym pojęciem, zaczął obejmować swoim znaczeniem całą teorię związaną z konstrukcją, analizą i zastosowaniem układów falkowych, analiz wielorozdzielczych i innych podobnych obiektów[1].

Istnieje wiele powiązań między analizą wielorozdzielczą a falką i bazą falkową. Zauważmy bowiem, że skoro podprzestrzenie domknięte V_j i V_{j+1} spełniają warunek $V_j \subset V_{j+1}$, to istnieje podprzestrzeń W_j prostopadła do V_j taka, że $V_{j+1} = V_j \oplus W_j$. Okazuje się, że w przestrzeni W_0 można znaleźć taką funkcję Ψ, że zbiór $\{\Psi(\cdot - k) : k \in \mathbb{Z}\}$ jest bazą ortonormalną w W_0. Z własności analizy wielorozdzielczej wynika wtedy, że zbiór $\{\Psi_{j,k} : j, k \in \mathbb{Z}\}$ jest bazą ortonormalną w $L_2(\mathbb{R})$. Co więcej istnieje dość prosty algorytm pozwalający na znalezienie funkcji Ψ w oparciu o funkcję skalującą i jej równanie skalujące. Dzięki temu można konstruować falki spełniające pewne dodatkowe warunki nakładając na funkcję skalującą odpowiednie założenia.

Głównym zastosowaniem analizy falkowej jest poszukiwanie aproksymacji zadanej funkcji f. Dążymy do znalezienie funkcji $f_j \in V_j$, które byłyby „coraz bliższe" f wraz ze wzrostem j. Szukając takich przybliżeń możemy użyć jednego z następujących sposobów:

a) korzystamy z równości $V_j = V_0 \oplus W_0 \oplus W_1 \oplus \ldots \oplus W_{j-1}$ i otrzymujemy (dla $j > 0$):

$$f_j = \sum_{k\in\mathbb{Z}} c_k\Phi_{0,k} + \sum_{k\in\mathbb{Z}} d_{0,k}\Psi_{0,k} + \ldots + \sum_{k\in\mathbb{Z}} d_{j-1,k}\Psi_{j-1,k},$$

co interpretujemy w ten sposób, że startujemy z określonego „zerowego" poziomu rozdzielczości i dodajemy kolejne detale z wyższych poziomów;

b) budujemy f_j z samych detali, tzn.:

$$f_j = \sum_{i\leqslant j}\sum_{k\in\mathbb{Z}} d_{i,k}\Psi_{i,k};$$

[1] Spotykamy się na przykład z określeniami „teoria falek" lub „analiza falkowa".

c) opieramy się jedynie o funkcję skalującą i otrzymujemy:

$$f_j = \sum_{k\in\mathbb{Z}} c_k \Phi_{j,k}.$$

W niniejszej pracy korzystać będziemy z ostatniej z podanych metod.

1.3. Przykłady

1.3.1. Falka Haara

Najbardziej znanym przykładem falki jest tak zwana falka Haara. Pojawiła się ona w literaturze dużo wcześniej niż sama teoria falek[2] i ma bardzo prostą postać:

$$H(x) = \begin{cases} 1 & x \in [0, \frac{1}{2}) \\ -1 & x \in [\frac{1}{2}, 1) \\ 0 & x \notin [0, 1) \end{cases}.$$

Mimo tej prostoty sformułowania, dowód zgodności z definicją 1.6 jest już dość skomplikowany[3]. Z względu na prostotę definicji i obliczeń falka Haara jest szeroko stosowana. Na przykład w książce [9] pokazano szereg algorytmów aproksymacji rozwiązań różnego rodzaju równań różniczkowych. Niestety przy bardzo wielu zaletach falki Haara obecna jest też jedna bardzo znaczna wada. Wszystkie otrzymane za pomocą tej falki aproksymacje, są podobnie jak sama falka, nieciągłe. Wada ta ogranicza zastosowanie falki Haara w dziedzinie aproksymacji procesów czy sygnałów ciągłych, nie przeszkadza jednak stosować jej w przetwarzaniu sygnałów dyskretnych – np. w algorytmach kompresji obrazów cyfrowych[4].

Warto dodatkowo odnotować fakt, że funkcją skalującą (falką ojcem) dla układu falkowego Haara jest funkcja $\mathbf{1}_{[0,1)}$.

1.3.2. Funkcje gięte

Definicja 1.8 (funkcja gięta)**.** Niech $a \in \mathbb{R}$ oraz $d \in \mathbb{N}_+$. Funkcją giętą rzędu d z węzłami w punktach $a\,\mathbb{Z} = \{az : z \in \mathbb{Z}\}$ nazywamy dowolną funkcję $f\colon \mathbb{R} \to \mathbb{R}$, która jest klasy C^{d-2} oraz jest wielomianem stopnia co najwyżej $d-1$ po obcięciu do każdego z odcinków $[ja, (j+1)a]$ dla $j \in \mathbb{Z}$. Przestrzeń wszystkich funkcji giętych rzędu d z węzłami w $a\,\mathbb{Z}$ będziemy oznaczać $\mathcal{S}^d(a\,\mathbb{Z})$.

[2] Falka Haara pochodzi z pracy opublikowanej przez A. Haara w 1910 roku, natomiast teoria falek jako taka zaczęła się rozwijać dopiero na początku lat '80 XX wieku.

[3] Można go znaleźć na przykład w [15].

[4] Na stronie `http://www.tomgibara.com/computer-vision/haar-wavelet` można znaleźć interaktywną demonstrację algorytmu kompresji i dekompresji obrazu za pomocą układu Haara.

Podaną wyżej definicję można łatwo uogólnić na ogólny przypadek węzłów w dowolnym dyskretnym podzbiorze $\mathbb{R}$ oraz na funkcje określone na przedziale.

Uwaga 1.9. Przyjmujemy konwencję, że przestrzeń C^{-1} oznacza przestrzeń funkcji mierzalnych, a C^0 funkcji ciągłych.

Definicja 1.10 (splot)**.** Niech $f, g \in L_1(\mathbb{R})$. Iloczynem splotowym (splotem) funkcji f, g nazywamy funkcję $f * g$ zdefiniowaną następującym wzorem:

$$[f * g](x) = \int_{\mathbb{R}} f(x-y)g(y)dy.$$

Definicja 1.11 (bazowe funkcje gięte)**.** Dla $d = 1, 2, \ldots$ określamy funkcje $N_d(x)$ zwane bazowymi funkcjami giętymi w następujący sposób:

(a) $N_1 = \mathbf{1}_{[0,1)}$,

(b) $N_{d+1} = N_d * N_1$ dla $d > 1$, gdzie $*$ oznacza iloczyn splotowy.

Fakt 1.12. *Niech $d \geqslant 2$. Funkcja N_d ma następujące własności:*

a) $N_d(x) > 0$ *dla* $x \in (0, d)$,

b) $\operatorname{supp} N_d = [0, d]$,

c) $N_d \in \mathcal{S}^d(\mathbb{Z})$,

d) $\sum_{k\in\mathbb{Z}} N_d(x-k) \equiv 1$,

e) $N_d(\frac{d+1}{2} - x) = N_d(\frac{d+1}{2} + x)$, *dla* $x \in \mathbb{R}$,

f) $N'_{d+1}(x) = N_d(x) - N_d(x-1)$.

Następujące twierdzenia motywują niejako nazwę „bazowe funkcje gięte".

Twierdzenie 1.13. *Jeśli $f \in \mathcal{S}^d(\mathbb{Z})$ oraz $\operatorname{supp} f \subset [0, d]$, to $f = cN_d$ dla pewnego $c \in \mathbb{R}$.*

Twierdzenie 1.14. *Niech w będzie wielomianem stopnia nie większego niż $d-1$. Wtedy istnieje jedyna funkcja gięta $f \in \mathcal{S}^d(\mathbb{Z})$ taka, że:*

$$f(x) = \sum_{k=-d}^{0} a_k N_d(x-k),$$

oraz $f|_{[0,1]} = w|_{[0,1]}$.

Twierdzenie 1.15. *Każdą funkcję $f \in \mathcal{S}^d(\mathbb{Z})$ można jednoznacznie przedstawić jako:*

$$f(x) = \sum_{k\in\mathbb{Z}} a_k N_d(x-k).$$

Uwaga 1.16. Ze względu na to, że $\operatorname{supp} N_d(\cdot - k)$ jest zwarty, w sumie w powyższym twierdzeniu, dla dowolnego ustalonego x jedynie skończenie wiele wyrazów różnych jest zera.

Przez $\mathcal{S}_2^d(2^j\,\mathbb{Z})$ będziemy oznaczać przestrzeń $\mathcal{S}^d(2^j\,\mathbb{Z}) \cap L_2(\mathbb{R})$. Z wyżej wymienionych faktów i twierdzeń wynika następujący fakt.

Fakt 1.17. *Przestrzeń $\mathcal{S}_2^d(2^j\,\mathbb{Z})$ jest domkniętą podprzestrzenią $L_2(\mathbb{R})$, a układ $\{N_d(\cdot - k) : k \in \mathbb{Z}\}$ jest bazą Riesza w $\mathcal{S}_2^d(\mathbb{Z})$.*

Stąd wynika następujące twierdzenie.

Twierdzenie 1.18. *Dla dowolnego $d \in \mathbb{N}_+$ przestrzenie $V_j := \mathcal{S}_2^d(2^{-j}\,\mathbb{Z})$, gdzie $j \in \mathbb{Z}$, tworzą analizę wielorozdzielczą, a funkcja N_d jest jej funkcją skalującą.*

Następny fakt podaje wzór na współczynniki z równania skalującego funkcji N_d.

Fakt 1.19. *Niech $d \in \mathbb{N}_+$. Funkcja N_d spełnia następujące równanie skalujące:*

$$N_d(x) = \sum_{k=0}^{d} 2^{1-d} \binom{d}{k} N_d(2x - k).$$

Jak już wspomnieliśmy wcześniej istnieją ogólne metody konstrukcji falki na bazie funkcji skalującej. Wiemy więc, że istnieją falki będące funkcjami giętymi. Podstawy takich konstrukcji można znaleźć w [15]. Bardziej zaawansowane konstrukcje oparte o funkcje gięte można natomiast znaleźć w pracy [3].

1.4. Zastosowania falek

Teoria falek znajduje szerokie zastosowanie głównie w teorii aproksymacji. Zarówno falki jak i funkcje skalujące tworzą bazy funkcyjne pozwalające w prosty i efektywny sposób wyliczać współczynniki rozwinięcia bazowego. Poza w miarę oczywistymi zastosowaniami teoretycznymi w zagadnieniach aproksymacji falki znalazły również wiele praktycznych zastosowań.

Falki powszechnie stosuje się w algorytmach kompresji danych (np. algorytm kompresji obrazów JPEG2000[5], algorytm kompresji filmów wideo Dirac[6]) oraz w algorytmach rozpoznawania wzorców (sygnał przedstawiany jest w formie współczynników rozwinięcia

[5]Więcej informacji o algorytmie JPEG2000 można znaleźć na oficjalnej stronie `http://www.jpeg.org/jpeg2000/`.

[6]Więcej informacji o projekcie Dirac można znaleźć na oficjalnej stronie projektu `http://dirac.sourceforge.net/` lub na stronie `http://www.diracvideo.org/` zawierającą aktualną wersję implementacji tego algorytmu.

w pewnej bazie falkowej i następnie przekazywany jest do algorytmów opartych np. o sieci neuronowe). Podobne algorytmy stosuje się również w analizie nieliniowych układów dynamicznych[7]

W analizie i przetwarzaniu sygnałów rozwinięto algorytmy szybkiej transforamty falkowej (FWT) i dyskretnej transformaty falkowej (DWT), które stosowane są nieraz jako zamienniki popularnych techniki takich jak szybka transformata Fouriera (FFT) lub dyskretna transformata kosinusowa (DCT).

Coraz popularniejsze jest również wykorzystanie układów falkowych w metodach numerycznych – np. do przybliżania rozwiązań równań różniczkowych lub całkowych oraz w algorymtach wykorzystywanych w grafice komputerowej – na przykład w algorytmach typu „global illumination”[8].

[7] W książce [2] opisano dokładnie jeden z takich algorytmów.

[8] Na stronie `http://www.multires.caltech.edu/teaching/courses/waveletcourse/` można znaleźć materiały do kursu „Wavelets in Computer Graphics” organizowanego w ramach konferencji SIGGRAPH.

ROZDZIAŁ 2

Podstawy teoretyczne i sformułowanie algorytmu

Algorytm prezentowany w tym rozdziale oparty jest na [16]. Składa się on z dwóch głównych części. W pierwszej, startując z pewnej analizy wielorozdzielczej $\{V_j\}$ określonej na prostej rzeczywistej, z funkcją skalująca ϕ o zwartym nośniku, budujemy nową analizę na odcinku $[0,1]$ dla funkcji spełniających warunek:

$$f(0) = f'(0) = f(1) = f'(1) = 0.$$

W tym celu będziemy przekształcać bazę $\{\phi_{j,k} : k \in \mathbb{Z}\}$ usuwając funkcje o nośnikach rozłączonych z $[0,1]$ i modyfikując funkcje brzegowe (takie, których nośnik „zahacza" o brzeg odcinka). Funkcje, których nośnik mieści się całkowicie w $[0,1]$ pozostaną niezmienione, a z funkcji brzegowych powstaną nowe funkcje $\phi^L_{j,k}, \phi^R_{j,k}$ spełniające warunki $\phi^L_{j,k}(0) = (\phi^L_{j,k})'(0) = \phi^R_{j,k}(1) = (\phi^R_{j,k})'(1) = 0$. Następnie z tak skonstruowanej bazy, za pomocą iloczynu tensorowego, zbudujemy analizę wielorozdzielczą w przestrzeni $V \subset H^1_0([0,1]^2)^2$ funkcji zerujących się wraz z pochodną na $\partial[0,1]^2$, o zerowej dywergencji w kwadracie $[0,1]^2$, w której będziemy poszukiwali rozwiązania naszego problemu.

W drugiej części rozdziału, stosujemy uzyskaną bazę, do sformułowania i rozwiązania ciągu zagadnień, których rozwiązania przybliżają poszukiwane rozwiązanie problemu Naviera–Stokes.

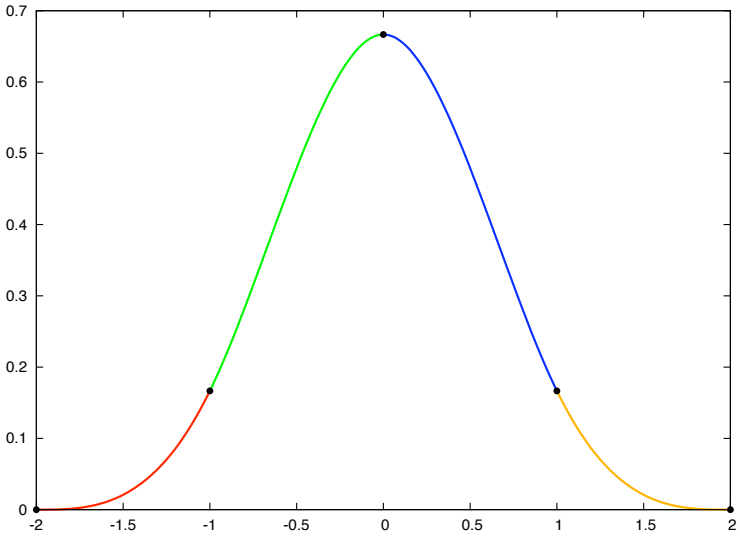
Wszystkie rozważania teoretyczne w tym rozdziale przeplatane są konkretnymi wyliczeniami przeprowadzonymi dla konkretnie ustalonej funkcji skalującej $\varphi(\cdot) = N_4(\cdot + 2)$.

Po wyliczeniu z definicji 1.11 funkcji N_4 i przesunięciu o 2 otrzymujemy następujący wzór jawny:

$$\varphi(x) = N_4(x+2) = \begin{cases} 0 & x \notin (-2,2) \\ \frac{1}{6}x^3 + x^2 + 2x + \frac{4}{3} & x \in [-2,-1] \\ -\frac{1}{2}x^3 - x^2 + \frac{2}{3} & x \in [-1,0] \\ \frac{1}{2}x^3 - x^2 + \frac{2}{3} & x \in [0,1] \\ -\frac{1}{6}x^3 + x^2 - 2x + \frac{4}{3} & x \in [1,2] \end{cases}$$

Na rysunku 2.1 przedstawiono wykres funkcji φ.

W dalszych rozważaniach w tym rozdziale, przyjmuje się konwencję, że ϕ oznacza pewną „dowolną" funkcję skalującą o zwartym nośniku, pewnej analizy wielorozdzielczej określonej na prostej rzeczywistej, natomiast $\varphi(\cdot) = N_4(\cdot + 2)$ jest konkretną wybraną funkcją, dla której prowadzić będziemy szczegółowe wyliczenia.



Rysunek 2.1. Wykres funkcji φ. Czarną kropką zaznaczono miejsca sklejeń wykresów poszczególnych wielomianów.

2.1. Konstrukcja funkcji bazowych na odcinku $[0, 1]$

Konstrukcja przedstawiona w tym rozdziale oparta jest, poza wspomnianą pracą [16], na pracy [6].

W niniejszych rozważaniach zakładamy, że dana jest pewna funkcja skalująca ϕ o nośniku $[a, b]$ dla pewnych liczb całkowitych a, b. Dla uproszczenia załóżmy, że $N := b-a \geqslant 4$. Wiadomo wtedy, że funkcja ϕ spełnia następujące równanie skalujące:

$$\phi(x) = \sum_{k=a}^{b} h_k \phi(2x - k) \tag{2.1}$$

dla pewnych liczb h_k. Zakładamy dodatkowo, że funkcja ϕ spełnia warunki Stranga–Fixa:

$$\hat{\phi}^{(n)}(2k\pi) = 0$$

dla $k \in \mathbb{Z}\backslash\{0\}$ i $n = 0, 1, \ldots, N-1$, gdzie $\hat{\phi}$ oznacza transformatę Fouriera[1] funkcji ϕ. Można pokazać, że warunki te implikują, że jednomiany $1, x, \ldots, x^{N-1}$ są kombinacjami liniowymi funkcji $\phi(x-k)$.

W naszym konkretnym przypadku funkcji φ wiadomo, że $\operatorname{supp}\varphi = [-2, 2]$, wobec czego $N = 4$. Nie musimy sprawdzać warunków Stranga–Fixa dla φ, ponieważ z twierdzeń 1.13, 1.14, 1.15 wiemy, że jednomiany $1, x, x^2, x^3$ są kombinacjami liniowymi $\varphi(x-k)$. Na mocy

[1]Definicję i podstawowe własności transformaty Fouriera można znaleźć w dodatku A.2.1.

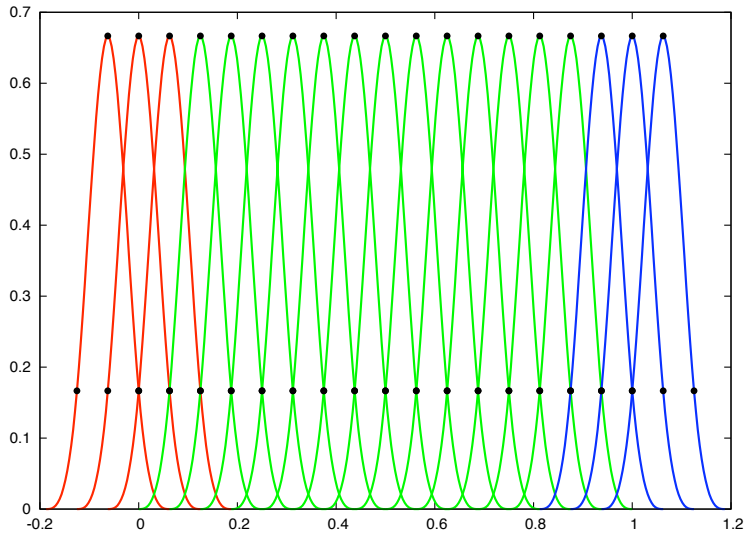
faktu 1.19 wiadomo, że $N_4(x) = \sum_{k=0}^{4} 2^{-3}\binom{4}{k} N_4(2x-k)$, a stąd mamy:

$$\varphi(x) = N_4(x+2) = \sum_{k=0}^{4} 2^{-3}\binom{4}{k} N_4(2(x+2)-k) =$$
$$\sum_{k=0}^{4} 2^{-3}\binom{4}{k} N_4(2x+4-k) = \sum_{k=0}^{4} 2^{-3}\binom{4}{k} \varphi(2x+2-k) =$$
$$\sum_{k=-2}^{2} 2^{-3}\binom{4}{k+2} \varphi(2x-k),$$

czyli φ spełnia równanie skalujące z współczynnikami h_k postaci:

$$h_k = 2^{-3}\binom{4}{k+2} \quad \text{dla } k = -2, -1, \ldots, 2.$$

Ustalmy $j \in \mathbb{Z}$. Rozważmy rodzinę $\phi_{j,k} = 2^{j/2}\phi(2^j x - k)$ dla $k \in Z$. Niech Δ będzie takim zbiorem indeksów k dla których $\operatorname{supp}\phi_{jk} \cap (0,1) \neq \emptyset$. Zbiór Δ można rozbić na trzy podzbiory: zbiór indeksów wewnętrznych Δ_I zawierający te k dla których $\operatorname{supp}\phi_{j,k} \subset (0,1)$, zbiór indeksów lewych Δ_L zawierający te $k \notin \Delta_I$ dla których $0 \in \operatorname{supp}\phi_{j,k}$ oraz zbiór indeksów prawych Δ_R zawierający $k \notin \Delta_I$ takie, że $1 \in \operatorname{supp}\phi_{j,k}$. Łatwo można pokazać, że dla $j > log_2(b-a)$ wspomniane zbiory są rozłączne i niepuste. Opierając się na podanych definicjach, korzystając z tego, że $\operatorname{supp}\phi = [a,b]$ możemy dokładnie wyliczyć poszczególne zbiory: $\Delta_j^L = \{-b+1, \ldots, -a-1\}$, $\Delta_j^I = \{-a, -a+1, \ldots, 2^j - b\}$, $\Delta_j^R = \{2^j - b + 1, \ldots, 2^j - a - 1\}$.



Rysunek 2.2. Wykresy funkcji $\varphi_{4,k}$ dla $k \in \Delta_4^L$ (kolor czerwony), $k \in \Delta_4^I$ (kolor zielony), $k \in \Delta_4^R$ (kolor niebieski).

Na rysunku 2.2 pokazano wykresy funkcji $\varphi_{4,k}$ dla k odpowiednio w zbiorach lewym, środkowym i prawym. Łatwo sprawdzić, że dla dowolnej ϕ, niezależnie od j, zbiory „lewy" i „prawy" mają, tak jak pokazano to na wykresie, $N-1$ elementów.

Zbiór funkcji $\{\phi_{jk} : k \in \Delta_j^I\}$ nazywać będziemy bazą wewnętrzną przestrzeni $V_0^j([0,1])$ funkcji zerujących się wraz z pochodną na brzegu $[0,1]$. W dalszej części zajmiemy się modyfikacją funkcji o indeksach w zbiorach prawym i lewym. Zaczniemy od konstrukcji nowych funkcji bazowych dla lewego brzegu (blisko zera). Pierwszym krokiem konstrukcji jest zdefiniowanie nowych funkcji:

$$\phi_{j,k}^0(x) = 2^{j/2} \sum_{n=k}^{N-1} \binom{n}{k} \phi(2^j x + n + a)\chi_{[0,+\infty)}$$

dla $k = 0, \ldots, N-1$. Przeanalizujmy postać tej funkcji dla $j = 0$ (będziemy pisać $\phi_k^0(x)$ zamiast $\phi_{0k}^0(x)$):

$$\phi_k^0(x) = \sum_{n=k}^{N-1} \binom{n}{k} \phi(x + n + a)\chi_{[0,+\infty)}. \tag{2.2}$$

Ponieważ ϕ spełnia równanie (2.1), to również ϕ_k^0 daje się rozwinąć w podobną sumę. Stosując wzór (2.1) do definicji ϕ_k^0 i odpowiednio grupując składniki otrzymujemy:

$$\phi_k^0(x) = \sum_{n=0}^{k} \alpha_{k,n}^0 \phi_n^0(2x) + \sum_{n=1-a}^{b-2a} \beta_{k,n}^0 \phi(2x - n). \tag{2.3}$$

Można pokazać, że niezależnie od wyboru ϕ zachodzi $\alpha_{k,k}^0 = 2^{-k}$. Algorytm wyliczania liczb α i β można znaleźć w [14]. W dalszych rozważaniach będą potrzebne nam tylko liczby α.

W przypadku naszej funkcji φ, nowo powstałe funkcje φ_{jk}^0 mają postać:

$$\varphi_{jk}^0(x) = 2^{j/2} \sum_{n=k}^{3} \binom{n}{k} \varphi(2^j x + n - 2)\chi_{[0,+\infty)}(x) \tag{2.4}$$

dla $k = 0, \ldots, 3$. Mamy więc dokładnie 4 nowe funkcje (przyjmujemy $j = 0$ i taką samą konwencję oznaczeń jak poprzednio):

$$\begin{aligned}
\varphi_0^0(x) =& (\varphi(x-2) + \varphi(x-1) + \varphi(x) + \varphi(x+1))\chi_{[0,+\infty)}(x) \\
\varphi_1^0(x) =& (\varphi(x-1) + 2\varphi(x) + 3\varphi(x+1))\chi_{[0,+\infty)}(x) \\
\varphi_2^0(x) =& (\varphi(x) + 3\varphi(x+1))\chi_{[0,+\infty)}(x) \\
\varphi_3^0(x) =& (\varphi(x+1))\chi_{[0,+\infty)}(x)
\end{aligned}$$

Ponieważ znamy dokładny wzór na współczynniki h_k nie musimy korzystać z żadnych metod wyznaczania przybliżeń współczynników α. Korzystając z faktu, że poszczególne przesunięcia φ są liniowo niezależne z definicji, możemy wyznaczyć współczynniki α

poprzez porównanie współczynników przy odpowiednich przesunięciach. Z jednej strony będą to współczynniki z równania (2.3), w którym za $\varphi_n^0(2x)$ wstawiamy rozwinięcie z definicji (2.2):

$$\varphi_k^0(x) = \sum_{n=0}^{k} \alpha_{k,n}^0 \sum_{s=n}^{N-1} \binom{s}{n} \varphi(2x+s+a)\chi_{[0,+\infty)}(2x) + \sum_{n=1-a}^{b-2a} \beta_{k,n}^0 \varphi(2x-n). \tag{2.5}$$

Z drugiej strony do wzoru z definicji (2.2) podstawiamy rozwinięcia φ z równania skalującego (2.1):

$$\varphi_k^0(x) = \sum_{n=k}^{N-1} \binom{n}{k} \sum_{s=a}^{b} h_s \varphi(2x+2n+2a-s)\chi_{[0,+\infty)}. \tag{2.6}$$

W powyższych równaniach występują tylko kombinacje liniowe funkcji postaci $\varphi(2x+A)$. Ze względu na $\chi_{[0,+\infty)}$ możemy pominąć składniki gdzie $A \geqslant 2$ ponieważ nośniki tych funkcji są rozłączone z półprostą $[0,\infty)$. W poniższych tabelach zestawiono współczynniki przy poszczególnych funkcjach wraz z liczbą A (interesują nas tylko liczby α więc pomijamy wiersze z liczbami β):

A	(2.6)	(2.5)
-2	$h_{-2}+h_0+h_2$	$\alpha_{0,0}$
-1	$h_{-1}+h_1$	$\alpha_{0,0}$
0	$h_{-2}+h_0+h_2$	$\alpha_{0,0}$
1	$h_{-1}+h_1$	$\alpha_{0,0}$

Tabela 2.1. $k=0$

A	(2.6)	(2.5)
-2	h_0+2h_2	$\alpha_{1,0}$
-1	$h_{-1}+2h_1$	$\alpha_{1,0}+\alpha_{1,1}$
0	$h_{-2}+2h_0+3h_2$	$\alpha_{1,0}+2\alpha_{1,1}$
1	$2h_{-1}+3h_1$	$\alpha_{1,0}+3\alpha_{1,1}$

Tabela 2.2. $k=1$

A	(2.6)	(2.5)
-2	h_2	$\alpha_{2,0}$
-1	h_1	$\alpha_{2,0}+\alpha_{2,1}$
0	h_0+3h_2	$\alpha_{2,0}+2\alpha_{2,1}+\alpha_{2,2}$
1	$h_{-1}+3h_1$	$\alpha_{2,0}+3\alpha_{2,1}+3\alpha_{2,2}$

Tabela 2.3. $k=2$

A	(2.6)	(2.5)
-2	0	$\alpha_{3,0}$
-1	0	$\alpha_{3,0}+\alpha_{3,1}$
0	h_2	$\alpha_{3,0}+2\alpha_{3,1}+\alpha_{3,2}$
1	h_1	$\alpha_{3,0}+3\alpha_{3,1}+3\alpha_{3,2}+\alpha_{3,3}$

Tabela 2.4. $k=3$

Rozwiązując układy równań zapisane w powyższych tabelach wyliczamy wartości parametrów α. Zebrano je w tabeli 2.5.

W pracach [4],[1],[14] pokazano, że funkcje $\phi_{j,k}^0$ są na odcinku $[0,2^{-j}]$ wielomianami stopnia k, oraz $\operatorname{supp}\phi_{j,k}^0 = [0,\frac{N-k}{2^j}]$. Naszym celem jest teraz modyfikacja funkcji $\phi_{j,k}^0$ tak aby spełniony był warunek $f(0)=f'(0)=0$. Musimy więc zapewnić aby nowo otrzymana

Tabela 2.5. Wartości parametrów $\alpha_{k,n}$.

$k\backslash n$	0	1	2	3
0	1	–	–	–
1	1	$\frac{1}{2}$	–	–
2	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{1}{4}$	–
3	0	0	$\frac{1}{8}$	$\frac{1}{8}$

funkcja $\tilde{\phi}^0_{j,k}$ obcięta do odcinka $[0, 2^{-j}]$ była wielomianem o zerowych współczynnikach przy jednomianach $1, x$. Definiujemy funkcje $\tilde{\phi}^0_{j,k}$ dla $j = 0$:

$$\tilde{\phi}^0_k = \phi^0_k - \lambda_k \phi^0_0 - \mu_k \phi^0_1$$

dla $k = 2, \ldots, N-1$. Funkcje ϕ^0_0 i ϕ^0_1 są wielomianami odpowiednio zerowego i pierwszego stopnia (na odcinku $[0, 2^{-j}]$), wobec tego stałe λ_k, μ_k możemy dobrać tak aby składniki przy $1, x$ zredukowały się. Można pokazać, że w ogólnym przypadku zachodzą wzory:

$$\begin{cases} \mu_2 = \frac{\alpha^0_{2,1}}{\alpha^0_{1,1} - \alpha^0_{2,2}} \\ \mu_k = \frac{\alpha^0_{k,1} + \sum_{i=2}^{k-1} \alpha^0_{k,i}\mu_i}{\alpha^0_{1,1} - \alpha^0_{k,k}}, \quad k > 2 \end{cases} \quad \begin{cases} \lambda_2 = \frac{\alpha^0_{2,0} - \mu_2 \alpha^0_{1,0}}{\alpha^0_{0,0} - \alpha^0_{2,2}} \\ \lambda_k = \frac{\alpha^0_{k,0} - \mu_k \alpha^0_{1,0} + \sum_{i=2}^{k-1} \alpha^0_{k,i}\lambda_i}{\alpha^0_{0,0} - \alpha^0_{k,k}}, \quad k > 2 \end{cases} \tag{2.7}$$

Tak skonstruowane $\tilde{\phi}^0_k$ spełniają warunek $\tilde{\phi}^0_k(0) = (\tilde{\phi}^0_k)'(0) = 0$.

W naszym konkretnym przypadku funkcji φ musimy skonstruować dwie funkcje $\tilde{\varphi}^0_2, \tilde{\varphi}^0_3$. Wartości odpowiednich współczynników μ, λ zebrano w tabeli 2.6.

Tabela 2.6. Współczynniki μ, λ.

k	μ_k	λ_k
2	$\frac{3}{2}$	$-\frac{11}{6}$
3	$\frac{1}{2}$	$-\frac{5}{6}$

Otrzymujemy w ten sposób nowe funkcje:

$$\tilde{\varphi}^0_2 = \varphi^0_2 + \frac{11}{6}\varphi^0_0 - \frac{3}{2}\varphi^0_1$$
$$\tilde{\varphi}^0_3 = \varphi^0_3 + \frac{5}{6}\varphi^0_0 - \frac{1}{2}\varphi^0_1$$

które oczywiście spełniają odpowiednie warunki „znikania w zerze".

Korzystając z wzoru (2.2) oraz z definicji funkcji $\tilde{\phi}^0_k$ mamy, że:

$$\tilde{\phi}^0_k(x) = \sum_{n=0}^{N-1} a_{k,n}\phi(x + n + a)$$

dla $k = 2, \ldots, N-1$ i pewnych współczynników $a_{k,n}$. Definiujemy nowe funkcje bazowe dla lewego brzegu ϕ_k^L eliminując z powyższej sumy funkcję $\phi(x+a)$ (która dla odpowiednio dużych j będzie funkcją „wewnętrzną"):

$$\phi_k^L(x) = \tilde{\phi}_{k+2}^0(x) - \frac{a_{k+2,0}}{a_{2,0}} \tilde{\phi}_2^0(x),$$

dla $k = 1, \ldots, N-3$. Tak otrzymane funkcje są kombinacjami funkcji „lewych":

$$\phi_k^L(x) = \sum_{n=1}^{N-1} b_{k,n}^L \phi(x+n+a), \quad \text{dla } k = 1, \ldots, N-3, \tag{2.8}$$

przy czym zachodzi wzór: $b_{k,n}^L = a_{k+2,n} - \frac{a_{k+2,0}}{a_{2,0}} a_{2,n}$. Funkcje ϕ_k^L spełniają warunki $\phi_k^L(0) = (\phi_k^L)'(0) = 0$. W ten sposób zdefiniowaliśmy bazowe funkcje „lewe" dla $j = 0$. Dla $j > 0$ definiujemy je następująco:

$$\phi_{j,k}^L(x) = 2^{j/2} \phi_k^L(2^j x), \quad \text{dla } k = 1, \ldots, N-3.$$

Warto zauważyć, że niezależnie od j mamy zawsze $N-3$ brzegowych funkcji bazowych po lewej stronie.

W przypadku funkcji φ dążymy do otrzymania dokładnie jednej funkcji brzegowej φ_1^L. Współczynniki $a_{k,n}$ potrzebne do jej wyliczenia możemy łatwo wyliczyć z wzoru (2.2) oraz równania skalującego. W tabeli 2.7 zebrano wartości tych współczynników.

Tabela 2.7. Współczynniki $a_{k,n}$

$k\backslash n$	0	1	2	3
2	$\frac{11}{6}$	$\frac{1}{3}$	$-\frac{1}{6}$	$\frac{1}{3}$
3	$\frac{5}{6}$	$\frac{1}{3}$	$-\frac{1}{6}$	$\frac{1}{3}$

Na bazie wyliczonych wartości $a_{k,n}$ wyznaczamy trzy współczynniki $b_{k,n}^L$. W ten sposób możemy przedstawić funkcję φ_1^L w postaci:

$$\varphi_1^L(x) = \frac{2}{11}\varphi(x-1) - \frac{1}{11}\varphi(x) + \frac{2}{11}\varphi(x+1).$$

Liczby $\frac{2}{11}$ i $\frac{1}{11}$ mogą być niewygodne w obliczeniach. Zauważmy jednak, że funkcja φ_1^L pomnożona przez dowolną stałą niezerową c zachowuje interesujące dla nas własności, w związku z tym ogólny wzór ma następującą postać:

$$\varphi_1^L(x) = c\left(2\varphi(x-1) - \varphi(x) + 2\varphi(x+1)\right), \tag{2.9}$$

gdzie $c \neq 0$ jest dowolne (wybrane stosownie do zastosowań – w przypadku obliczeń numerycznych wygonie przyjąć $c = 1$ lub takie c aby norma funkcji lewej była równa normie funkcji wewnętrznych).

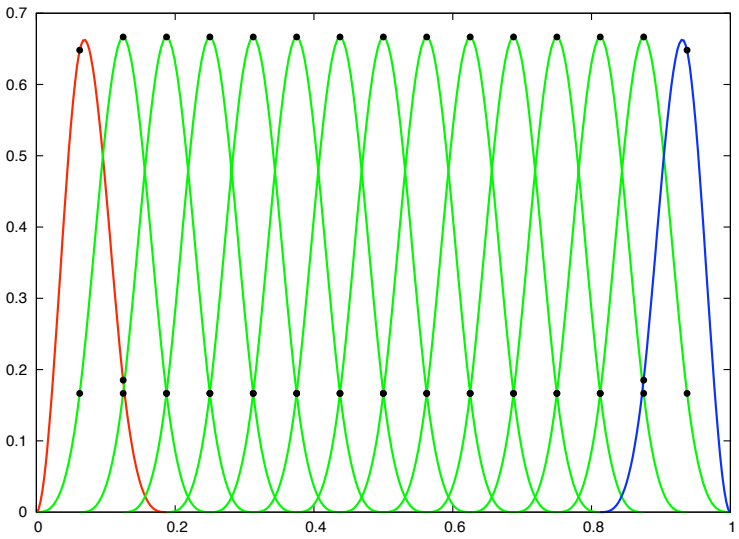
W bardzo podobny sposób możemy skonstruować funkcje brzegowe z prawej strony, które oznaczać będziemy symbolem $\phi^R_{j,k}$. Będzie ich również $N-3$ niezależnie od wyboru j. W przypadku gdy funkcja ϕ jest symetryczna względem zera (a tak jest właśnie w przypadku naszej funkcji φ) możemy zdefiniować „prawe" funkcje jako odbicia: $\phi^R_k = \phi^L_k(1-x)$. Funkcje bazowe mają wówczas postać:

$$\phi^R_{j,k}(x) = 2^{j/2}\phi^L_k(2^j(1-x)). \tag{2.10}$$

Fakt 2.1. *Dla $j > j_0 = \log_2(b-a)$, funkcje bazowe „lewe" i „prawe" mają rozłączne nośniki.*

W dalszych rozważaniach zakładamy, że $j > j_0$ zgodnie z powyższym faktem.

Tak otrzymany układ $\{\phi^L_{j,k}, \phi^R_{j,k} : k = 1, \ldots, N-3\} \cup \{\phi_{j,k} : k \in \Delta_I\}$ stanowi bazę Riesza przestrzeni $V_0^j([0,1])$. Na rysunku 2.3 przedstawiono uzyskany układ bazowy dla funkcji skalującej φ. Jako stałą c przyjęto w tym przypadku $\frac{5}{9}$.



Rysunek 2.3. Otrzymany układ bazowy dla $j = 4$.

2.2. Konstrukcja bazy na $[0,1]^2$

Dla uproszczenia zapisów wprowadźmy następujące oznaczenia. $\Lambda^I_j = \Delta^I_j$, $\Lambda^R_j = \Lambda^L_j = \{1, \ldots, N-3\}$ oraz $\phi^I_{j,k} = \phi_{j,k}$.

W dalszej części pracy będziemy chcieli zapisać rozwiązanie naszego równania w postaci skończonej kombinacji liniowej funkcji bazowych zbudowanych z funkcji skonstruowanych w poprzednim podrozdziale. W naszym konkretnym przypadku poszukiwana funkcja prędkości dana jest przez funkcje współrzędne $u = (u_1, u_2)$, gdzie $u_1, u_2 \colon [0,1]^2 \to \mathbb{R}$.

Ponieważ wiemy, że u spełnia warunek $\operatorname{div} u = 0$ wiadomo, że istnieje taka funkcja rzeczywista dwóch zmiennych, że:

$$u_1 = \frac{\partial w}{\partial y}, \quad u_2 = -\frac{\partial w}{\partial x}. \tag{2.11}$$

Załóżmy, że w_j jest aproksymacją funkcji w w przestrzeni $V_0^j([0,1]^2)$ i załóżmy, że znamy współczynniki rozwinięcia bazowego tej aproksymacji. Wiemy, że baza w przestrzeni funkcji dwóch zmiennych zbudowana jest z funkcji postaci $\phi^X_{j,k}(x)\phi^Y_{j,m}(y)$ dla $X, Y \in \{L, I, R\}$ oraz $k \in \Lambda^X_j, m \in \Lambda^Y_j$. W związku z tym funkcję w_j możemy zapisać jako:

$$w_j(x,y) = \sum_{X,Y\in\{L,I,R\}} \sum_{p\in\Lambda^X_j, q\in\Lambda^Y_j} w^{XY}_{j,(p,q)} \phi^X_{j,p}(x)\phi^Y_{j,q}(y)$$

Przy takich oznaczeniach, na mocy (2.11), aproksymacja funkcji u, którą będziemy oznaczać przez u_j ma postać:

$$u_j(x,y) = \sum_{X,Y\in\{L,I,R\}} \sum_{p\in\Lambda^X_j, q\in\Lambda^Y_j} w^{XY}_{j,(p,q)} \Phi^{XY}_{j,(p,q)}(x,y), \tag{2.12}$$

gdzie:

$$\Phi^{XY}_{j,(p,q)}(x,y) = \begin{bmatrix} \phi^X_{j,p}(x)(\phi^Y_{j,q})'(y) \\ -(\phi^X_{j,p})'(x)\phi^Y_{j,q}(y) \end{bmatrix} = 2^j \begin{bmatrix} \phi^X_{j,p}(x)((\phi^Y)')_{j,q}(y) \\ -((\phi^X)')_{j,p}(x)\phi^Y_{j,q}(y) \end{bmatrix}$$

2.3. Schemat iteracyjny i sformułowanie układu równań

W odróżnieniu od poprzednich podrozdziałów, gdzie rozważania były zupełnie „oderwane" od problemu Naviera–Stokesa i stanowiły ogólną konstrukcję użyteczną w wielu różnych sytuacjach, tak w dalszych rozważaniach będziemy już ściśle związani z równaniem:

$$-\nu\Delta u + (u\cdot\nabla)u + \nabla p = f \tag{2.13}$$

z warunkami:

$$\operatorname{div} u = 0 \text{ dla } x \in [0,1]^2, \tag{2.14}$$

$$u = 0 \text{ dla } x \in \partial[0,1]^2. \tag{2.15}$$

W powyższym równaniu niewiadomymi są funkcje u oraz p. W naszych rozważaniach zajmujemy się znalezieniem przybliżenia szukanej funkcji u. Funkcja p może zostać wyliczona później, korzystając z podobnych do opisanych tu metod, co zostało przedstawione na przykład w [13] i [11]. Dla zachowania prostoty zapisów w dalszych rozważaniach zakładamy $p = 0$.

Aby ułatwić dalsze rozważania definiujemy odwzorowania a, b odpowiednio dwu i trzyliniowe określone w następujący sposób:

$$a\colon X \times X \to \mathbb{R} \quad a(u,v) = \nu\langle \nabla u, \nabla v\rangle \tag{2.16}$$

$$b\colon X \times X \times X \to \mathbb{R} \quad b(u,v,w) = \langle (u\cdot\nabla)v, w\rangle \tag{2.17}$$

gdzie $X = H_0^1([0,1]^2)^2$.

Rozwiązanie naszego problemu (ze względu na niewiadomą u) przy tych oznaczeniach, przyjmując $V = \{v \in X : \operatorname{div} v = 0\}$, jest równoważne ze znalezieniem takiego u, że dla każdego $v \in V$ zachodzi:

$$a(u,v) + b(u,u,v) = \langle f, v\rangle. \tag{2.18}$$

Powyższe równanie wyraża w istocie układ nieskończenie wielu (nieprzeliczalnie wielu) równań. Korzystając z liniowości funkcjonałów a, b oraz iloczynu skalarnego można wykazać, że wystarczy rozwiązać te równania dla v stanowiących bazę przestrzeni V (a wiadomo, że w tej przestrzeni bazy są przeliczalne). W ten sposób otrzymujemy przeliczalnie wiele równań, w których niewiadomymi są współczynniki w rozwinięciu bazowym. Następnym krokiem jest skorzystanie z analizy wielorozdzielczej w V zdefiniowanej w poprzednim punkcie. Dla ustalonego $j \in \mathbb{Z}$ szukamy u_j takiego, że dla wszystkich $v_j \in V_j$:

$$a(u_j, v_j) + b(u_j, u_j, v_j) = \langle f, v_j\rangle \tag{2.19}$$

co znów równoważne jest rozwiązaniu równań tylko dla funkcji bazowych z V_j. Ponieważ dla dowolnego j przestrzenie V_j są skończenie wymiarowe, dochodzimy w ten sposób do układów skończenie wielu równań. Niestety wciąż są to równania algebraiczne nieliniowe ze względu na składnik $b(u_j, u_j, v_j) = \langle (u_j\cdot\nabla)u_j, v_j\rangle$ w którym, po zastąpieniu u_j przez rozwinięcie bazowe, szukane współczynniki będą występować w drugich potęgach. Zamiast rozwiązywania tych równań bezpośrednio (co mogłoby być trudne) proponujemy następujący schemat iteracyjny.

Ustalmy $j \in \mathbb{Z}$ i zdefiniujmy następujący ciąg funkcji $(u_j^\alpha)_{\alpha=0}^\infty \subset V_j$. Funkcja u_j^0 niech będzie rozwiązaniem równania:

$$a(u_j^0, v_j) = \langle f, v_j\rangle \quad \text{dla dowolnego } v_j \in V_j, \tag{2.20}$$

natomiast dla $\alpha > 0$ definiujemy u_j^α, jako rozwiązanie zagadnienia (dla dowolnego $v_j \in V_j$):

$$a(u_j^\alpha, v_j) + b(u_j^{\alpha-1}, u_j^\alpha, v_j) + b(u_j^\alpha, u_j^{\alpha-1}, v_j) = b(u_j^{\alpha-1}, u_j^{\alpha-1}, v_j) + \langle f, v_j\rangle. \tag{2.21}$$

Zauważmy, że wyliczenie poszczególnych u_j^α jest równoznaczne z rozwiązaniem skończenie wymiarowego układu równań liniowych.

Twierdzenie 2.2. *Niech f i ν spełniają warunek:*

$$\frac{c}{\nu^2}\|f\|_{-1} < 1, \tag{2.22}$$

dla pewnej stałej $c > 0$, gdzie $\|f\|_{-1} = \sup_{v\in X}\frac{\langle f,v\rangle}{\|\nabla v\|}$. Wtedy równanie (2.18) ma dokładnie jedno rozwiązanie $u \in V$ takie, że:

$$\|u\| \leqslant \frac{1}{\nu}\|f\|_{-1}.$$

Co więcej równanie (2.19) ma również dokładnie jedno rozwiązanie $u_j \in V_j$ które jest granicą ciągu $(u_j^\alpha)_\alpha$ zdefiniowanego w (2.20) i (2.21) oraz dla u_j również zachodzi:

$$\|u_j\| \leqslant \frac{1}{\nu}\|f\|_{-1}. \tag{2.23}$$

Dowód powyższego twierdzenia opiera się na rozważaniach zawartych w [12],[10].

Uwaga 2.3. Okazuje się, że wystarczy podać metodę szukania rozwiązania równania (2.13) w przypadku gdy $\nu = 1$. Jeśli bowiem poszukujemy rozwiązania równania:

$$-\nu\Delta u + (u\cdot\nabla)u = f, \tag{2.24}$$

gdzie $\frac{c}{\nu^2}\|f\|_{-1} < 1$ dla pewnego $c > 0$, to z twierdzenia 2.2 wiemy, że takie rozwiązanie istnieje i jest jedyne. Dla równania:

$$-\Delta u + (u\cdot\nabla)u = \tilde{f}, \tag{2.25}$$

w którym $\nu = 1$ i $\tilde{f} = \frac{f}{\nu^2}$ mamy wtedy:

$$c\left\|\tilde{f}\right\|_{-1} = c\cdot\sup_{v\in X}\frac{\langle\tilde{f},v\rangle}{\|\nabla v\|} = c\cdot\frac{1}{\nu^2}\|f\|_{-1} < 1.$$

Zatem na mocy twierdzenia 2.2 wynika, że również równanie (2.25) ma dokładnie jedno rozwiązanie. Oznaczmy je przez u_1. Wtedy dla $u_\nu = \nu\cdot u_1$ zachodzi:

$$-\nu\Delta u_\nu + (u_\nu\cdot\nabla)u_\nu = -\nu^2\Delta u_1 + \nu^2(u_1\cdot\nabla)u_1 = \nu^2(-\Delta u_1 + (u_1\cdot\nabla)u_1) = \nu^2\tilde{f} = f,$$

czyli u_ν jest rozwiązaniem równania (2.24).

W dalszej części tego podrozdziału zajmiemy się rozpisaniem równań definiujących ciąg u_j^αdla $\nu = 1$. Podamy metody wyliczania współczynników układu równań liniowych, których rozwiązanie będzie równoznaczne ze znalezieniem u_j^α. Opis ten rozbito na dwa przypadki $\alpha = 0$ i $\alpha > 1$ ze względu na definicję ciągu u_j^α.

2.3.1. Układ równań dla $\alpha = 0$

W tym i następnym punkcie wracamy do konstrukcji bazowej przedstawione w skrócie w podrozdziale 2.2. Poszukujemy funkcji $u_j^0 = (u_{j1}^0, u_{j2}^0)^2$ z przestrzeni V_j. Zgodnie z (2.12) istnieją liczby $u_{p,q}^{XY}$ takie, że:

$$u_{j1}^0(x,y) = \sum_{X,Y\in\{L,I,R\}} \sum_{p\in\Lambda_j^X, q\in\Lambda_j^Y} u_{p,q}^{XY} \phi_{j,p}^X(x)((\phi^Y)')_{j,q}(y) \tag{2.26}$$

$$u_{j2}^0(x,y) = -\sum_{X,Y\in\{L,I,R\}} \sum_{p\in\Lambda_j^X, q\in\Lambda_j^Y} u_{p,q}^{XY} ((\phi^X)')_{j,p}(x)\phi_{j,q}^Y(y) \tag{2.27}$$

Niech $f = (f_1, f_2)$, oraz dla $T, Z \in \{L, I, R\}$ i $m \in \Lambda_j^T, n \in \Lambda_j^Z$ niech

$$v_j = \begin{bmatrix} \phi_{j,m}^T(x)((\phi^Z)')_{j,n}(y) \\ -((\phi^T)')_{j,m}(x)\phi_{j,n}^Z(y) \end{bmatrix}.$$

Rozpisujemy teraz wzór (2.20) z wykorzystaniem powyższych oznaczeń otrzymujemy układ równań:

$$2^{2j} \sum_{X,Y\in\{L,I,R\}} \sum_{p\in\Lambda_j^X, q\in\Lambda_j^Y} \left(\Gamma_{p,m}^{XT}\Upsilon_{q,n}^{YZ} + 2\Omega_{p,m}^{XT}\Omega_{q,n}^{YZ} + \Upsilon_{p,m}^{XT}\Gamma_{q,n}^{YZ}\right) u_{p,q}^{XY} = F_{m,n}^{TZ},$$

gdzie $T, Z \in \{L, I, R\}$, $m \in \Lambda_j^T, n \in \Lambda_j^Z$. Poniżej zebrano definicje liczb $\Gamma, \Omega, \Upsilon, F$:

$$\begin{aligned}
\Gamma_{p,m}^{XT} &= \int_0^1 \phi_{j,p}^X(x)\phi_{j,m}^T(x)dx, \\
\Omega_{p,m}^{XT} &= \int_0^1 ((\phi^X)')_{j,p}(x)((\phi^T)')_{j,m}(x)dx, \\
\Upsilon_{p,m}^{XT} &= \int_0^1 ((\phi^X)'')_{j,p}(x)((\phi^T)'')_{j,m}(x)dx, \\
F_{m,n}^{TZ} &= \int_{[0,1]^2} \left[f_1(x,y)\phi_{j,m}^T(x)((\phi^Z)')_{j,n}(y) - f_2(x,y)((\phi^T)')_{j,m}(x)\phi_{j,n}^Z(y)\right] dxdy.
\end{aligned}$$

Zauważmy, że liczby Γ, Ω, Υ nie zależą w żaden sposób od parametrów układu, mogą więc zostać policzone raz (dla każdego ustalonego j), zapamiętane i używane do rozwiązywania różnych układów. W następnym rozdziale dodatkowo pokażemy, że współczynniki te nie są zależne od j i muszą być wyliczone tylko raz. W pracy [5] podano efektywny algorytm liczenia współczynników F. Liczby Γ, Ω, Υ można policzyć używając następującej metody. Po pierwsze definiujemy liczby:

$$\begin{aligned}
\Gamma_{p,m}^j &= \int_0^{2^j} \phi(x-p)\phi(x-m)dx, \\
\Omega_{p,m}^j &= \int_0^{2^j} \phi'(x-p)\phi'(x-m)dx, \\
\Upsilon_{p,m}^j &= \int_0^{2^j} \phi''(x-p)\phi''(x-m)dx.
\end{aligned}$$

[2]W tym miejscu zapis u_{j1}^0 nie podlega konwencji $u_{j,k} = 2^{j/2}u(2^j \cdot -k)$ – oznacza, że funkcja u_j^0 ma dwie funkcje współrzędne.

Następnie zauważamy, że wszystkie współczynniki w których $X = T = I$ wynoszą, na mocy twierdzenia o zamianie granic całkowania, dokładnie tyle co zdefiniowane wyżej liczby, bo na przykład:

$$\Gamma^{II}_{p,m} = \int_0^1 \phi^I_{j,p}(x)\phi^I_{j,m}(x)dx = \int_0^1 2^{j/2}\phi(2^j x - p)2^{j/2}\phi(2^j x - m)dx = \Gamma^j_{p,m},$$

Natomiast inne współczynniki możemy wyliczyć rozpisując funkcje $\phi^X_{j,p}$ ze wzoru (2.8). Mamy na przykład:

$$\begin{aligned}\Gamma^{LI}_{p,m} &= \int_0^1 \phi^L_{j,p}(x)\phi^I_{j,m}(x)dx = \int_0^1 2^j\phi^L_p(2^j x)\phi(2^j x - m)dx \\ &= \int_0^{2^j} \phi^L_p(x)\phi(x-m)dx = \int_0^1 \sum_{n=1}^{N-1} b^L_{p,n}\phi(x+n+a)\phi(x-m)dx \\ &= \sum_{n=1}^{N-1} b^L_{p,n}\Gamma^j_{-n-a,m}.\end{aligned}$$

2.3.2. Układ równań dla $\alpha > 0$

Załóżmy, że $\alpha > 0$ jest ustalone i że znamy już funkcję $u^\alpha_j = (u^\alpha_{1,j}, u^\alpha_{2,j})$, która dana jest przez:

$$\begin{aligned}u^\alpha_{1,j}(x,y) &= \sum_{V,W\in\{L,I,R\}} \sum_{r\in\Lambda^V_j, s\in\Lambda^W_j} u^{VW,\alpha}_{r,s}\phi^V_{j,r}(x)((\phi^W)')_{j,s}(y), \\ u^\alpha_{2,j}(x,y) &= -\sum_{V,W\in\{L,I,R\}} \sum_{r\in\Lambda^V_j, s\in\Lambda^W_j} u^{VW,\alpha}_{r,s}((\phi^V)')_{j,r}(x)\phi^W_{j,s}(y).\end{aligned}$$

Zakładamy że znamy współczynniki $u^{VW,\alpha}_{r,s}$. Szukaną funkcję $u^{\alpha+1}_j$ można zapisać w postaci:

$$\begin{aligned}u^{\alpha+1}_{1,j}(x,y) &= \sum_{X,Y\in\{L,I,R\}} \sum_{p\in\Lambda^X_j, q\in\Lambda^Y_j} u^{XY}_{p,q}\phi^X_{j,p}(x)((\phi^Y)')_{j,q}(y), \\ u^{\alpha+1}_{2,j}(x,y) &= -\sum_{X,Y\in\{L,I,R\}} \sum_{p\in\Lambda^X_j, q\in\Lambda^Y_j} u^{XY}_{p,q}((\phi^X)')_{j,p}(x)\phi^Y_{j,q}(y).\end{aligned}$$

Wstawiając powyższe do równania (2.21) otrzymujemy układ równań w którym nie-

wiadomymi są współczynniki $\{u_{p,q}^{XY}\}$:

$$2^{2j}\sum_{X,Y\in\{L,I,R\}}\sum_{p\in\Lambda_j^X,q\in\Lambda_j^Y}\left(\Gamma_{p,m}^{XT}\Upsilon_{q,n}^{YZ}+2\Omega_{p,m}^{XT}\Omega_{q,n}^{YZ}+\Upsilon_{p,m}^{XT}\Gamma_{q,n}^{YZ}\right)u_{p,q}^{XY}$$

$$+2^{2j}\sum_{X,Y\in\{L,I,R\}}\sum_{p\in\Lambda_j^X,q\in\Lambda_j^Y}\left\{\sum_{V,W\in\{L,I,R\}}\sum_{r\in\Lambda_j^V,s\in\Lambda_j^W}\left(\kappa_{r,p,m}^{VXT}\zeta_{q,s,n}^{YWZ}-\kappa_{p,r,m}^{XVT}\varepsilon_{s,n,q}^{WZY}\right.\right.$$

$$\left.\left.+\varepsilon_{r,m,p}^{VTX}\kappa_{q,s,n}^{YWZ}-\zeta_{p,r,m}^{XVT}\kappa_{s,q,n}^{WYZ}\right)u_{r,s}^{VW,\alpha}\right\}u_{p,q}^{XY}$$

$$+2^{2j}\sum_{X,Y\in\{L,I,R\}}\sum_{p\in\Lambda_j^X,q\in\Lambda_j^Y}\left\{\sum_{V,W\in\{L,I,R\}}\sum_{r\in\Lambda_j^V,s\in\Lambda_j^W}\left(\kappa_{p,r,m}^{XVT}\zeta_{q,s,n}^{YWZ}-\kappa_{r,p,m}^{VXT}\varepsilon_{q,n,s}^{YZW}\right.\right.$$

$$\left.\left.+\varepsilon_{p,m,r}^{XTV}\kappa_{s,q,n}^{WYZ}-\zeta_{r,p,m}^{VXT}\kappa_{q,s,n}^{YWZ}\right)u_{r,s}^{VW,\alpha}\right\}u_{p,q}^{XY}=\tilde{F}_{m,n}^{TZ},$$

gdzie $T,Z\in\{L,I,R\}$, $m\in\Lambda_j^T, n\in\Lambda_j^N$ oraz:

$$\kappa_{p,r,m}^{XVT}=2^{-j/2}\int_0^1((\phi^X)')_{j,p}(x)\phi_{j,r}^V(x)\phi_{j,m}^T(x)dx,$$

$$\zeta_{p,r,m}^{XVT}=2^{-j/2}\int_0^1((\phi^X)')_{j,p}(x)((\phi^V)')_{j,r}(x)((\phi^T)')_{j,m}(x)dx,$$

$$\varepsilon_{p,r,m}^{XVT}=2^{-j/2}\int_0^1((\phi^X)'')_{j,p}(x)((\phi^V)')_{j,r}(x)\phi_{j,m}^T(x)dx,$$

$$\tilde{F}_{m,n}^{TZ}=F_{m,n}^{TZ}+2^{2j}\sum_{X,Y\in\{L,I,R\}}\sum_{p\in\Lambda_j^X,q\in\Lambda_j^Y}\left\{\sum_{V,W\in\{L,I,R\}}\sum_{r\in\Lambda_j^V,s\in\Lambda_j^W}\left(\kappa_{p,r,m}^{XVT}\zeta_{q,s,n}^{YWZ}\right.\right.$$

$$\left.\left.-\kappa_{r,p,m}^{VXT}\varepsilon_{q,n,s}^{YZW}+\varepsilon_{p,m,r}^{XTV}\kappa_{s,q,n}^{WYZ}-\zeta_{r,p,m}^{VXT}\kappa_{q,s,n}^{YWZ}\right)u_{r,s}^{VW,\alpha}\right\}u_{p,q}^{XY,\alpha}.$$

Do wyliczenia współczynników κ,ζ,ε możemy wykorzystać podobną metodę do tej, którą stosowaliśmy w poprzednim punkcie. Definiujemy nowe współczynniki:

$$\kappa_{p,m,r}^{j}=\int_0^{2^j}\phi'(x-p)\phi(x-r)\phi(x-m)dx,$$

$$\zeta_{p,m,r}^{j}=\int_0^{2^j}\phi'(x-p)\phi'(x-r)\phi'(x-m)dx,$$

$$\varepsilon_{p,m,r}^{j}=\int_0^{2^j}\phi''(x-p)\phi'(x-r)\phi(x-m)dx,$$

i korzystamy ze wzoru (2.8). Na przykład mamy, że:

$$\kappa_{p,r,m}^{LLI}=\sum_{n,k=1}^{N-1}b_{p,n}^L b_{r,k}^L\kappa_{-n-a,-k-a,m}^{j},$$

oraz $\kappa_{p,r,m}^{III}=\kappa_{p,r,m}^{j}$.

Zauważmy, że współczynniki κ,ζ,ε nie zależą od prawej strony równania oraz od α, wobec czego mogą być wyliczone i zapamiętane tylko raz dla ustalonego j i następnie

używane do rozwiązywania różnych układów i do liczenia kolejnych iteracji. W następnym rozdziale pokażemy dodatkowo, że powyższe współczynniki nie zależą w istocie od wyboru j i mogą być policzone tylko raz i zapamiętane.

2.4. Oszacowanie błędu

Lemat 1. *Funkcjonał b zdefiniowany przez* (2.17) *spełnia:*

(a) dla dowolnego $u \in V$ i $v, w \in X$:

$$b(u, v, w) = -b(u, w, v), \tag{2.28}$$

(b) dla dowolnych $u, v, w \in X$ i pewnego $c \in \mathbb{R}$:

$$|b(u, v, w)| \leqslant c \|\nabla u\| \|\nabla v\| \|\nabla w\| \tag{2.29}$$

Lemat 2. *Dla dowolnego $v \in H^2([0,1]^2)^2 \cap V$ zachodzi:*

$$\inf_{v_j \in V_j} \|\nabla v - \nabla v_j\| \leqslant c 2^{-j} \|v\| .$$

Dowód Lematu 1 można znaleźć w [12],[10], a Lematu 2 w pracy [11].

Lemat 3. *Niech A, B, F, s_1, s_2 będą dowolnymi liczbami dodatnimi. Jeśli liczba $c \in \mathbb{R}$ spełnia: $c \geqslant \frac{(s_1+s_2)^2}{s_1}$, to:*

$$2A(s_1 + s_2)B \leqslant s_1 A^2 F + cB^2 F^{-1}$$

Dowód. Rozważmy trójmian kwadratowy:

$$s_1 A^2 F^2 - 2A(s_1 + s_2)BF + cB^2.$$

Z założenia o c wiemy, że wyróżnik tego trójmianu jest niedodatni:

$$\Delta = 4A^2(s_1 + s_2)^2 B^2 - 4s_1 A^2 cB^2 = 4A^2 B^2((s_1 + s_2)^2 - s_1 c) \leqslant 0.$$

Zatem dla dowolnego F mamy:

$$s_1 A^2 F^2 - 2A(s_1 + s_2)BF + cB^2 \geqslant 0.$$

Przenosząc drugi składnik na prawą stronę i dzieląc obie strony przez F (przy założeniu $F > 0$) otrzymujemy tezę. □

Twierdzenie 2.4. *Niech u będzie rozwiązaniem zagadnienia* (2.13)*, a u_j aproksymacją zdefiniowaną powyżej. Jeśli spełniony jest warunek istnienia i jednoznaczności* (2.22)*, to zachodzi:*

$$\|\nabla u - \nabla u_j\| \leqslant c 2^{-j} \left(1 + (1 - \frac{c}{\nu^2} \|f\|_{-1})^{-1}\right).$$

Dowód. Biorąc $v = v_j$ w równaniu (2.18) i odejmując je stronami od równania (2.19) dostajemy:

$$a(e, v_j) + b(e, u, v_j) + b(u_j, e, v_j) = a(v_j - u, v_j) + b(v_j - u, u, v_j) + b(u_j, v_j - u, v_j)$$

dla dowolnego $v_j \in V_j$, gdzie $e = v_j - u_j$. Do powyższego równania wstawmy e za v_j. Korzystając z (2.28) i faktu, że $a(e, e) = \nu \|\nabla e\|^2$ mamy:

$$\nu \|\nabla e\|^2 + b(e, u, e) = a(v_j - u, e) + b(v_j - u, u, e) + b(u_j, v_j - u, e). \tag{2.30}$$

Korzystając z (2.29) i twierdzenia o istnieniu i jednoznaczności, mamy:

$$|b(e, u, e)| \leqslant c \|\nabla u\| \|\nabla e\|^2 \leqslant \frac{c}{\nu} \|f\|_{-1} \|\nabla e\|^2 .$$

Będziemy teraz szacować drugą stronę równości (2.30). Korzystając z lematu 2 mamy:

$$|a(v_j - u, e) + b(v_j - u, u, e) + b(u_j, v_j - u, e)| \leqslant \|\nabla v_j - \nabla u\| (\nu + c \|\nabla u\| + c \|\nabla u_j\|) \|\nabla e\| \leqslant$$

Korzystając z (2.23) oraz z analogicznego twierdzenia dla u mamy dalej:

$$\leqslant \|\nabla v_j - \nabla u\| (\nu + c\frac{1}{\nu} \|f\|_{-1} + c\frac{1}{\nu} \|f\|_{-1}) \|\nabla e\| = 2 \|\nabla v_j - \nabla u\| (\frac{\nu}{2} + \frac{c}{\nu} \|f\|_{-1}) \|\nabla e\| \leqslant$$

Stosując lemat 3 podstawiając $A = \|\nabla e\|$, $B = \|\nabla v_j - \nabla u\|$, $s_1 = \frac{\nu}{2}$, $s_2 = \frac{c}{\nu} \|f\|_{-1}$, $F = (1 - \frac{c}{\nu^2} \|f\|_{-1})$ mamy:

$$\leqslant \frac{\nu}{2} \|\nabla e\|^2 (1 - \frac{c}{\nu^2} \|f\|_{-1}) + c' \|\nabla v_j - \nabla u\|^2 (1 - \frac{c}{\nu^2} \|f\|_{-1})^{-1},$$

dla c' takiego, że: $c' \geqslant \frac{(\frac{\nu}{2} + \frac{c}{\nu} \|f\|_{-1})^2}{\frac{\nu}{2}}$.

Możemy teraz przeformułować równość (2.30) tak aby otrzymać nierówność:

$$\nu \|\nabla e\|^2 \leqslant |a(v_j - u, e) + b(v_j - u, u, e) + b(u_j, v_j - u, e)| + |b(e, u, e)| \leqslant$$

Stosując otrzymane wyżej oszacowania mamy stąd dalej:

$$\leqslant \frac{\nu}{2}(1 - \frac{c}{\nu^2} \|f\|_{-1}) \|\nabla e\|^2 + c'(1 - \frac{c}{\nu^2} \|f\|_{-1})^{-1} \|\nabla u - \nabla v_j\|^2 + \frac{c}{\nu} \|f\|_{-1} \|\nabla e\|^2 .$$

Przenosząc składniki zawierające $\|\nabla e\|^2$ na lewą stronę mamy stąd:

$$(\nu - \frac{\nu}{2} + \frac{c}{2\nu} \|f\|_{-1} - \frac{c}{\nu} \|f\|_{-1}) \|\nabla e\|^2 \leqslant c'(1 - \frac{c}{\nu^2} \|f\|_{-1})^{-1} \|\nabla u - \nabla v_j\|^2 .$$

Upraszczając wyrażenia po obu stronach otrzymujemy:

$$(\frac{\nu}{2} - \frac{c}{2\nu} \|f\|_{-1}) \|\nabla e\|^2 \leqslant c'(1 - \frac{c}{\nu^2} \|f\|_{-1})^{-1} \|\nabla u - \nabla v_j\|^2 .$$

Dzielimy teraz obie strony nierówności przez $\frac{\nu}{2}(1 - \frac{c}{\nu^2} \|f\|_{-1})$ i obustronnie pierwiastkujemy. Mamy stąd:

$$\|\nabla e\| \leqslant c''(1 - \frac{c}{\nu^2} \|f\|_{-1})^{-1} \|\nabla u - \nabla v_j\| ,$$

gdzie $c'' > 0$ stała zależna od c' i ν. Korzystając z powyższego oszacowania na $\|\nabla e\|$ możemy szacować $\|\nabla(u - u_j)\|$:

$$\|\nabla(u - u_j)\| \leqslant \|\nabla(u - v_j)\| + \|\nabla e\| \leqslant \|\nabla(u - v_j)\| + c''(1 - \frac{c}{\nu^2}\|f\|_{-1})^{-1}\|\nabla(u - v_j)\| =$$

$$\|\nabla(u - v_j)\|\,(1 + c''(1 - \frac{c}{\nu^2}\|f\|_{-1})^{-1})$$

Ponieważ v_j było dowolnym elementem V_j więc na mocy lematu 2 mamy:

$$\|\nabla(u - u_j)\| \leqslant \tilde{c}2^{-j}\|u\|\,(1 + c''(1 - \frac{c}{\nu^2}\|f\|_{-1})^{-1}).$$

Z teorii równań różniczkowych wiadomo, że rozwiązanie u spełnia $\|u\| \leqslant C$ dla pewnego $C > 0$, ale to znaczy, że:

$$\|\nabla(u - u_j)\| \leqslant \tilde{c}2^{-j}C(1 + c''(1 - \frac{c}{\nu^2}\|f\|_{-1})^{-1}) \leqslant \mathbf{c}2^{-j}(1 + (1 - \frac{\mathbf{c}}{\nu^2}\|f\|_{-1})^{-1}). \qquad \square$$

ROZDZIAŁ 3

Implementacja

Niniejszy rozdział poświęcony jest opisowi implementacji algorytmów zaprezentowanych wcześniej w oparciu o zdefiniowaną w poprzednim rozdziale funkcję $\varphi = N_4(\cdot + 2)$. Opis podzielony jest na dwie części. W pierwszej opisujemy optymalizacje i uproszczenia, których dokonano w algorytmie. Dzięki odpowiednim modyfikacjom definicji współczynników w układzie równań udało się uniezależnić większość obliczeń od wyboru j oraz podać zależności, które znacznie ograniczają liczbę całek do policzenia. Pozwoliło to na zastosowanie w części obliczeń operacji na dokładnych liczbach wymiernych (w postaci ułamków zwykłych), co wpływa na poprawę dokładności. W drugiej części opisano szczegóły techniczne implementacji – w szczególności strukturę rozwiązania, typowe ścieżki wykonania, wykorzystane biblioteki oraz możliwości ponownego wykorzystania kodu. Pełny kod źródłowy aplikacji wraz z dodatkową dokumentacją znajduje się na dołączonej do pracy płycie CD oraz na stronie internetowej autora (`http://www.hope.art.pl/wavelets/`). Przykładowe wyniki wraz z czasem działania algorytmu znajdują się w rozdziale 4.

3.1. Uproszczenie i optymalizacja algorytmu

3.1.1. Zwiększenie dokładności obliczeń

Jednym z kluczowych problemów w implementacji algorytmów numerycznych jest tzw. stabilność numeryczna – wrażliwość algorytmu na błędy wynikające z niedokładności liczb zmiennopozycyjnych dostępnych w większości procesorów. Wstępne badania implementacji opartej o wbudowany typ liczbowy `double` pokazały, że w wielu przypadkach nawet operacje takie jak dodawanie i mnożenie wielomianów mogą być istotnie obciążone błędami zaokrągleń. Do przeprowadzenia tych badań wykorzystano program do obliczeń symbolicznych Maxima, który dysponuje dokładniejszą (niż wbudowana w procesor komputera) arytmetyką liczb zmiennopozycyjnych, oraz dokładną arytmetyką liczb wymiernych. Wszystkie operacje dodawania, mnożenia i całkowania funkcji, zawarte w prezentowanej implementacji - poza zwróceniem wyniku generowały fragment skryptu dla pakietu Maxima[1], który porównywał wynik uzyskany w programie z wynikiem Maximy. Ponieważ zaobserwowane rozbieżności były znaczne pojawiła się konieczność wykorzystania metody podobnej do tej stosowanej w Maxima, czyli zwiększenie dokładności

[1]W obecnej wersji mechanizmy te są wciąż obecne, jednak domyślnie są wyłączone.

poprzez zastosowanie specjalizowanych (obsługiwanych programowo a nie czysto sprzętowo) typów liczbocyh. Za pomocą biblioteki GMP[2] (GNU Multiple Precision Arithmetic Library) zaimplementowano dokładne wersje podstawowych operacji na wielomianach i funkcjach sklejanych. Oczywiście taka zmiana spowodowała znaczące obniżenie wydajności i zwiększenie zapotrzebowania na pamięć operacyjną. Należało więc dokonać wyboru miejsc, w których konieczne było zastosowanie liczb o dużej dokładności. W ostatecznej wersji programu zdecydowano się na zastosowanie ich do budowania kolejnych macierzy układów liniowych (dla kolejnych α). Po wyliczeniu macierze rzutowane są na typ `double` i układ rozwiązywany jest standardowymi metodami. Zastosowanie typów dokładnych do rozwiązania układu równań powoduje bowiem znaczne problemy wynikające z natury algorytmów iteracyjnych. Zapotrzebowania na pamięć operacyjną rosną z każdą iteracją, także otrzymanie wyników dla $\alpha > 3$ w rozsądnym czasie nie byłoby możliwe (liczniki i mianowniki liczb pojawiających się w tych obliczeniach potrafią mieć po kilkaset cyfr i z każdą iteracją się wydłużają!).

Możliwość zastosowania dokładnych obliczeń na liczbach wymiernych wynika ze szczególnej postaci stosowanej funkcji skalującej φ, która w swojej definicji ma tylko współczynniki wymierne. Narzuca to również ograniczenie na funkcję f występująca z prawej strony równania Naviera–Stokes. W prezentowanej implementacji musi ona być również wielomianem o współczynnikach wymiernych[3].

Poza zastosowaniem dokładniejszego typu danych liczbowych, cały algorytm został przerobiony w taki sposób aby wyeliminować wszelkie potencjalnie niewymierne składniki. W szczególności usunięto czynnik $2^{\frac{j}{2}}$ z definicji funkcji $\varphi_{j,k}$. Okazuje się bowiem, że zawsze mamy do czynienia albo z iloczynem dwóch takich funkcji – czyli potrzebujemy czynnik wymierny 2^j albo z iloczynem trzech funkcji pomnożonych dodatkowo właśnie przez $2^{\frac{j}{2}}$ co daje w rezultacie 2^{2j}.

3.1.2. Ograniczenie liczby całek

Po pierwsze zauważmy, że całki w definicjach liczb $\Gamma^j_{p,m}$, $\Omega^j_{p,m}$, $\Upsilon^j_{p,m}$, $\kappa^j_{p,m,r}$, $\zeta^j_{p,m,r}$, $\varepsilon^j_{p,m,r}$ w istocie nie zależą od j i równe są całce niewłaściwej na przedziale $[0, \infty)$, gdyż z założenia $p, m, r \leqslant 2^j - b$ i $b \geqslant 2$ czyli we wszystkich przypadkach nośniki funkcji pod całką są ograniczone z prawej strony przez $b + 2^j - b = 2^j$. Wobec czego wszystkie te funkcje są równe zero na przedziale $[2^j, \infty)$, tak więc dodanie tego przedziału nie zmienia wartości całki. Zależny od j pozostaje jedynie zakres liczb p, m, r ale same symbole przy ustalonych wartościach indeksów mają tą samą wartość. Będziemy więc pomijać górny indeks j.

[2]Więcej informacji o bibliotece można znaleźć na stronie: `http://www.gmplib.org/`.

[3]W ogólnym przypadku można najpierw aproksymować funkcję f wielomianem o współczynnikach wymiernych i następnie stosować prezentowaną metodę dla tej aproksymacji.

Po drugie zauważmy, że korzystając tylko z informacji o tym, że nośnik funkcji ϕ jest równy $[a, b]$, możemy przewidzieć z góry, które całki są równe zero. Mianowicie te, w których występuje iloczyn funkcji o rozłącznych nośnikach[4]. Ma to miejsce - w przypadku trzech pierwszych symboli, gdy wartość bezwzględna różnicy $|p - m|$ jest większa lub równa długości przedziału $[a, b]$ (która równa jest N), a w przypadku trzech pozostałych, gdy dowolna z liczb $|p - m|$, $|m - r|$, $|p - r|$ jest większa lub równa N.

Po trzecie w oczywisty sposób wartości współczynników Γ, Ω, Υ, ζ nie zależą od kolejności parametrów p, m, r, natomiast κ nie zależy od kolejności parametrów m, r.

Po czwarte w końcu, łatwo pokazać, że wszystkie współczynniki są niewrażliwe na przesunięcia parametrów – to znaczy np. $\Gamma_{p,m} = \Gamma_{p+1,m+1} = \Gamma_{p+k,m+k}$. Reasumując – w przypadku współczynników z dwoma parametrami wartość zależy tylko od liczby $|p - m|$. Jak już powiedzieliśmy wcześniej jeśli $|p - m| \geqslant N$ to współczynnik równy jest zero. Musimy więc policzyć jedynie N współczynników Γ, Ω, Υ – na przykład dla następujących par parametrów (p, m): $\{(0, 0), (0, 1), \ldots, (0, N - 1)\}$. Podobnie w przypadku współczynników z trzema parametrami. Dla ζ musimy wyliczyć wartości jedynie dla parametrów ze zbioru $\{(0, m, r) : 0 \leqslant m \leqslant r < N\}$ – mamy więc $\frac{N(N+1)}{2}$ możliwości. Dla κ jest nieco więcej możliwości – musimy bowiem wyliczyć wartości dla parametrów ze zbioru $\{(p, 0, r) : 0 \leqslant p, r < N\}$ – mamy więc N^2 możliwości. Najmniej „symetryczny" współczynnik wymaga od nas policzenia N^3 całek.

Podsumowując – mamy do policzenia dokładnie $3.5N + 1.5N^2 + N^3$ całek, co w przypadku naszej funkcji φ (gdzie $N = 4$) równe jest 102. Dla porównania w oryginalnym algorytmie zaczerpniętym z pracy [16], dla $j = 3$ mamy 756 całek, dla $j = 4$ jest ich 8820, a dla $j = 5$ mamy 83700 całek. Ogólnie jest ich $3(2^j - 2)^2 + 3(2^j - 2)^3$. Widać więc, że zysk płynący z opisanych optymalizacji jest znaczny.

W prezentowanej tu implementacji całki potrzebne do wyznaczenia macierzy głównej układu równań nie są liczone za pomocą symboli opisanych powyżej. Stosujemy podobną metodę polegającą na „rozłożeniu" każdej z funkcji bazowych na składowe wielomiany (przypomnijmy, że nasze funkcje bazowe składają się zawsze z trzech lub czterech wielomianów na odpowiednich przedziałach) i rozbiciu całek na sumy w których występują iloczyny konkretnych wielomianów. Następnie, stosując liniową zamianę zmiennych, przesuwamy wszystkie przedziały całek do zera (podobnie jak w powyższej metodzie ustaliliśmy jeden z indeksów na 0) i liczymy całki ze wszystkich unikalnych iloczynów tego typu a następnie sumujemy odpowiednie składniki tak aby uzyskać poszukiwane wyrazy macierzy. Metoda taka prowadzi do takich samych oszczędności i okazała się nieco wygodniejsza w implementacji. Jest jednak trudniejsza do dokładnego opisania i zrozumienia oraz w odróżnieniu od opisanego wyżej sposobu, nie uogólnia się łatwo na przypadek

[4]Rozłączność w sensie $\left|\bigcap_{t \in T} \operatorname{supp} f_t\right| = 0$.

ogólnej, dowolnej funkcji ϕ (która nie musi przecież koniecznie być funkcją kawałkami wielomianową).

Możliwe jest dalsze zmniejszenie liczby całek poprzez dokładniejsze wykorzystanie faktu, że rozważana funkcja φ jest parzysta oraz uogólnienie tych metod dla kolejnych bazowych funkcji giętych (wyższych rzędów). Prace nad dalszą optymalizacją i rozbudową algorytmu są przedmiotem dalszych badań autora.

Z powyższych rozważań wynika, że liczba unikalnych wartości symboli występujących we wzorach definiujących układy równań jest stała (niezależna od j). Stąd wynika, że również stała jest liczba możliwych iloczynów i sum tych symboli (por. sformułowanie układu równań w rozdziale 2). Okazuje się, że istnieją metody zdefiniowania macierzy tych wyznaczników (bądź odpowiednich ich kombinacji) w sposób rekurencyjny – tak, że macierz dla $j+1$ zależy od macierzy dla j. Co więcej zależność ta sprowadza się jedynie do przepisywania odpowiednich liczb i wypełniania „pustych miejsc" zerami. Co więcej otrzymane macierze będą charakterystycznymi macierzami pasmowymi (macierz pasmowa to uogólnienie pojęcia macierzy diagonalnej i trójdiagonalnej, gdzie niezerowe wyrazy skupione są w pewnym pasie o ustalonej szerokości, wokół przekątnej). Mówiąc nieco obrazowo, przejście od macierzy wyliczonej dla j do macierzy dla $j+1$ polega na „rozciągnięciu środka (pasma) macierzy wejściowej. W prezentowanej implementacji nie korzystamy w pełni z tych zależności. Tablicujemy jedynie wszystkie możliwe, unikalne wartości parametrów i rozmieszczamy je w macierzy oddzielnie dla każdego j, stąd pomijamy również szczegółowy opis tych zależności.

3.2. Szczegóły techniczne rozwiązania

Razem z pracą dostarczono implementację omawianych algorytmów napisaną w obiektowym języku programowania C# kompilowanym i uruchamianym w ramach platformy Microsoft .NET[5]. Rozwiązanie podzielone jest na dwie części. Pierwsza – ważniejsza, to biblioteka o nazwie WaveletNavierStokesSolver, w której mieści się właściwa implementacja. Druga składa się z przykładów zastosowania tej biblioteki do szacowania błędu aproksymacji, generowania wykresów i wydruków testowych. W tym punkcie skupimy się na opisie pierwszej części, czyli biblioteki klas implementujących algorytm opisany w rozdziale 2 i zoptymalizowany zgodnie z opisem z poprzedniego podrozdziału. Przykłady zastosowań programów z drugiej części znajdują się w rozdziale 4.

[5]Do uruchomienia opisywanych programów użytkownik musi mieć zainstalowane środowisko uruchomieniowe .NET Framework w wersji 3.5. Jest ono dostępne dla systemów Windows jako pakiet do ściągnięcia ze strony firmy Microsoft (`http://msdn.microsoft.com/en-us/netframework/default.aspx`) lub za pomocą zintegrowanego systemu aktualizacji Microsoft Update. W innych systemach można skorzystać z platformy Mono (`http://go-mono.com/`) - autor nie testował jednak oprogramowania w systemach innych niż Windows.

Prezentowana biblioteka składa się z następujących klas[6]:

- `Base`, w której wyliczane są funkcje bazowe różnego typu (lewe, środkowe, prawe);
- `BiPolynomial`, która reprezentuje wielomiany dwóch zmiennych (do swojego działania klasa ta korzysta z klasy `Index2` odpowiadającej matematycznemu pojęciu dwuwskaźnika $\alpha = (\alpha_1, \alpha_2)$);
- `Equation`, w której jest budowany i rozwiązywany układ równań;
- `Interval`, która odpowiada przedziałom (np. liczbowym) – klasa ta jest klasą generyczną, czyli obsługuje zarówno przedziały liczbowe, jak i przedziały dowolnych innych obiektów, które można porównywać ze sobą (muszą one implementować interfejs `IComparable`);
- `MatrixBlocks`, która używana jest przez `Equation` przy budowaniu macierzy głównych w układzie równań – właśnie tu zaimplementowano uproszczone liczenie całek opisane w poprzednich punktach;
- `Polynomial`, która implementuje operacje na wielomianach jednej zmiennej;
- `Spline`, która implementuje funkcje jednej zmiennej kawałkami wielomianowe (określone na przedziałach) – nie jest to pełna implementacja pojęcia funkcji giętych (ang. spline), gdyż nie są sprawdzane warunki czy dana funkcja jest gięta w sensie definicji 1.8 (w szczególności czy jest ciągła i odpowiednio gładka), co jednak nie przeszkadza w jej stosowaniu w prezentowanym algorytmie, gdyż sami z góry zapewniamy, że wszystkie funkcje są „poprawnymi" funkcjami giętymi.

W skład rozwiązania wchodzą też komponenty zewnętrzne: wspomniana już biblioteka GMP wraz z obiektowym „opakowaniem"[7] oraz prosta biblioteka rozwiązująca liniowe układy równań (zawarta w klasie `Matrix`) oparta na programie zamieszczony w serwisie internetowym CodeProject[8]. Mechanizm z klasy `Matrix` można prosto zastąpić bardziej zaawansowanym algorytmem. W tym celu wystarczy zaimplementować interfejs `ILinearSolver` składający się z jednej metody **Solve** i dostarczyć egzemplarz takiej klasy do konstruktora `Equation`.

[6] Celowo pominięto kilka klas narzędziowych, które z punktu widzenia naszych rozważań są mało istotne – implementują na przykład interfejsy pozwalające porównywać pewne obiekty i są wykorzystywane wewnętrznie przez platformę uruchomieniową.

[7] W ramach prac nad biblioteką autor stworzył i opublikował w sieci na zasadach „open source" interfejs obiektowy do biblioteki GMP dla platformy .NET. Projekt rozwijany jest zespołowo i dostępny jest w Internecie na stronie `http://www.codeplex.com/gnumpnet/`.

[8] VijayaSekhar Gullapalli. *Library for linear algebra methods in C#.NET and it's application to computational heat transfer*. CodeProject, 2005, `http://www.codeproject.com/KB/recipes/matrixoperations.aspx`.

W następnych punktach opisano działanie poszczególnych klas biblioteki i powiązania między nimi.

3.2.1. Operacje podstawowe – klasy `Polynomial`, `Spline`, `BiPolynomial`

Zaczynamy od opisu podstawowych klas stanowiących bazę do implementacji samego algorytmu. Wszystkie trzy opisywane tu klasy, umożliwiają proste operowanie na podstawowych typach funkcji. W dalszej części tekstu pisząc „funkcja" mamy na myśli sens matematyczny tego pojęcia (funkcją jest wielomian, funkcja gięta itd.). Jeśli chodzi nam o funkcję w sensie języku programowania, zawsze używać będziemy słowa metoda. Będziemy utożsamiać poszczególne klasy z pojęciami, które opisują – kiedy piszemy wielomian mamy więc na myśli klasę `Polynomial`, funkcji giętej lub funkcji sklejanej odpowiada klasa `Spline`, a wielomianom dwóch zmiennych klasa `BiPolynomial`.

Podstawy

Wielomiany jednej zmiennej to w istocie jednowymiarowa tablica liczb typu `Rational` (liczby wymierne) odpowiadająca współczynnikom wielomianu, wraz z zestawem operacji na tej tablicy, odpowiadających typowym operacjom na wielomianach. Stworzenie egzemplarza klasy `Polynomial` wymaga podania tablicy współczynników (niepodanie jej skutkuje utworzeniem wielomianu zerowego).

Funkcje gięte są w istocie zbiorem powiązanych ze sobą przedziałów i wielomianów, zaimplementowanym za pomocą wbudowanej w platformę .NET klasy słownika – `Dictionary`. Aby utworzyć taki obiekt musimy dostarczyć do konstruktora przygotowany słownik (w istocie może to być dowolny obiekt implementujący odpowiedni interfejs `IEnumerable`). Przyjmuje się, że poza zadanymi przedziałami funkcja gięta przyjmuje pewną, zadaną z góry wartość (domyślnie jest to zero).

Wielomiany dwóch zmiennych, tak jak wspomniano wcześniej opierają się na wielowskaźnikach typu `Index2` i są oparte na słowniku przypisującym danemu wielowskaźnikowi współczynnik przy odpowiednim jednomianie. Utworzenie wielomianu wielu zmiennych, trochę podobnie jak w przypadku funkcji giętych, polega na dostarczeniu kolekcji implementującej ogólny interfejs `IEnumerable<KeyValuePair<Index2, Rational>>` – w szczególności może to być słownik `Dictionary<Index2,Rational>`.

Wartość funkcji w punkcie

Liczenie wartości funkcji w danym punkcie jest z punktu widzenia wykorzystania z zewnątrz takie samo w przypadku wszystkich typów funkcji. Każda z funkcji implementuje tzw. indeksator (ang. indexer), oryginalnie wprowadzony z myślą o dostępie do obiektów jak do tablic. W naszym przypadku pozwala to jednak uzyskać bardzo czytelny zapis:

```
var f = new Polynomial(new[] {new Rational(1,2),0,1,new Rational(5,8)});
var x = new Rational(2,3);
var y = f[x];
```

Jeśli więc `f` jest wielomianem (lub innym typem funkcji) to `f[x]` jest wartości tej funkcji w punkcie `x`.

Funkcje gięte definiują dodatkowo podobny ineksator, który dla danego przedziału zwraca funkcję odpowiadającą temu przedziałowi, czyli jeśli `s` jest funkcją giętą, a `I` jest przedziałem to `s[I]` jest wielomianem (jeśli w definicji `s` nie pojawił się przedział `I` to zwracany jest wielomian stały, równy domyślnej wartości funkcji giętej).

Operacje arytmetyczne, składanie, przesunięcia i skalowanie

Wszystkie typy funkcji obsługują operacje dodawania, odejmowania, mnożenia i mnożenia przez stałą. Dzięki implementacji odpowiednich operatorów mając dane dwie funkcje tego samego typu `f,g` możemy po prostu napisać `f+g` i otrzymujemy nową funkcję będący sumą dwóch podanych.

Dodatkowo wielomiany jednej zmiennej posiadają metodę **Composite** służącą do wyliczenia złożenia jednego wielomianu z drugim $f(g(x))$. Funkcje gięte i wielomiany mają też, zaimplementowane na tej podstawie, metody **Scale** i **Shift** służące odpowiednio do wyliczenia funkcji postaci $f(ax)$ oraz $f(x+a)$. W przypadku funkcji giętych, poza poszczególnymi wielomianami, przekształceniu ulegają oczywiście również przedziały na których funkcja jest określona.

Różniczkowanie i całkowanie

Dla wielomianów i funkcji giętych zdefiniowano metodę **D**, która zwraca pochodną (która w przypadku wielomianu jest nowym wielomianem a w przypadku funkcji giętej jest nową funkcją tego typu, określoną na tych samych przedziałach co wejściowa, gdzie każda funkcja składowa została zróżniczkowana). Dodatkowo istnieje wersja metody **D** z parametrem określającym krotność pochodnej (zamiast pisać `f.D().D()` możemy napisać `f.D(2)`).

Wielomiany wielu zmiennych zamiast wspomnianej metody **D** posiadają metodę **Partial** która liczy pochodną cząstkową względem zadanej zmiennej.

Wszystkie typy funkcji posiadają metodę **Primitive** wyznaczającą funkcję pierwotną (w przypadku wielu zmiennych trzeba podać zmienną, względem której liczymy). Na jej podstawie zdefiniowano metody **Integral** liczące całkę oznaczoną na zadanym przedziale (w przypadku wielomianu dwóch zmiennych potrzebna jest tablica dwóch przedziałów).

Dodatkowo dla wielomianów dwóch zmiennych zdefiniowano operator projekcji. Metoda **Projection** zwraca nowy wielomian `BiPolynomial`, w którym za podaną zmienną o numerze `v` podstawiono zadaną wartość liczbową `x`.

Konwersje

Istnieje możliwość konwersji (rzutowania) wielomianu jednej zmiennej na wielomian dwóch zmiennych. Służy do tego metoda **CastToBiPolynomial** zdefiniowana w klasie `Polynomial`. Jako argument przyjmuje ona numer zmiennej, który ma być przypisany jedynej zmiennej z wejściowego wielomianu. Konieczność wskazania zmiennej, do której ma być przypisana jedyna zmienna z wielomianu jednej zmiennej uniemożliwia zastosowanie prostszego podejścia z przeciążonym operatorem rzutowania.

3.2.2. Baza funkcyjna – klasa `Base`

Klasa `Base` definiuję bazę funkcji (składającą się z funkcji giętych) i jej główną funkcjonalnością jest dostarczenie indeksatorów pozwalających pobrać daną funkcję bazową. Należy przy tym zaznaczyć, że w prezentowanej bibliotece nie korzystamy z oznaczeń z rozdziału 2 typu $\varphi^X_{j,p}$. Wszystkie potrzebne funkcje bazowe stanowią tablicę, na której początku znajduje się funkcja „lewa", następnie kolejne przesunięcia funkcji „środkowej" i na końcu funkcja „prawa". Dzięki temu cała baza ma jednorodną numerację: $0, \ldots, 2^j - 2$. Dla zachowania możliwości programowej weryfikacji wzorów z rozdziału 2 stworzono również enumeracja `FType` (zawierająca wartości Left, Internal, Right) i odpowiedni indeksator, który jest bezpośrednim tłumaczeniem wspomnianego oznaczenia $\varphi^X_{j,k}$. W sumie dostępne są trzy indeksatory:

public `Spline` **this[int** `i`**, int** `n`**]**
public `Spline` **this[**`FType type`**, int** `j`**, int** `k`**]**
public `Spline` **this[**`FType type`**, int** `j`**, int** `k`**, int** `n`**]**

Pierwszy z nich umożliwia dostęp do i-tej funkcji w bazie o krotności pochodnej n (możliwe wartości to $0, 1, 2$). Drugi indeksator odpowiada oznaczeniu $\varphi^X_{j,k}$ gdzie X oznaczono przez *type*. Trzeci indeksator dodatkowo pozwala podać krotność pochodnej n (możliwe są dowolne nieujemne wartości). Należy dodatkowo zaznaczyć, że w momencie tworzenia egzemplarza klasy `Base`, wyliczane są i zapamiętywane wszystkie funkcje bazowe wraz z ich pierwszymi i drugimi pochodnymi, czyli pierwszy indeksator zwraca po prostu dane zapisane w tablicy.

Warto również zaznaczyć, że funkcje lewe i prawe nie są wyliczane krok po kroku z funkcji wewnętrznej tak jak to pokazano na początku rozdziału 2. Zamiast tego w kodzie

klasy `Base` wpisano na stałe wzory (2.9) i (2.10), które jednoznacznie definiują funkcje lewe i prawe.

3.2.3. Sformułowanie i rozwiązanie układu równań

Zadaniem klasy `Equation` jest zbudowanie macierzy głównej i wektora wyrazów wolnych zgodnie z definicjami z rozdziału 2 a następnie przekazanie ich do dostarczonej klasy implementującej interfejs `ILinearSolver`, która rozwiązuje liniowy układ równań. Standardowo jest to klasa oparta o wspomniany wcześniej kod klasy `Matrix`, jednak może zostać łatwo zastąpiona przez inny specjalizowany system. Dodatkowo klasa `Equation` odpowiada za zapamiętywanie wyliczonych rozwiązań (dla różnych α). Dostęp do rozwiązań uzyskujemy przez indeksator, który dla podanej α zwraca zapamiętaną macierz współczynników bazowych rozwiązania (o ile jest już wyliczona), lub wywołuje metodę poszukującą rozwiązania, zapamiętuje wynik i zwraca go. Rozwiązanie zwracane jest jako macierz kwadratowa odpowiadająca notacji $u_{p,q}$ z podrozdziałów 2.3.1 i 2.3.2.

Dodatkowo, aby podnieść wydajność, w klasie `Equation` gromadzone są (celem wielokrotnego wykorzystania) liczby:

$$\begin{aligned}
g_{m,n,p,q} =& \Gamma^{XT}_{p,m}\Upsilon^{YZ}_{q,n} + 2\Omega^{XT}_{p,m}\Omega^{YZ}_{q,n} + \Upsilon^{XT}_{p,m}\Gamma^{YZ}_{q,n},\\
f_{m,n,p,q,r,s} =& \kappa^{XVT}_{p,r,m}\zeta^{YWZ}_{q,s,n} - \kappa^{VXT}_{r,p,m}\varepsilon^{YZW}_{q,n,s} + \varepsilon^{XTV}_{p,m,r}\kappa^{WYZ}_{s,q,n} - \zeta^{VXT}_{r,p,m}\kappa^{YWZ}_{q,s,n},\\
a_{m,n,p,q,r,s} =& \kappa^{VXT}_{r,p,m}\zeta^{YWZ}_{q,s,n} - \kappa^{XVT}_{p,r,m}\varepsilon^{WZY}_{s,n,q} + \varepsilon^{VTX}_{r,m,p}\kappa^{YWZ}_{q,s,n} - \zeta^{XVT}_{p,r,m}\kappa^{WYZ}_{s,q,n} + f_{m,n,p,q,r,s}.
\end{aligned}$$

Liczby te wyliczane są za pomocą dokładnych typów z biblioteki GMP a następnie rzutowane są na wbudowany typ `double` i dalej używane są w tej formie. Przyglądając się wyżej zdefiniowanym liczbom oraz układom równań z rozdziału drugiego łatwo zauważyć, że układ dla $\alpha = 0$ ma teraz postać:

$$2^{2j}\sum_{X,Y\in\{L,I,R\}}\sum_{p\in\Lambda_j^X,q\in\Lambda_j^Y} g_{m,n,p,q}u^{XY}_{p,q} = F^{TZ}_{m,n},$$

a dla $\alpha > 0$ układ ma postać:

$$2^{2j}\sum_{X,Y\in\{L,I,R\}}\sum_{p\in\Lambda_j^X,q\in\Lambda_j^Y}\left(g_{m,n,p,q} + \sum_{V,W\in\{L,I,R\}}\sum_{r\in\Lambda_j^V,s\in\Lambda_j^W} a_{m,n,p,q,r,s}u^{VW,\alpha}_{r,s}\right)u^{XY}_{p,q} = \tilde{F}^{TZ}_{m,n},$$

przy czym $\tilde{F}$ przyjmuje uproszczoną postać:

$$\tilde{F}^{TZ}_{m,n} = F^{TZ}_{m,n} + 2^{2j}\sum_{X,Y\in\{L,I,R\}}\sum_{p\in\Lambda_j^X,q\in\Lambda_j^Y}\left\{\sum_{V,W\in\{L,I,R\}}\sum_{r\in\Lambda_j^V,s\in\Lambda_j^W} f_{m,n,p,q,r,s}u^{VW,\alpha}_{r,s}\right\}u^{XY,\alpha}_{p,q}.$$

Zastosowanie takiego podejścia skraca czas obliczeń kosztem większego zużycia pamięci operacyjnej. Okazuje się jednak, że dla $j > 4$ rozmiar potrzebnej pamięci przekracza możli-

wości typowej architektury 32-bitowej[9]. W związku z tym mechanizm zaimplementowany w programie działa następująco. Najpierw próbuje zaalokować odpowiedni blok pamięci. Jeśli się to uda to wylicza wszystkie wartości i zapamiętuje je. Jeśli jednak nie jest to możliwe ustawiana jest specjalna flaga, która informuje dalszy kod, że zamiast korzystać ze zmagazynowanych wyników należy stosować procedury liczące liczby g, f, a zawsze wtedy gdy trzeba. Ograniczenia co do możliwości alokacji ciągłych bloków pamięci są dużo mniej restrykcyjne na platformach 64-bitowych. Niestety autor nie miał okazji do przetestowania implementacji na takich systemach – jednak dzięki zastosowanemu podejściu mamy pewność, że w tych systemach mechanizm magazynowania będzie działał dla wszystkich j dla których to możliwe.

Liczby g, f, a wyznaczane są w odpowiednich metodach (które właściwie przepisują powyższe definicje), które korzystają z wyników uzyskanych przez klasę `MatrixBlocks`. W klasie tej tablicujemy wartości sześciu współczynników Γ, Ω, Υ, κ,ζ, ε. Wartości liczone są w oparciu o zoptymalizowany algorytm, podobny do tego, który opisano w punkcie 3.1.2. Współczynniki Γ, Ω, Υ są do siebie bardzo podobne (różnią się tylko rzędem pochodnej) w związku z czym liczone są w jednej metodzie **bigG**. Metoda **smallG** odpowiedzialna jest za wyznaczenie wszystkich unikalnych wartości całek występujących w definicji współczynników. Następnie w metodzie **bigG** są one odpowiednio rozmieszczone w tablicy (zgodnie z opisanymi pokrótce zasadami symetrii). Podobnie jest w przypadku pozostałych współczynników. Każdemu z nich odpowiada metoda **small***, która wyznacza wszystkie unikalne wartości, oraz metoda **big*** która rozmieszcza współczynniki w tablicy. Wszystkie metody w tej klasie korzystają z narzędziowych klas `MyArray`, które są rozszerzeniem zwykłych tablic o funkcjonalność wartości domyślnej (jeśli indeksy są poza zakresem nie jest generowany błąd - tylko zwracana jest wartość domyślna – w naszym przypadku jest to zero)[10].

Zgodnie z wstępnymi rozważaniami zawartymi w punkcie 3.1.2 możliwe jest udoskonalenie klasy `MatrixBlocks` tak aby na podstawie macierzy dla j można było uzyskać macierz dla $j + 1$ tylko poprzez kopiowanie i odpowiednie umiejscowienie elementów. Jest to przedmiotem dalszych prac nad rozwojem biblioteki prowadzonych przez autora. Informacje o postępach prac i publikacji osiągniętych wyników można będzie znaleźć na stronie internetowej `http://www.hope.art.pl/wavelets`.

[9] Problemem jest limit na długość ciągłego bloku pamięci (korzystamy z wbudowanych w platformę tablic wielowymiarowych) jaką może zaalokować proces w systemach Win32.

[10] Dodatkowo tablice typu `MyArray` numerują elementy od 1 a nie jak się to przyjęło w językach wywodzących się z C od 0.

ROZDZIAŁ 4

Wyniki

W tej części pracy prezentujemy zestawienie wyników uzyskanych poprzez opisaną wcześniej implementację. Aproksymować będziemy rozwiązanie układu dla którego znamy również rozwiązanie dokładne – tak aby można było policzyć różnicę między przybliżeniem a wynikiem dokładnym. Rozwiązaniem tym jest:

$$u_1(x,y) = 20x^2(x-1)^2y(y-1)(2y-1), u_2(x,y) = -20x(x-1)(2x-1)y^2(y-1)^2.$$

Łatwo sprawdzić, że takie $u = (u_1, u_2)$ spełnia warunki $\operatorname{div} u = 0$ na $[0,1]^2$ oraz $u = 0$ na $\partial[0,1]^2$. Wstawiając u do równania możemy wyznaczyć funkcję $f = (f_1, f_2)$ stojącą po prawej stronie. Ma ona postać:

$$\begin{aligned} f_1(x,y) &= 400(x-1)^3x^3\,(2x-1)\,(y-1)^2y^2\left(2y^2-2y+1\right) \\ &\quad + 40\,(2y-1)\left(\left(-6x^2+6x1-1\right)(y-1)\,y - 3(x-1)^2x^2\right) \\ f_2(x,y) &= 40\,(2x-1)\left((x-1)\,x\left(6y^2-6y+1\right) + 3(y-1)^2y^2\right) \\ &\quad + 400(x-1)^2x^2\left(2x^2-2x+1\right)(y-1)^3y^3\,(2y-1) \end{aligned}$$

W skład rozwiązania poza biblioteką zawierająca sam algorytm wchodzą dwa programy testowe korzystające z tej biblioteki. W programach tych wpisany jest na stałe, powyższy układ równań. Bardzo łatwo można go zmodyfikować – zmieniając treść metody **getU** w plikach *Program.cs* w odpowiednich katalogach (katalogi są takie jak nazwy programu).

Program o nazwie *Error* liczy błąd względny aproksymacji zdefiniowany:

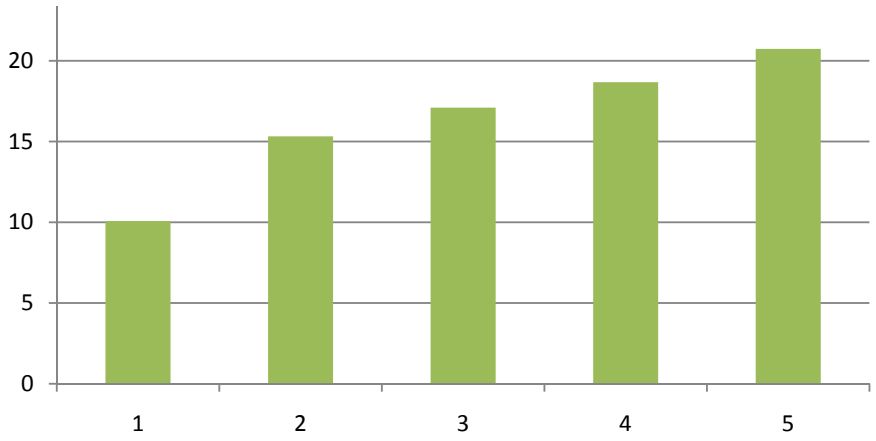
$$e_j^\alpha = \frac{\left\|\nabla(u-u_j^\alpha)\right\|}{\|\nabla u\|},$$

gdzie u_j^α to aproksymacja, natomiast u to rozwiązanie. Program uruchamia się z linii poleceń podając dwa argumenty. Pierwszym z nich jest poziom rozdzielczości j a drugi to liczba α_1. Program liczy błędy e_j^α dla $\alpha = 0, 1, \ldots, \alpha_1 - 1$. Wyniki wyświetlane są w formie α_1 linii tekstu. W każdej linii podana jest aktualna wartość α oraz wartość błędu. Po wypisaniu wyników program wyświetla czas swojego działania.

W tabeli 4.1 zamieszczono wyniki działania programu *Error*. Na ich podstawie możemy zaobserwować kilka ważnych rzeczy. Po pierwsze – wraz ze wzrostem j błąd maleje. Jest to zgodne z oczekiwaniami – pod koniec rozdziału 2 pokazaliśmy bowiem, że metoda jest zbieżna do rozwiązania. Po drugie, wraz ze wzrostem α dla ustalonego j błąd się stabilizuje. Co więcej stabilizacja następuje szybko (dla $j = 5$ różnica między wynikiem dla $\alpha = 0$ a

$\alpha \backslash j$	3	4	5
0	0.0139	0.0035	0.0012
1	0.0541	0.0047	0.0012
2	0.0660	0.0048	0.0012
3	0.0666	0.0048	0.0012
4	0.0666	0.0048	0.0012

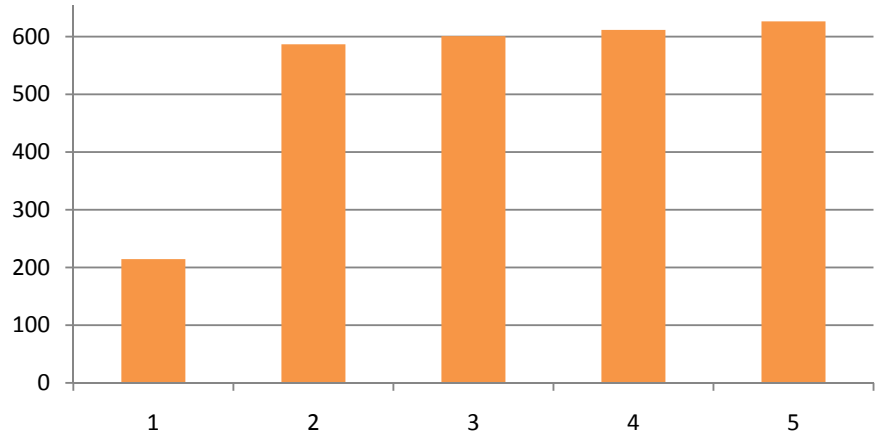
Tabela 4.1. Wyniki uzyskane z programu *Error* zaokrąglone do 4 miejsc po przecinku.



Rysunek 4.1. Czas (w sekundach) wykonania polecenia „*Error* 3 α_1" dla $\alpha_1 = 1, \ldots, 5$.

$\alpha = 1$ pojawiła się dopiero na 7 miejscu po przecinku). Jest to ważna informacja gdyż tak naprawdę nie mamy metody stwierdzenia jak duże powinno być α aby przyjąć otrzymany wynik. Stabilizacja pozwala napisać program, który przerywa liczenie w momencie gdy kolejna iteracja wnosi niewiele zmian. Trzecią, nieco zaskakującą informacją jest fakt, że początkowo wraz ze wzrostem α błąd rośnie. Tu teoria nie gwarantowała nam nic poza tym, że granica przy $\alpha \to \infty$ ma być aproksymacją u_j na danym poziomie rozdzielczości, która spełnia odpowiednie szacowanie błędu – nie oczekujemy więc żadnego szczególnego zachowania błędu dla małych α. Okazuje się, że obserwowane zachowanie jest specyficzne dla wybranego przykładu.

Na rysunkach 4.1 i 4.2, w formie wykresów słupkowych pokazano czas działania programu *Error*. Pomiaru dokonano na komputerze z procesorem Intel Core 2 Duo T7400 o taktowaniu zegara 2.16GHz. Komputer wyposażony był w 3GB pamięci operacyjnej i



Rysunek 4.2. Czas (w sekundach) wykonania polecenia „*Error* 4 α_1" dla $\alpha_1 = 1, \ldots, 5$.

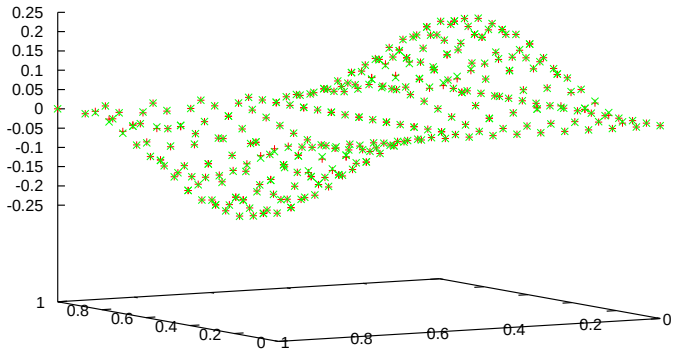
działał pod kontrolą systemu Windows Vista Business w wersji 32-bitowej[1]. Warto zauważyć, że wraz z wzrostem α_1 czas działania rośnie nieznacznie. Jest to efekt zastosowania mechanizmów magazynowania symboli g, a, f opisany w podrozdziale 3.2.3. Dodatkowy wzrost wydajności (i dokładności) można osiągnąć stosując bardziej zaawansowany algorytm rozwiązywania układów równań liniowych. W prezentowanym rozwiązaniu zastosowano bardzo prosty (i niestety niewydajny) mechanizm. W dalszych pracach planowane jest zastosowanie zaawansowanych i zoptymalizowanych bibliotek takich jak np. Intel Math Kernel Library[2] lub rozwiązań opartych na GPU (np. biblioteka CUDA BLAS[3] firmy nVidia).

Drugi program, o nazwie *Plots*, generuje dane pozwalające narysować wykres funkcji współrzędnych u_{j1}, u_{j2}, współrzędnych podanego rozwiązania u_1, u_2 oraz różnicy punktowej $r_1 := u_{j1} - u_1, r_2 := u_{j2} - u_2$. Podobnie jak program liczący błąd, pobiera on dwa argumenty: liczbę j oraz α_1. Generowane są wykresy dla $alpha = 0, 1, \ldots, \alpha_1 - 1$. Program generuje wyniki w postaci tekstu (stanowiącego tabelę danych w której kolejne wiersze oddzielone są znakiem tabulatora). Wyniki zapisywane są w plikach tekstowych – konkretny

[1] Z uwagi na wykorzystanie biblioteki GMP, użycie 64-bitowej wersji systemu i 64-bitowej wersji biblioteki mogłoby przyczynić się do znacznego wzrostu wydajności. Badania pokazują bowiem znaczny wzrost wydajności samej biblioteki GMP w przypadku korzystania z instrukcji 64-bitowych.

[2] Więcej informacji o modułach BLAS i LAPACK z biblioteki Intel Math Kernel Library można znaleźć na stronie `http://www.intel.com/cd/software/products/asmo-na/eng/266858.htm`.

[3] Więcej informacji o języku CUDA można znaleźć na stronie `http://www.nvidia.com/object/cuda_learn.html`.

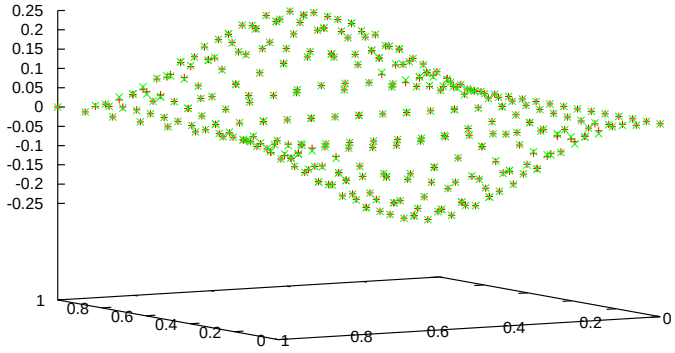


Rysunek 4.3. Wykres pierwszej współrzędnej rozwiązania dokładnego i aproksymacji. Kolorem czerwonym zaznaczono wartości aproksymacji, zielonym rozwiązania dokładnego.

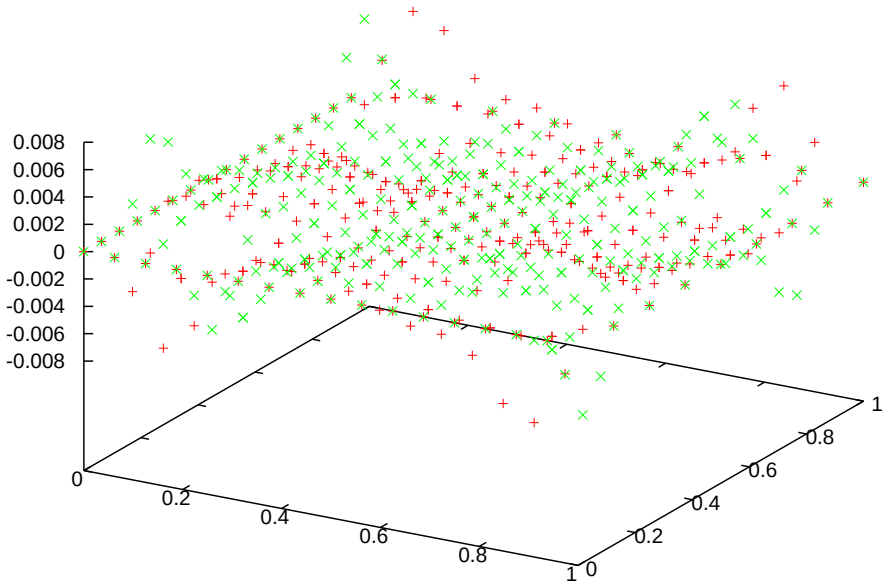
plik odpowiada konkretnemu parametrowi α (w czasie działania program wyświetla informacje gdzie zapisał wyniki). W tabli wynikowej jest osiem kolumn – kolejno: x, y, $u_{j1}(x, y)$, $u_{j2}(x, y)$, $u_1(x, y)$, $u_2(x, y)$, $r_1(x, y)$, $r_2(x, y)$. Poniżej prezentujemy wyniki uzyskane za pomocą tych danych w programie gnuplot[4].

Rysunki 4.3, 4.4 i 4.5 przedstawiają wykresy uzyskane z danych wygenerowanych przez program *Plots* dla $j = 3, \alpha = 1$. Rysunku 4.6 oraz 4.7 przedstawiają obraz pola wektorowego dla $j = 3, \alpha = 1$ oraz $j = 4, \alpha = 1$.

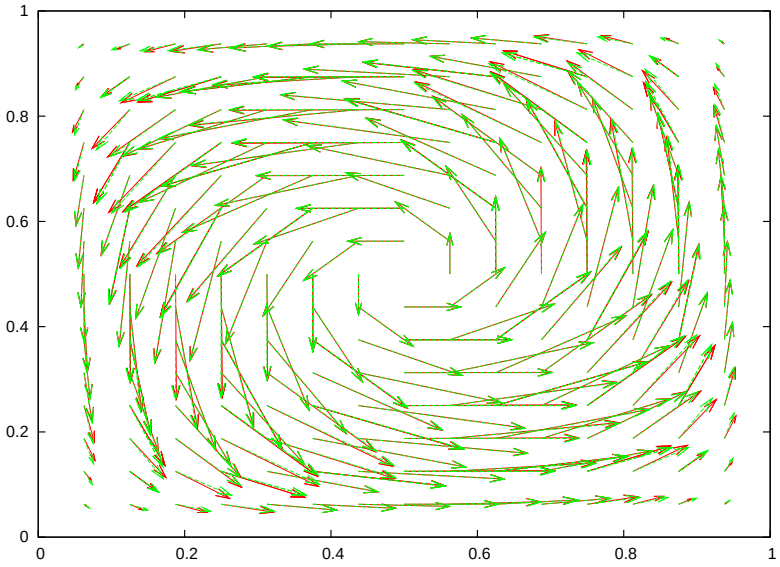
[4]Program ten jest dostępny wraz z kodem źródłowym na stronie `http://www.gnuplot.info`.



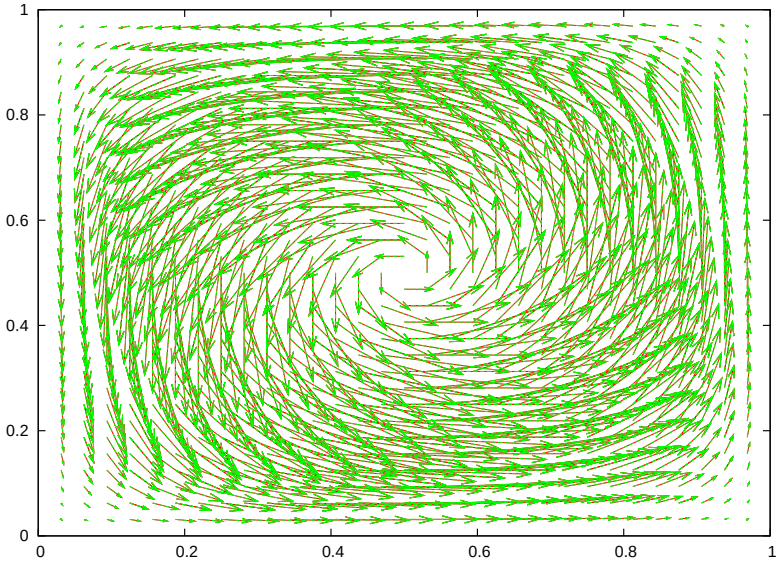
Rysunek 4.4. Wykres drugiej współrzędnej rozwiązania dokładnego i aproksymacji. Kolorem czerwonym zaznaczono wartości aproksymacji, zielonym rozwiązania dokładnego.



Rysunek 4.5. Wykres przedstawiający różnice punktowe pomiędzy aproksymacją a rozwiązaniem dokładnym odpowiednio pierwszej (kolor czerwony) i drugiej (kolor zielony) współrzędnej.



Rysunek 4.6. Wykres przedstawiający pole wektorowe $u(x, y)$. Kolorem czerwonym zaznaczono aproksymację ($j = 3, \alpha = 1$), a kolorem zielonym rozwiązanie dokładne.



Rysunek 4.7. Wykres przedstawiający pole wektorowe $u(x, y)$. Kolorem czerwonym zaznaczono aproksymację ($j = 4, \alpha = 1$), a kolorem zielonym rozwiązanie dokładne.

DODATEK A

Preliminaria

W tym dodatku zebrano kilka najważniejszy, używanych w pracy, pojęć z zakresu algebry liniowej, analizy funkcjonalnej i analizy matematycznej. Informacje w tej części zostały opracowane w oparciu o [7],[8],[15].

A.1. Przestrzenie Hilberta

A.1.1. Podstawowe definicje

Przestrzenie Hilberta są bardzo istotne z punktu widzenia analizy funkcjonalnej, w związku z tym zaczniemy od podania podstawowych definicji i faktów z nimi związanych.

Definicja A.1 (ciąg Cauchy'ego)**.** Niech $(x_n)_{n\in\mathbb{N}}$ będzie ciągiem elementów unormowanej przestrzeni liniowej X. Ciąg $(x_n)_{n\in\mathbb{N}}$ nazywamy ciągiem Cauchy'ego wtedy i tylko wtedy, gdy:

$$\forall_{\varepsilon>0}\exists_{N>0}\forall_{n,m>N} \|x_m - x_n\| < \varepsilon.$$

Fakt A.2. *Każdy ciąg zbieżny jest ciągiem Cauchy'ego.*

Definicja A.3 (przestrzeń Banacha)**.** Niech X przestrzeń liniowa unormowana. Mówimy, że X jest przestrzenią Banacha wtedy i tylko wtedy, gdy każdy ciąg Cauchy'ego zawarty w X jest zbieżny.

Definicja A.4 (zbiór zwarty)**.** Niech X przestrzeń liniowa unormowana. Zbiór $U \subset X$ nazywamy zwartym wtedy i tylko wtedy, gdy z każdego ciągu zawartego w U można wybrać podciąg zbieżny w U.

Definicja A.5 (iloczyn skalarny, przestrzeń unitarna)**.** Niech X będzie przestrzenią liniową. Funkcję $\langle\cdot,\cdot\rangle\colon X \times X \to \mathbb{R}$ nazywamy iloczynem skalarnym wtedy i tylko wtedy, gdy:

a) $\langle x, y\rangle = \langle y, x\rangle$,

b) $\langle cx, y\rangle = c\langle x, y\rangle$,

c) $\langle x + y, z\rangle = \langle x, z\rangle + \langle y, z\rangle$,

d) $\langle x, x\rangle = 0$ wtedy i tylko wtedy, gdy $x = 0$,

e) $\langle x, x\rangle > 0$ dla $x \neq 0$.

Przestrzeń X z określonym iloczynem skalarnym nazywamy przestrzenią unitarną.

Fakt A.6. *Niech X przestrzeń unitarna z iloczynem skalarnym $\langle\cdot,\cdot\rangle$. Wówczas $\|x\| = \sqrt{\langle x, x\rangle}$ jest poprawnie określoną normą w przestrzeni X.*

Definicja A.7 (elementy ortogonalne)**.** Dwa elementy x, y przestrzeni unitarnej X nazywamy ortogonalnymi wtedy i tylko wtedy, gdy $\langle x, y\rangle = 0$.

Definicja A.8 (baza ortonormalna)**.** Niech $(e_i)_{i\in I}$ będzie bazą przestrzeni unitarnej X. Mówimy, że (e_i) jest bazą ortonormalną wtedy i tylko wtedy, gdy:

$$\langle e_i, e_j\rangle = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}.$$

Definicja A.9 (baza Riesza)**.** Niech $\{e_i\}_{i\in I}$ będzie bazą przestrzeni unitarnej X. Mówimy, że jest to baza Riesza, gdy istnieją stałe $0 < c \leqslant C$ takie, że dla wszystkich ciągów liczbowych $\{a_i\}_{i\in I}$:

$$c\left(\sum_{i\in I}|a_i|^2\right)^{1/2} \leqslant \left\|\sum_{i\in I} a_i x_i\right\| \leqslant C\left(\sum_{i\in I}|a_i|^2\right)^{1/2}.$$

Definicja A.10 (przestrzeń Hilberta)**.** Niech X będzie przestrzenią unitarną. Jeśli X jest równocześnie przestrzenią Banacha (z normą określoną w poprzednim fakcie), to X nazywamy przestrzenią Hilberta.

A.1.2. Przykłady

Poniżej zebrano najważniejsze, z punktu widzenia niniejszej rozprawy, przykłady przestrzeni Hilberta.

Definicja A.11 (przestrzeń Lebesgue'a)**.** Niech $d \in \mathbb{N}_+$ oraz $A \subset \mathbb{R}^d$ dowolny. Przez $L_2(A)$ oznaczamy przestrzeń wszystkich (w sensie klas abstrakcji relacji równości) rzeczywistych funkcji mierzalnych takich, że:

$$\left(\int_A |f|^2\, d\mu\right)^{1/2} =: \|f\| \leqslant \infty,$$

i nazywamy przestrzenią Lebesgue'a.

Fakt A.12. *Przestrzeń Lebesgue'a z iloczynem skalarnym:*

$$\langle f, g\rangle := \int_A f \cdot g \; d\mu,$$

jest przestrzenią Hilberta.

Definicja A.13 (przestrzeń Sobolewa)**.** Niech A podobnie jak poprzednio dowolny podzbiór $\mathbb{R}^d$ dla pewnego $d \in \mathbb{N}_+$. Niech przestrzeń $H^1(A)$ będzie przestrzenią funkcji należących do $L_2(A)$ takich, że ich pochodna pierwszego rzędu również należy do $L_2(A)$. Przestrzeń $H^1(A)$ z normą:

$$\|f\|_{H^1(A)} := \|f\|_{L_2(A)} + \|f'\|_{L_2(A)},$$

gdzie norma $\|\cdot\|_{L_2(A)}$ jest normą przestrzeni $L_2(A)$, nazywamy przestrzenią Sobolewa.

Fakt A.14. *Przestrzeń Sobolewa'a z iloczynem skalarnym:*

$$\langle f, g\rangle := \int_A f \cdot g \; d\mu + \int_A f'g' d\mu,$$

jest przestrzenią Hilberta.

Definicja A.15. Niech $f\colon A \to \mathbb{R}$. Zbiór:

$$\operatorname{supp} f = \overline{\{x \in A : f(x) \neq 0\}}$$

nazywamy nośnikiem funkcji f.

A.2. Operatory i funkcjonały liniowe

A.2.1. Transformata Fouriera

Definicja A.16 (transformata Fouriera)**.** Niech $f \in L_1(\mathbb{R})$. Transformatą Fouriera funkcji f nazywamy funkcję:

$$\hat{f}(\xi) = \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} e^{-i\langle \xi, x\rangle} f(x) dx.$$

Przez $\mathcal{F}$ będziemy oznaczać odwzorowanie $\mathcal{F}\colon L_1(\mathbb{R}) \to L_1(\mathbb{R})$ dane wzorem: $\mathcal{F}(f) = \hat{f}$. Oczywiście tak określone $\mathcal{F}$ jest odwzorowaniem liniowym. Można pokazać, że $\mathcal{F}$ jest izometrią przestrzeni $L_2(\mathbb{R})$, przy czym transformata odwrotna dana jest wzorem $(\mathcal{F}^{-1}f)(x) = \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} f(\xi) e^{i\langle x, \xi\rangle} dx$.

Istnieje ponadto ścisły związek między transformatą Fouriera a iloczynem spolotowym, zdefiniowanym w rozdziale pierwszym (definicja 1.10). Jeśli bowiem oznaczymy $\mathcal{F}_1 := \sqrt{2\pi}\mathcal{F}$ to:

$$\mathcal{F}_1(f * g) = \mathcal{F}_1(f)\mathcal{F}_1(g).$$

A.2.2. Operatory różniczkowe

Poniżej znajduje się definicja operatorów występujących w równaniach Naviera–Stokesa.

Załóżmy, że $U \subset \mathbb{R}^2$ jest otwarty, $(x_0, y_0) \in U$, $f\colon U \to \mathbb{R}$ jest różniczkowalna w otoczeniu (x_0, y_0).

Definicja A.17 (gradient)**.** Gradientem funkcji f w punkcie (x_0, y_0) nazywamy:

$$\nabla f = \left[\frac{\partial f}{\partial x}(x_0, y_0), \frac{\partial f}{\partial y}(x_0, y_0)\right].$$

Uwaga A.18. Jeśli $f\colon U \subset \mathbb{R}^n \to \mathbb{R}^n$ to gradientem jest macierz kwadratowa zawierająca wszystkie pochodne cząstkowe:

$$\nabla f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ & \vdots & \\ \frac{\partial f_n}{\partial x_1} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

Definicja A.19 (laplasjan)**.** Laplasjanem funkcji f nazywamy w (x_0, y_0):

$$\Delta f(x_0, y_0) = \frac{\partial^2 f}{\partial x^2}(x_0, y_0) + \frac{\partial^2 f}{\partial y^2}(x_0, y_0).$$

Definicja A.20 (dywergencja)**.** Dywergencją funkcji $f\colon U \subset \mathbb{R}^2 \to \mathbb{R}^2$ w punkcie $(x_0, y_0) \in U$ nazywamy:

$$\operatorname{div} f(x_0, y_0) = \frac{\partial f_1}{\partial x}(x_0, y_0) + \frac{\partial f_2}{\partial y}(x_0, y_0).$$

Uwaga A.21. Bazując na powyższych definicjach lokalnych (w punkcie (x_0, y_0)) można łatwo sformułować globalne definicje operatorów gradientu, laplasjanu i dywergencji, które danej funkcji f przypisują nową funkcję odpowiednio ∇f, Δf i $\operatorname{div} f$, która na pewnym zbiorze otwartym przyjmuje wartości zgodne z powyższymi definicjami.

Bibliografia

[1] P. Vial A. Cohen, I. Daubechies. *Wavelets on the interval and fast wavelet transforms*. *Applied and Computational Harmonic Analysis*, pages 54–81, 1993.

[2] J. T. Białasiewicz. *Falki i aproksymacje*. Wydawnictwa Naukowo-Techniczne, Warszawa, 2004.

[3] K. Bittner. *A new view on biorthogonal spline wavelets*. Preprint, Universitat Ulm, 2005.

[4] I. Daubechies. *Orthonormal bases of compactly supported wavelets*. *Communications On Pure and Applied Mathematics*, 41:909–996, 1988.

[5] A. Barinka et al. *Some Remarks on Quadrature Formulas for Refinable Functions and Wavelets*. *ZAMM - Journal of Applied Mathematics and Mechanics*, 81:839–855, 2001.

[6] J. Liandrat G. Chiavassa. *On the effective construction of compactly supported wavelets satisfying homogeneous boundary conditions on the interval*. *Applied and Computational Harmonic Analysis*, pages 62–73, 1997.

[7] S.G. Krejna. *Analiza funkcjonalna*. PWN, Warszawa, 1967.

[8] J. Musielak. *Wstęp do analizy funkcjonalnej*. PWN, Warszawa, 1989.

[9] A. Zielonka R. Grzymkowski. *Zastosowania teorii falek w zagadnieniach brzegowych*. Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice, 2004.

[10] R. Temam. *Navier-Stokes Equations: Theory and Numerical Analysis*. American Mathematical Society, New York, 2001.

[11] K. Urban. *Using divergence free wavelets for the numerical solution of the Stokes problem*. In *AMLI'96: Proceedings of the Conference on Algebraic Multilevel Iteration Methods with Applications*, volume 2, pages 261–277, Nijmegan, The Netherlands, 1996. University of Nijmegan.

[12] P.A. Raviart V. Girault. *Finite element methods for Navier-Stokes equations: Theory and algorithms*. Springer-Verlag, Berlin, 1981.

[13] K. Urban W. Dahmen, A. Kunoth. *A Wavelet Galerkin Method for the Stokes Equations*. *Computing*, 56(3):259–301, 1996.

[14] Y. Xu W. Dahmen, R. Schneider. *Nonlinear functionals of wavelet expansions –– adaptive reconstruction and fast evaluation*. *Numerische Mathematik*, 86(1):49–101, 2000.

[15] P. Wojtaszczyk. *Teoria Falek*. PWN, Warszawa, 2000.

[16] Y. He X. Zhou. *Using divergence free wavelets for the numerical solution of the 2–D stationary Navier–Stokes equations*. *Applied Mathematics and Computation*, 163:593–607, 2005.

Oświadczenie

Ja, niżej podpisany(a) oświadczam, iż przedłożona praca dyplomowa została wykonana przeze mnie samodzielnie, nie narusza praw autorskich, interesów prawnych i materialnych innych osób.

.....	.....
data	podpis