AGH
Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Fizyki i Informatyki Stosowanej

Praca inżynierska

Dariusz Rębisz

kierunek studiów: **informatyka stosowana**

Biblioteka numeryczna do rozwiązywania algebraicznych układów równań liniowych metodami iteracyjnymi

Opiekun: dr hab. inż. Tomasz Chwiej

Kraków, styczeń 2020

Wstęp

Rewolucja naukowo-techniczna, rozumiana jako proces wielkich przemian w technice, produkcji oraz nauce zapoczątkowana została w latach 50. XX wieku i trwa do dnia dzisiejszego. Jednymi z najważniejszych jej elementów są komputeryzacja i automatyzacja procesów pracy. Komputery stały się nieodzownymi narzędziami wykorzystywanymi niemal we wszystkich dziedzinach nauki. Równolegle z dynamicznie postępującym wzrostem ich wydajności, opracowywane były coraz to nowsze metody pozwalające na rozwiązywanie problemów obliczeniowych stawianych przez naukę. Metody te nazywane są metodami numerycznymi.

Metodą numeryczną jest każda metoda obliczeniowa sprowadzalna do operacji dodawania, odejmowania, mnożenia i dzielenia. Dopóki były one nieskomplikowane, dawały się rozwiązywać analitycznie, z użyciem pewnych przekształceń prowadzących do otrzymania ścisłych rozwiązań. Z biegiem czasu powstawały jednak coraz to bardziej spomplikowane teorie opisujące różnego rodzaju zjawiska, powodując znaczący wzrost złożoności problemów, tak, że ich ścisłe rozwiązanie było zbyt czasochłonne lub wręcz niemożliwe. Stworzone zostały więc algorytmy numeryczne, pozwalające na znajdowanie przybliżonych rozwiązań z zadaną dokładnością.

Rozwiązywanie układów równań liniowych to jeden z najczęściej spotykanych problemów algebraicznych w zagadnieniach fizyki[1]. Pojawia się on między innymi w takich zagadnieniach jak diagonalizacja metodą iteracji odwrotnej, charakterystyka układu wielu cząstek opisanego funkcjami Greena czy konstrukcja bazy funkcji kształtu wyższych rzędów w metodzie elementów skończonych.

Układ równań liniowych można przedstawić w postaci równania macierzowego

$$A \cdot \boldsymbol{x} = \boldsymbol{b} \Rightarrow \boldsymbol{x} = A^{-1}\boldsymbol{b} \tag{1}$$

Niezbędne więc było opracowanie aparatu analizy również w formie wektorowej i macierzowej. Podejście analityczne sprowadzające się do wyznaczenia macierzy odwrotnej okazywało się często zbyt czasochłonne, a rozwiązania przybliżone na ogół spełniały kryteria dokładności stąd opracowany został szereg algorytmów zmniejszających ilośc niezbędnych zasobów (sprzętowych, czasowych) koniecznych do uzyskania wyniku.

Macierz A z równania (1) w szczególnym przypadku może w większości składać się z elementów zerowych. Taką macierz nazywamy rzadką. Najczęściej odpowiada ona układom o bardzo dużej liczbie stopni swobody, z których każdy wiąże się bezpośrednio z niewielką ilością innych stopni swobody. Macierze rzadkie znajdują szczególne zastosowanie w teorii grafów oraz dyscyplinach pochodnych, np. teorii sieci. Dzięki zastosowaniu specjalnych struktur danych,

przechowywanie i operacje na macierzach rzadkich wiążą się z dużo mniejszym zużyciem pamięci operacyjnej niż w przypadku macierzy gęstych.

W poniższej pracy przedstawiona została procedura implementacji pięciu algorytmów numerycznych służących do rozwiązywania układów równań liniowych: sprzężonego gradientu (CG), kwadratowego sprzężonego gradientu (CGS), stabilizowanego gradientu bisprzężonego (BICGSTAB), kwazi-najmniejszego residuum bez transpozycji macierzy (TFQMR) oraz uogólnionej metody najmniejszego residuum (GMRES) dla macierzy rzadkich. W rozdziale pierwszym, stanowiącym wstęp teoretyczny, opisano podstawy każdej z tych metod oraz stojący za nimi aparat matematyczny. Wymienione zostały również różnice pomiędzy algorytmami a także zakresy ich stosowalności. W rozdziale drugim przedstawiona zostanie krótka charakterystyka programu oraz zaperezentowane zostaną, za pomocą pseudokodu, zaimplementowane algorytmy. Rozdział poświęcony jest również testom aplikacyjnym, mianowicie algorytmy implemetujące wymienione wyżej metody są badane pod kątem rozwiązywalności układu równań i szybkości zbieżności. Testowanie wykonano na zestawie macierzy rzadkich znajdujących się w kolekcjach Matrix Market, które najczęściej występują w benchmarkach dedykowanych metodom iteracyjnym dla macierzy rzadkich. Dzięki takiemu podejściu, takie zestandaryzowane benchmarki pozwalają obiektywnie określić wydajność poszczególnych algorytmów i metod. Dołączony został również dodatek zawierający deklaracje zaimplementowanych procedur oraz wyjaśniający znaczenie zwracanych komunikatów zwrotnych.

1. WYBRANE METODY ITERACYJNEGO ROZWIĄZYWANIA UKŁADÓW RÓWNAŃ LINIOWYCH

Pierwszymi metodami iteracyjnymi stosowanymi do rozwiązywania dużych układów równań liniowych były metody stacjonarne[2], dla których kolejne przybliżenia wektora rozwiązań można wyrazić w postaci

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + M^{-1}\boldsymbol{r}_k \tag{2}$$

Do metod stacjonarnych zaliczane są m.in. metoda Jacobiego czy Gaussa-Seidela. Mechanizm ich działania rozpoczyna się od przyjęcia pewnego startowego wektora rozwiązań, a następnie modyfikacji jego elementów w ustalonej kolejności aż do osiągnięcia zbieżności. Modyfikacje te ukierunkowane są na wyeliminowaniu kolejnych komponentów wektora reszt. Współcześnie metody stacjonarne są rzadko stosowane ze względu na ich wolną zbieżność. Pomimo prób poprawy ich szybkości poprzez wprowadzenie nadrelaksacji zostały one w znacznej mierze wyparte przez metody oparte na wykorzystaniu podprzestrzeni Kryłowa, opisane w kolejnych punktach tego rozdziału.

1.1 Metoda sprzężonego gradientu

Algorytm sprzężonego gradientu (*Conjugate Gradient)* jest jedną z najbardziej znanych technik iteracyjnych służących do rozwiązywania algebraicznych układów równań z symetryczną, dodatniookreśloną macierzą rzadką. Został on opracowany przez Magnusa Hestenesa oraz Eduarda Stiefela[3]. Posługując się formalizmem matematycznym metodę sprzężonego gradientu można przedstawić jako projekcję ortogonalną na podprzestrzeń Kryłowa K_m ($\mathbf{r_0}$, A).

Zakładając, że $\{\mathbf{p_k}\}$ jest ciągiem n liniowo niezależnych wektorów w R^n, opisują one kierunki wzajemnie sprzężone. W bazie tej rozwiązanie równania (1) możemy przedstawić w postaci kombinacyjnej:

$$\boldsymbol{x} = \sum_{i=1}^{n} \alpha_i \boldsymbol{p}_i \tag{3}$$

Algorytm CG wyprowadzony został jako sposób poszukiwania wartości minimalnej formy kwadratowej:

$$\phi(x) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{b} \tag{4}$$

gdzie rozwiązanie **x** minimalizuje tę formę. Wartość fromy (4) dla $\mathbf{x} \neq \mathbf{x_{dok}}$ jest dodatnia ze względu na dodatniookreśloność macierzy A. Jako pierwszy wektor bazowy $\mathbf{p_1}$ należy wybrać gradient φ

dla wektora startowego $\mathbf{x_0}$, który wynosi $A\mathbf{x_0} - \mathbf{b}$. Pozostałe wektory w bazie będą sprzężone do gradientu.

Przyjmując $\mathbf{r_k}$ jako residuum w k-tym kroku:

$$\boldsymbol{r}_k = \boldsymbol{b} - A\,\boldsymbol{x}_k \tag{5}$$

dążymy do tego, aby $<\mathbf{x_k} \mid \mathbf{r_k}> = 0$ (=) $\varphi = 0$. Ze względu na założenie wzajemnej sprzężoności kierunków $\mathbf{p_k}$ wybierany jest kierunek najbliższy do $\mathbf{r_k}$ pod warunkiem sprzężoności. Jako kolejny wektor bazowy $\mathbf{p_{k+1}}$ przyjmujemy $\mathbf{r_k}$, z którego usuwamy wkład (informacje) pochodzący od $\mathbf{p_k}$

$$\boldsymbol{p}_{k+1} = \boldsymbol{r}_k - \frac{\boldsymbol{p}_k^T A\,\boldsymbol{r}_k}{\boldsymbol{p}_k^T A\,\boldsymbol{p}_k}\,\boldsymbol{p}_k \tag{6}$$

gdzie $\mathbf{p^T_k}A\mathbf{r_k}$ jest iloczynem macierzowym a $\mathbf{p^T_k}A\mathbf{p_k}$ określa normę wektora $\mathbf{p_k}$. Dzięki tej modyfikacji wektory $\mathbf{p_{k+1}}$ i $\mathbf{p_k}$ stają się A-sprzężone.

Poniżej zamieszczono algorytm sprzężonego gradientu w notacji pseudokodu.

Algorytm 1. Sprzężony gradient

1. *Oblicz* $\boldsymbol{r}_0 := \boldsymbol{b} - A\,\boldsymbol{x}_0$, $\boldsymbol{p}_0 := \boldsymbol{r}_0$
2. *For* $j = 0,1,\ldots,$ *Do*:
3. $a_j := (\boldsymbol{r}_j, \boldsymbol{r}_j)/(A\,\boldsymbol{p}_j, \boldsymbol{p}_j)$
4. $\boldsymbol{x}_{j+1} := \boldsymbol{x}_j + \alpha_j\,\boldsymbol{p}_j$
5. $\boldsymbol{r}_{j+1} := \boldsymbol{r}_j - \alpha_j A\,\boldsymbol{p}_j$
6. $\beta_j := (\boldsymbol{r}_{j+1}, \boldsymbol{r}_{j+1})/(\boldsymbol{r}_j, \boldsymbol{r}_j)$
7. $\boldsymbol{p}_{j+1} := \boldsymbol{r}_{j+1} + \beta_j\,\boldsymbol{p}_j$
8. *EndDo*

W algorytmie tym występuje pojedyńcze mnożenie wektor-macierz, a w pamięci przechowywane są cztery wektory (**x**, **p**, A**p** oraz **r**). Za $\mathbf{x_0}$ przyjęty został wektor zerowy. Zakończenie algorytmu następuje, gdy norma z wektora reszt $\|\mathbf{r_{j+1}}\|$ jest mniejsza od arbitralnie przyjętej wartości ε.

1.2 Metody gradientów bisprzężonych oraz kwadratowego gradientu sprzężonego

Na podstawie algorytmu sprzężonego gradientu opracowany został szereg metod pochodnych, w których zastosowane modyfikacje pozwalają na szybsze osiągnięcie zbieżności lub znoszą warunki narzucone na macierz A w równaniu (1). Pierwszą z tych metod jest opracowana w 1952 r. przez Corneliusa Lanczosa metoda gradientów bisprzężonych (*Biconjugate Gradient*)[4]. Pozwala ona na uzyskanie rozwiązania nie tylko oryginalnego równania (1) ale również równania dualnego z macierzą transponowaną:

$$A^T \boldsymbol{x}^* = \boldsymbol{b}^* \tag{7}$$

Wadą tego algorytmu jest konieczność wykonywania kosztownych operacji mnożenia wektora przez macierz A oraz A^T w każdej iteracji. Podjęte zostały więc próby wyeliminowania odwołania do macierzy transponowanej z algorytmu, zmniejszając w ten sposób wymagany nakład obliczeniowy. Doprowadziło to do opracowania w 1984 r. przez Pietera Sonnevelda metody kwadratowego gradientu sprzężonego (C*onjugate Gradient Squared)*[5]. CGS pozwala na szybsze osiągnięcie zbieżności niż BCG kosztem podobnego nakładu obliczeniowego.

W algorytmie BCG wektor reszt w j-tej iteracji spełnia zależność:

$$\boldsymbol{r}_j = \phi_j(A)\boldsymbol{r}_0 \tag{8}$$

Gdzie φ_j jest pewnym wielomianem macierzowym stopnia j spełniającym warunek $\varphi_j(0) = 1$.

Podobnie, kierunek kolejnych gradientów zadany jest przez:

$$\boldsymbol{p}_j = \pi_j(A)\boldsymbol{r}_0 \tag{9}$$

Gdzie π_j również jest wielomianem stopnia j. W celu wyliczenia współczynników skalarnych potrzebnych do wyznaczenia kolejnych przybliżeń wektora **x** konieczne jest zadziałanie powyższymi wielomianami nie tylko na pierwotną macierz A, ale również na A^T, co bardzo często komplikuje i spowalnia proces iteracji. Dokonując przekształceń matematycznych uzyskujemy wzór na współczynniki skalarne, gdzie zamiast wielomianów φ_j i π_j zastosowane są ich kwadraty. Tak zrodził się pomysł algorytmu kwadratowego sprzężonego gradientu, gdzie kolejne wektory reszt spełniają zależność:

$$\boldsymbol{r}_j = \phi_j^2(A)\boldsymbol{r}_0 \tag{10}$$

Jak wspomniano wyżej, przekształcenia prowadzące do metody CGS są czysto matematyczne. Przewagą metody CGS w stosunku do BCG jest pozbycie się mnożenia wektora przez A^T. Algorytm CGS sprawdza się w wielu przypadkach, jednakże ze względu na kwadratową postać wielomianów błędy zaokrągleń stają się większe w porównaniu do BCG.

Poniżej zamieszczono algorytm kwadratowego gradientu sprzężonego w notacji pseudokodu.

Algorytm 2. Kwadratowy sprzężony gradient

1. *Oblicz* $\boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0$, $\boldsymbol{r}_0^*$ *dowolne*
2. *Oblicz* $\boldsymbol{p}_0 := \boldsymbol{u}_0 := \boldsymbol{r}_0$
3. *For* $j = 0,1,2,\ldots$, *Do*:
4. $\alpha_j = (\boldsymbol{r}_j, \boldsymbol{r}_0^*) / (A\boldsymbol{p}_j, \boldsymbol{r}_0^*)$
5. $\boldsymbol{q}_j = \boldsymbol{u}_j - \alpha_J A\boldsymbol{p}_j$
6. $\boldsymbol{x}_{j+1} = \boldsymbol{x}_j + \alpha_j(\boldsymbol{u}_j + \boldsymbol{q}_j)$
7. $\boldsymbol{r}_{j+1} = \boldsymbol{r}_j - \alpha_j A(\boldsymbol{u}_j + \boldsymbol{q}_j)$
8. $\beta_j = (\boldsymbol{r}_{j+1}, \boldsymbol{r}_0^*) / (\boldsymbol{r}_j, \boldsymbol{r}_0^*)$
9. $\boldsymbol{u}_{j+1} = \boldsymbol{r}_{j+1} + \beta_j \boldsymbol{q}_j$
10. $\boldsymbol{p}_{j+1} = \boldsymbol{u}_{j+1} + \beta_j(\boldsymbol{q}_j + \beta_j \boldsymbol{p}_j)$
11. *EndDo*

W każdej iteracji wykonywane są dwa mnożenia wektor-macierz ($A\mathbf{p_j}$ i $A(\mathbf{u_j} + \mathbf{q_j})$) oraz trzy iloczyny skalarne a w pamięci przechowywanych jest sześć wektorów. Za $\mathbf{x_0}$ przyjęty został wektor zerowy a $\mathbf{r^*_0}$ generowany jest w sposób losowy. Ze względu na brak konieczności mnożenia wektorów przez macierz A^T spodziewana jest dwukrotnie szybsza zbieżność algorytmu w stosunku do BCG.

1.3 BICGSTAB

Algorytm CGS polega na zastosowaniu kwadratów wielomianów resztowych i w przypadku nieregularnej zbieżności, może prowadzić do stopniowego narastania błędów zaokrągleń lub nawet do przepełnienia. Algorytm stabilizowanego gradientu bisprzężonego jest wariantem CGS opracowanym w celu ograniczenia skutków tych problemów. Zamiast poszukiwania wektorów reszt określonych równaniem (9) BICGSTAB określa je zależnością:

$$\boldsymbol{r}_j = \psi_j(A)\,\phi_j(A)\,\boldsymbol{r}_0 \tag{11}$$

gdzie $\varphi_j(t)$ jest wielomianem resztowym stosowanym w algorytmie BCG a $\psi_j(t)$ jest nowym wielomianem zastosowanym w celu ustabilizowania zbieżności oryginalnego algorytmu. W szczególności, $\psi_j(t)$ może być zadane poprzez prostą rekurencje:

$$\psi_{j+1}(t) = (1 - \omega_j t)\,\psi_j(t) \tag{12}$$

w której należy wyznaczyć skalarną wielkość ω_j nazywaną parametrem stabilizującym. W podobny sposób obliczane są wektory bazy:

$$\boldsymbol{p}_j = \psi_j(A)\,\pi_j(A)\,\boldsymbol{r}_0 \tag{13}$$

gdzie wielomian $\pi_j(t)$ również zaczerpnięty został z algorytmu BCG. Oba wielomiany, $\varphi_j(t)$ oraz $\pi_j(t)$ spełniają zależność rekurencyjną:

$$\phi_{j+1}(t)=\phi_j(t)-\alpha_j t\,\pi_j(t)$$
$$\pi_{j+1}(t)=\phi_{j+1}(t)+\beta_j\pi_j(t) \qquad (14)$$

Znając wartości tych wielomianów w kolejnych iteracjach jesteśmy w stanie wyznaczyć wektory $\mathbf{p_{k+1}}$ oraz $\mathbf{r_{k+1}}$

$$\boldsymbol{r_{j+1}}=(I-\omega_j A)(\boldsymbol{r_j}-\alpha_j A\,\boldsymbol{p_j})$$
$$\boldsymbol{p_{j+1}}=\boldsymbol{r_{j+1}}+\beta_j(I-\omega_j A)\,\boldsymbol{p_j} \qquad (15)$$

Parametry α oraz β, obecne w powyższych wzorach, nie mogą zostać obliczone w sposób podobny do BCG, ponieważ żaden z wektorów $\varphi_j(A)r_0$, $\varphi_j(A^T)r^*$ czy $\varphi_j(A)^2r_0$ nie jest dostępny. Z tego powodu wprowadzony został parametr $\tilde{p}_j$ = ($\mathbf{r_j}$, $\mathbf{r^*_0}$) za pomocą którego wyrazić można oba parametry α oraz β

$$\beta_j=\frac{\tilde{p}_{j+1}}{\tilde{p}_j}\cdot\frac{\alpha_j}{\omega_j}$$
$$\alpha_j=\frac{\tilde{p}_j}{(A\,\boldsymbol{p_j},\boldsymbol{r_0^*})} \qquad (16)$$

Algorytm BICGSTAB jest w wielu przypadkach znacznie mniej kosztowny niż BCG. Poniżej zamieszczono algorytm BICGSTAB w notacji pseudokodu.

Algorytm 3. BICGSTAB

1. *Oblicz* $\boldsymbol{r_0}=\boldsymbol{b}-A\,\boldsymbol{x_0}$, $\boldsymbol{r_0}^*$ *dowolne*
2. *Oblicz* $\boldsymbol{p_0}=\boldsymbol{r_0}$
3. *For* $j=0,1,2,\ldots$, *Do*:
4. $a_j=(\boldsymbol{r_j},\boldsymbol{r_0}^*)/(A\,\boldsymbol{p_j},\boldsymbol{r_0}^*)$
5. $\boldsymbol{s_j}=\boldsymbol{r_j}-\alpha_J A\,\boldsymbol{p_j}$
6. $\omega_j=(A\,\boldsymbol{s_j},\boldsymbol{s_j})/(A\,\boldsymbol{s_j},A\,s_j)$
7. $\boldsymbol{x_{j+1}}=\boldsymbol{x_j}+\alpha_j\boldsymbol{p_j}+\omega_j\boldsymbol{s_j}$
8. $\boldsymbol{r_{j+1}}=\boldsymbol{s_j}-\omega_j A\,\boldsymbol{s_j}$
9. $\beta_j=\frac{(\boldsymbol{r_{j+1}},\boldsymbol{r_0}^*)}{(\boldsymbol{r_j},\boldsymbol{r_0}^*)}\cdot\frac{\alpha_j}{\omega_j}$
10. $\boldsymbol{p_{j+1}}=\boldsymbol{r_{j+1}}+\beta_j(\boldsymbol{p_j}-\omega_j A\,\boldsymbol{p_j})$
11. *EndDo*

Podobnie jak w algorytmie CGS, wykonywane są dwie operacje mnożenia wektor-macierz ($A\mathbf{p_j}$, $A\mathbf{s_j}$), jednak liczba iloryczynów skalarnych wynosi pięć. W pamięci przechowywanych jest sześć wektorów.

1.4 QMR i TFQMR

Metoda Lanczosa[6] jest bezpośrednim algorytmem wyznaczania m najbardziej użytecznych (zmierzających do maksymalnych bądź minimalnych) wartości i wektorów własnych n-wymiarowej kwadratowej macierzy hermitowskiej, gdzie m jest często znacznie mniejsze od rozmiaru tej macierzy. Generowana jest ortogonalna baza V_m, będąca kombinacją liniową wektorów przestrzeni Kryłowa, w taki sposób, aby przetransformowana macierz A miała postać trójdiagonalnej macierzy T_m

$$AV_m = V_{m+1} T_m \tag{17}$$

Macierz V_m zbudowana jest z m n-wymiarowych wektorów kolumnowych, macierz V_{m+1} tworzona jest z macierzy V_m poprzez dobudowanie do niej dodatkowej kolumny. Biortogonalizacja Lanczosa jest rozszerzeniem metody Lanczosa znoszącym warunek symetryczności macierzy A. W jego wyniku otrzymywane są dwie biortogonalne bazy dla dwóch podprzestrzeni Kryłowa : V_m oraz W_m.

W wyniku biortogonalizacji Lanczosa otrzymujemy bazę V_m oraz macierz T_m, za pomocą których można wyrazić wektor residualny dla $\mathbf{x} = \mathbf{x_0} + V_m\mathbf{y}$ poprzez:

$$\boldsymbol{r_m} = \boldsymbol{b} - A\,\boldsymbol{x_m} = V_{m+1}(\beta \boldsymbol{e_1} - T_m \boldsymbol{y}) \tag{18}$$

gdzie **y** jest dowolnym wektorem a parametr β spełnia zależność $\mathbf{v_0} = \beta \cdot \mathbf{r_1}$. Minimalizowana jest następnie wartość $\|\beta e_1 - T_m y\|_2$, a rozwiązanie $\mathbf{x} = \mathbf{x_0} + V_m\mathbf{y}$ jest tzw. rozwiązaniem o kwasi-minimalnym residuum na którym opiera się metoda QMR.

Algorytm TFQMR to kolejna z metod powstałych na bazie CGS. Zamiast wyznaczania kolejnych przybliżeń poszukiwanego wektora **x** w jednym kroku w każdej iteracji, robione jest to w dwóch etapach:

$$\begin{aligned} \boldsymbol{x_{j+1/2}} &= \boldsymbol{x_j} + \alpha_j \boldsymbol{u_j} \\ \boldsymbol{x_{j+1}} &= \boldsymbol{x_{j+1/2}} + \alpha_j \boldsymbol{q_j} \end{aligned} \tag{19}$$

Gdzie parametr α_j oraz wektory $\mathbf{u_j}$ i $\mathbf{q_j}$ zdefiniowane są w analogiczny sposób jak w metodzie CGS:

$$\begin{aligned} \alpha_j &= (\boldsymbol{r_j}, \boldsymbol{r_0^*}) / (A\,\boldsymbol{p_j}, \boldsymbol{r_0^*}) \\ \boldsymbol{q_j} &= \boldsymbol{r_j} + \beta_{j-1} \boldsymbol{q_{j-1}} - \alpha_j A\,\boldsymbol{p_j} \\ \boldsymbol{u_j} &= \boldsymbol{r_j} + \beta_{j-1} \boldsymbol{q_{j-1}} \end{aligned} \tag{20}$$

W standardowej metodzie QMR macierz transponowana A^T pojawia się jedynie w równaniu rekurencyjnym dla wektorów $\mathbf{w_j}$ z bazy W_m. Wektory te są konieczne do wyznacznia macierzy T_m jednak nie biorą jawnego udziału w obliczaniu kolejnych przybliżeń wektora **x**. Metoda TFQMR zakłada, że zawsze jest możliwym wyeliminowanie transpozycji macierzy poprzez dobór właściwego wektora startowego $\mathbf{w_1}$. Poniżej zamieszczono algorytm TFQMR w notacji pseudokodu.

Algorytm 4. TFQMR

1. *Oblicz* $\boldsymbol{w_0}=\boldsymbol{u_0}=\boldsymbol{r_0}=\boldsymbol{b}-A\,\boldsymbol{x_0}$, $\boldsymbol{v_0}=A\,\boldsymbol{u_0}$, $\boldsymbol{d_0}=\boldsymbol{0}$
2. *Oblicz* $\tau_0=\|\boldsymbol{r_0}\|_2$, $\theta_0=\eta_0=0$
3. *Ustal* $\boldsymbol{r_0}^*$ *takie, aby* $\rho_0\equiv(\boldsymbol{r_0}^*,\boldsymbol{r_0})\neq 0$
4. *For* $m=0,1,2,\ldots$, *Do*:
5. *If* $m\,\%\,2=0$ *then*:
6. $\alpha_{m+1}=\alpha_m=\rho_m/(\boldsymbol{v_m},\boldsymbol{r_0}^*)$
7. $\boldsymbol{u_{m+1}}=\boldsymbol{u_m}-\alpha_m\boldsymbol{v_m}$
8. *EndIf*
9. $\boldsymbol{w_{m+1}}=\boldsymbol{w_m}-\alpha_m A\,\boldsymbol{u_m}$
10. $\boldsymbol{d_{m+1}}=\boldsymbol{u_m}+(\theta_m^2/\alpha_m)\,\eta_m\boldsymbol{d_m}$
11. $\theta_{m+1}=\|\boldsymbol{w_{m+1}}\|_2/\tau_m$; $c_{m+1}=(1+\theta_{m+1}^2)^{-\frac{1}{2}}$
12. $\tau_{m+1}=\tau_m\,\theta_{m+1}c_{m+1}$; $\eta_{m+1}=c_{m+1}^2\,\alpha_m$
13. $\boldsymbol{x_{m+1}}=\boldsymbol{x_m}+\eta_{m+1}\boldsymbol{d_{m+1}}$
14. *If* $m\,\%\,2=1$ *then*
15. $\rho_{m+1}=(\boldsymbol{w_{m+1}},\boldsymbol{r_0}^*)$; $\beta_{m-1}=\rho_{m+1}/\rho_{m-1}$
16. $\boldsymbol{u_{m+1}}=\boldsymbol{w_{m+1}}+\beta_{m-1}\boldsymbol{u_m}$
17. $\boldsymbol{v_{m+1}}=A\,\boldsymbol{u_{m+1}}+\beta_{m-1}(A\,\boldsymbol{u_m}+\beta_{m-1}\boldsymbol{v_{m-1}})$
18. *EndIf*
19. *EndDo*

W algorytmie TFQMR mnożenie wektor-macierz wykonywane jest dwukrotnie, jednakże jedno z nich pojawia się tylko w iteracjach z m nieparzystym. Należy zauważyć, że obliczenia w pętli z warunkiem na nieparzyste m wykonywane są dla wartości z m parzystym. Liczone są trzy iloczyny skalarne a przechowywanych w pamięci jest osiem wektorów.

1.5 GMRES

Metoda GMRES została opracowana w 1986 r. przez Yousefa Saada i Martina Schultza i jest ona kolejną z serii metod bazujących na podprzestrzeni Kryłowa. Została uznana za standardową metodę przy rozwiązywaniu równań macierzowych z niesymetryczną macierzą A[7]. Metoda GMRES sprowadza się do skonstruowania ortonormalnych wektorów bazowych w podprzestrzeni Kryłowa i wyznaczania za ich pomocą kolejnych przybliżeń wektora **x**, tak, aby norma z wektora reszt była jak najmniejsza.

Podobnie, jak w przypadku metody TFQMR wyznaczana jest baza wektorów ortogonalnych V_m oraz macierz trójdiagonalna T_m. Następnie rozwiązywany jest problem najmniejszych kwadratów[8]:

$$\min\|f - A[\boldsymbol{x_0}+\boldsymbol{z}]\| = \min\|\boldsymbol{r_0} - A\,\boldsymbol{z}\| \tag{21}$$

Zakładając, że $\mathbf{z} = V_m\mathbf{y}$, gdzie **y** jest dowolnym wektorem, możemy rozpatrywać minimalizowaną normę jako funkcje **y**

$$J(\boldsymbol{y}) = \|\beta \boldsymbol{v_1} - A\,V_m\,\boldsymbol{y}\| \tag{22}$$

gdzie $\beta = \|\mathbf{r_0}\|$. Wykorzystując zależność (17) otrzymujemy

$$J(\boldsymbol{y}) = \|V_{m+1}[\beta \boldsymbol{e_1} - T_m\,\boldsymbol{y}]\| \tag{23}$$

Wektor $\mathbf{e_1}$ jest pierwszą kolumną macierzy jednostkowej o wymiarach (m+1) × (m+1). Ortonormalność wektorów bazy $\mathbf{V_{m+1}}$ skutkuje upraszczeniem równania (23) do postaci:

$$J(\boldsymbol{y}) = \|\beta \boldsymbol{e_1} - T_m\,\boldsymbol{y}\| \tag{24}$$

Wtedy rozwiązanie problemu najmniejszych kwadratów wyraża się następująco

$$\boldsymbol{x_j} = \boldsymbol{x_0} + V_m\,\boldsymbol{y_j} \tag{25}$$

gdzie $\mathbf{y_j}$ minimalizuje funkcję J(**y**).

Stosowalność metody dla równań macierzowych z niesymetryczną macierzą A związana jest z koniecznością zapamiętywania pełnej bazy wektorów podprzestrzeni Kryłowa, co oznacza, że wraz ze wzrostem liczby iteracji zwiększa się liczba wektorów przechowywanych w pamięci. To sprawia, że użyteczność GMRES determinowana jest szybkością, z jaką osiąga ona zbieżność. Poniżej zamieszczono algorytm GMRES (dla zmodyfikowanej metody, w której dodatkowo następuje jej restart - co m iteracji – zapobiegający generowaniu dużej bazy) w notacji pseudokodu.

Algorytm 5. GMRES

1. *Oblicz* $\boldsymbol{r}=\boldsymbol{b}-A\,\boldsymbol{x}_{\boldsymbol{0}}$
2. *For* $j=0,1,\ldots$
3. $\beta=\|\boldsymbol{r}\|_{2,}\,\boldsymbol{v}^{\boldsymbol{0}}=\boldsymbol{r}/\beta,\,\widetilde{\boldsymbol{b}}=\beta\boldsymbol{e}^{\boldsymbol{1}}$
4. *For* $i=0,1,\ldots,m$
5. $\boldsymbol{w}=A\,\boldsymbol{v}^{\boldsymbol{i}}$
6. *For* $k=0,\ldots,i$
7. $h_{k,i}=(\boldsymbol{v}^{\boldsymbol{k}})*\boldsymbol{w};\,\boldsymbol{w}=\boldsymbol{w}-h_{k,i}\,\boldsymbol{v}^{\boldsymbol{k}}$
8. *End For* k
9. $h_{i+1,i}=\|\boldsymbol{w}\|_2;\,\boldsymbol{v}^{\boldsymbol{i+1}}=\boldsymbol{w}/h_{i+1,i}$
10. *For* $k=1,\ldots,i$
11. $\gamma=h_{k-1,i}$
12. $h_{k-1,i}=c_{k-1}\,\gamma+s_{k-1}\,h_{k,i}$
13. $h_{k,i}=-s_{k-1}\,\gamma+c_{k-1}\,h_{k,i}$
14. *End For* k
15. $\delta=\sqrt{h_{i,i}^2+h_{i+1,i}^2};\;c_i=h_{i,i}/\delta;\;s_i=h_{i+1,i}/\delta$
16. $h_{i,i}=c_i\,h_{i,i}+s_i\,h_{i+1,i}$
17. $\widetilde{\boldsymbol{b}}_{\boldsymbol{i+1}}=-s_i\,\widetilde{\boldsymbol{b}}_{\boldsymbol{i}};\;\widetilde{\boldsymbol{b}}_{\boldsymbol{i}}=c_i\cdot\widetilde{\boldsymbol{b}}_{\boldsymbol{i}}$
18. $\rho=\|\widetilde{\boldsymbol{b}}_{\boldsymbol{i+1}}\|$
19. *if* ρ odpowiednio małe *then*
20. $n_r=i\,;\,goto\,SOL$
21. *End For* i
22. $n_r=m;\;y_{nr}=\widetilde{b}_{nr}/h_{nr,nr}$
23. $SOL:\,For\,k=nr-1,\ldots,0$
24. $y_k=(\widetilde{b}_k-\sum_{i=k+1}^{n_r}h_{k,i}\,y_i)/h_{k,k}$
25. *End For* k
26. $\boldsymbol{x}=\boldsymbol{x}+\sum_{i=0}^{n_r}y_i\,\boldsymbol{v}^{\boldsymbol{i}};$ *if* ρ odpowiednio małe *quit*
27. $\boldsymbol{r}=\boldsymbol{b}-A\,\boldsymbol{x}$
28. *End For* j

W algorytmie GMRES obecne są dwie pętle: j – zewnętrzna, określająca liczbę powtórzeń oraz i – wewnętrzna, do konstrukcji bazy. Rezerwowana jest pamięć na m wektorów bazy. W przypadku, gdy warunek stopu nie zostanie spełniony, obliczenia są powtarzane. Liczba iteracji w algorytmie GMRES dla zadanego problemu macierzowego może być regulowana poprzez odpowiednie dobranie wymiaru bazy $V_{\mathbf{m}}$. W liniach 10-17 zastosowana została rotacja Givensa, mająca na celu doprowadzenie macierzy H do postaci trójkątnej. Dzięki temu wektor **y** stanowiący rozwiązanie układu równiań znajdujemy stosując „postępowanie odwrotne”. Aby zapewnić stabilność algorytmu przeprowadzona jest również ortogonalizacja Grama-Schmidta, jednak odbywa się to kosztem wydajności.

2. Biblioteka numeryczna

2.1 Opis programu

Celem niniejszej pracy była implementacja wybranych algorytmów do rozwiązywania algebraicznych układów równań liniowych i zawarcie ich w jednym pakiecie w postaci biblioteki numerycznej. Biblioteka została napisana w języku C. Zimplementowano pięć algorytmów numerycznych (metodę sprzężonego gradientu CG, metodę kwadratowego gradientu sprzężonego CGS, BICGSTAB, TFQMR oraz GMRES) działających na ciele liczb rzeczywistych oraz zespolonych.

Macierze, dla których wykonano testowanie procedur numerycznych, zostały pobrane z serwisu Matrix Market[9] a następnie przekonwertowane poprzez zaimplementowaną procedurę do formatu CSR, który jest formatem natywnym biblioteki. Format CSR (*compressed sparse row*) przedstawia macierz M w postaci trzech jednowymiarowych tablic. Pierwsza z nich zawiera wartości niezerowych elementów, druga - indeksy kolumn a trzecia numery elementów (indeksy tabeli z wartościami), stanowiącymi pierwsze niezerowe elementy w danym wierszu (określonym poprzez indeks tabeli wierszy).

Testowe wyniki otrzymane w wyniku wywołania odpowiednich procedur zostały zweryfikowane poprzez wyznaczenie normy z wektora reszt, która – jak oczekujemy – granicznie maleje do zera w kolejnych iteracjach, aczkolwiek nie jest to gwarantowane w metodach iteracyjnych.

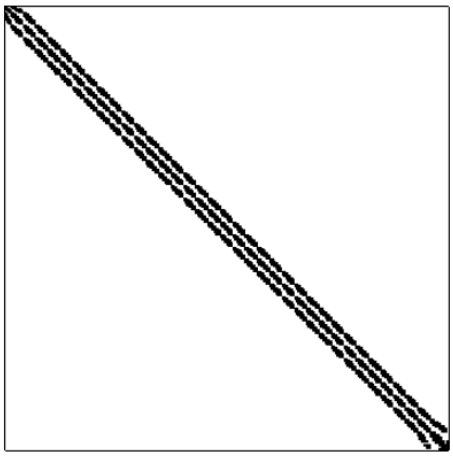
2.2 Badania zbieżności i wydajności

Użyteczność zaimplementowanych algorytmów została przetestowana na pięciu wybranych macierzach pobranych z witryny MatrixMarket. Macierze różnią się właściwościami tak, aby testy objęły jak najszerszy zakres przypadków. Na podstawie uzyskanych danych naszkicowane zostały wykresy zależności logarytmu dzisiętnego z normy wektora reszt od kolejnych iteracji. Obrazują one sposób oraz szybkość osiągania zbieżności. Algorytmy kończyły działanie, gdy wartość normy stawała się mniejsza od arbitralnie ustalonej wartości 10^{-6}. Należy mieć na uwadze, że liczba iteracji nie jest równoznaczna z czasem wykonywania się algorytmu – w zależności od zastosowanej metody ilość i rodzaj wykonywanych operacji jest różny. W Tabeli 1 przedstawiona jest krótka charakterystyka testowych macierzy.

Tabela 1. Charakterystyka macierzy.

Nazwa	N	NNZ	Typ	Rodzaj elementów
bcsstk27	1224	56126	symetryczna, dodatniookreślona	rzeczywiste
fs_760_1	760	5976	niesymetryczna	rzeczywiste
sherman4	1104	3786	niesymetryczna	rzeczywiste
dwg961b	961	10591	symetryczna	zespolone
qc324	324	26730	symetryczna	zespolone

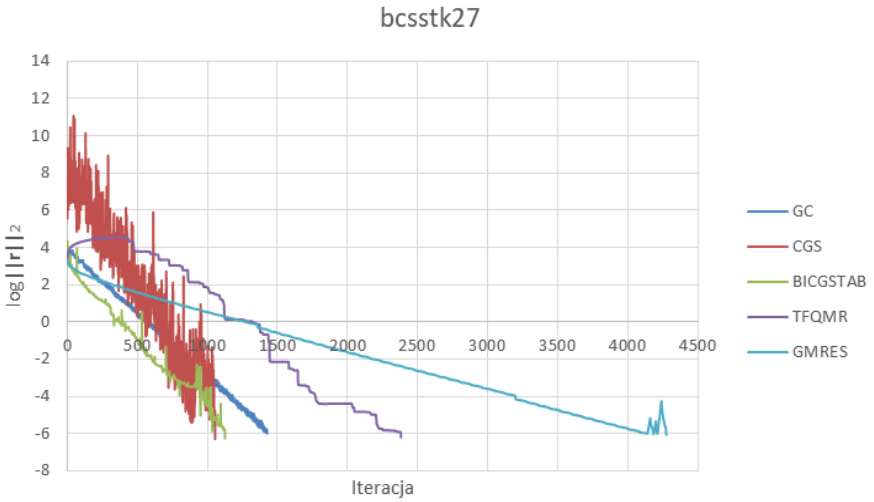
Pierwszą z testowanych macierzy była symetryczna macierz o nazwie bcsstk27 z elementami rzeczywistymi. Każdy z pięciu algorytmów okazał się zbieżny. Najmniejszą liczbę iteracji potrzebował algorytm BICGSTAB, największą – GMRES. Norma z wektora reszt dążyła monotonicznie do zera dla każdego z algorytmów z wyjątkiem CGS. Tutaj wykres jest szeroki i poszarpany.



Rysunek 1. Wizualizacja niezerowych elementów macierzy bcsstk27.

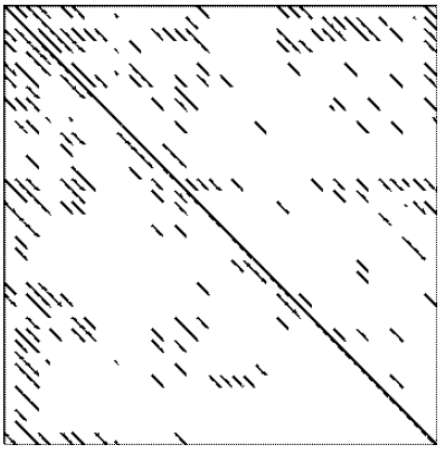
Źródło: MatrixMarket

.



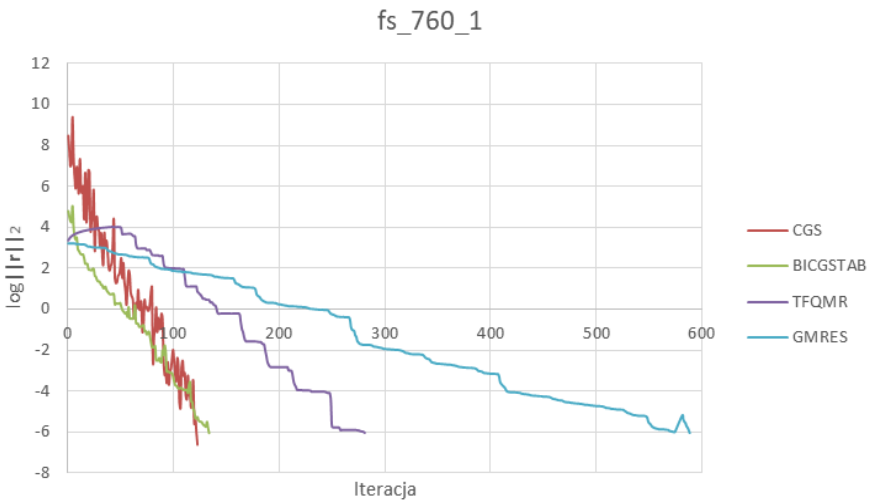
Wykres 1. Norma wektora reszt w kolejnych iteracjach dla macierzy bcsstk27.

Następnie algorytmy wywołane zostały na macierzy fs_760_1 zawierającej dane uzyskane z pomiarów kinetyki reakcji chemicznej. fs_760_1 również składała się z elementów rzeczywistych. Ze względu na brak symetryczności algorytm CG, zgodnie z przewidywaniami nie osiągnął zbieżności. Każdy z pozostałych algorytmów zwrócił wynik z zadaną dokładnością. Ponownie, najmniejszą liczbę iteracji wymagały algorytmy CGS i BICGSTAB, a największą GMRES.



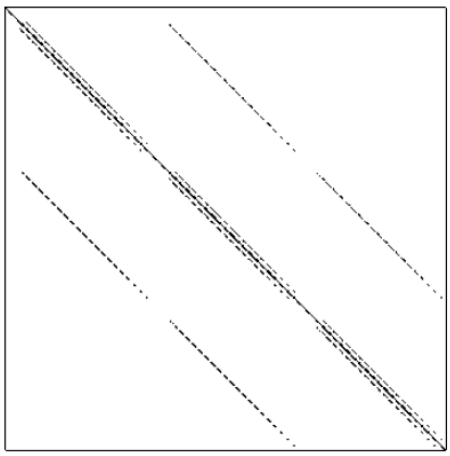
Rysunek 2. Wizualizacja niezerowych elementów macierzy fs_760_1.

Źródło: MatrixMarket.



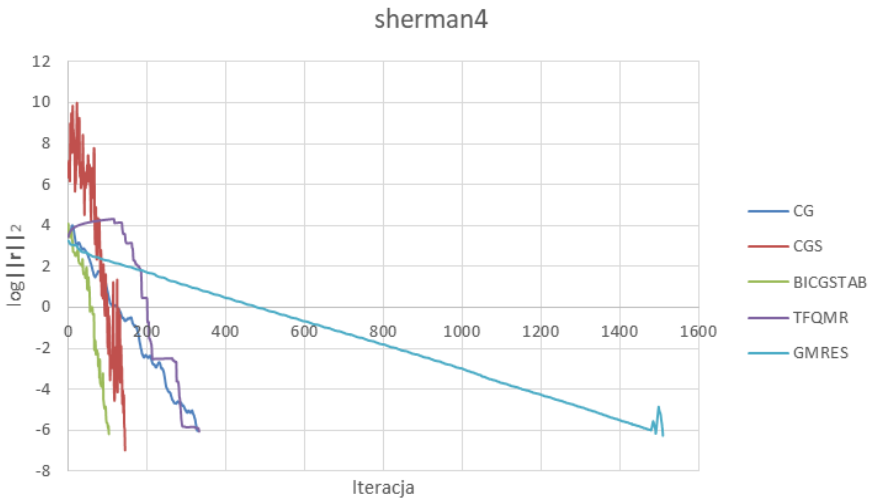
Wykres 2. Norma wektora reszt w kolejnych iteracjach dla macierzy fs_760_1.

Kolejną z macierzy była macierz sherman4 uzyskana w wyniku symulacji przepływu cieczy lepkiej. Pomimo braku symetryczności macierzy, algorytm gradientu sprzężonego okazał się zbieżny w niewielkiej liczbie iteracji. Najszybciej zbiegającym się algorytmem pozostał BICGSTAB.



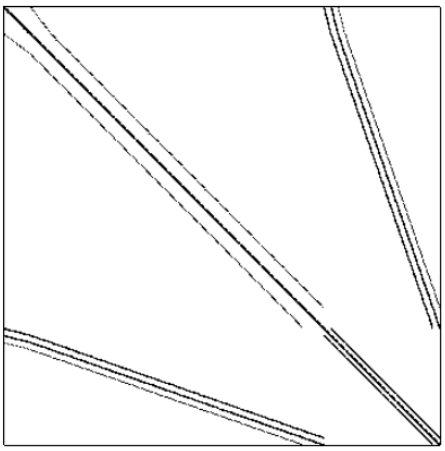
Rysunek 3. Wizualizacja niezerowych elementów macierzy sherman4.

Źródło: MatrixMarket.



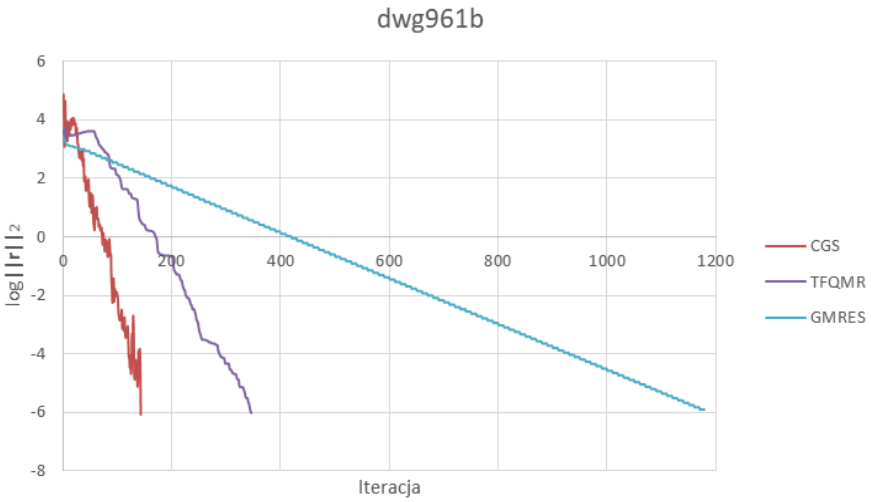
Wykres 3. Norma wektora reszt w kolejnych iteracjach dla macierzy sherman4.

Pozostałe dwie macierze (dwg961b i qc324) składały się z elementów zespolonych. Dla obu z nich algorytmy CG i BICGSTAB nie osiągnęły zbieżności. Najmniejszą liczbę iteracji wymagał algorytm CGS, a przebieg wykresu dla algorytmu GMRES był liniowy.

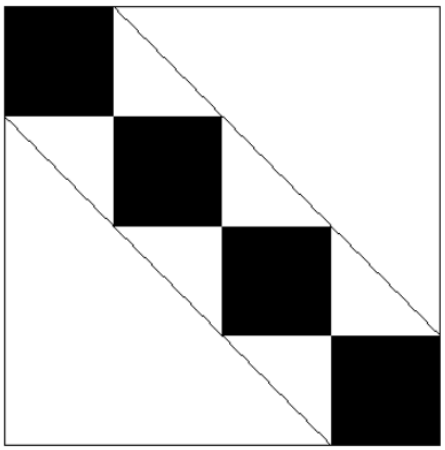


Rysunek 4. Wizualizacja niezerowych elementów macierzy dwg961b.

Źródło: MatrixMarket.

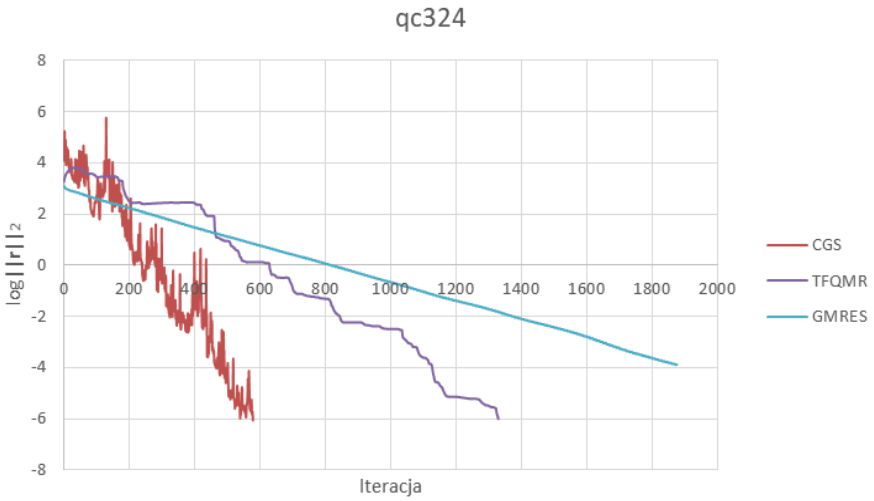


Wykres 4. Norma wektora reszt w kolejnych iteracjach dla macierzy dwg961b (algorytm BICGSTAB niezbieżny).



Rysunek 5**.** Wizualizacja niezerowych elementów macierzy qc324.

Źródło: MatrixMarket.



Wykres 5. Norma wektora reszt w kolejnych iteracjach dla macierzy qc324 (algorytm BICGSTAB niezbieżny).

2.3 Wnioski

Celem niniejszej pracy była implementacja algorytmów numerycznych służących do rozwiązywania układów równań liniowych dla macierzy rzadkich. Cel ten został zrealizowany zarówno dla macierzy o elementach rzeczywistych jak i zespolonych. Zaimplementowanych zostało pięć algorytmów, których poprawność została sprawdzona na pięciu testowych macierzach.

W wyniku przeprowadzonych benchmarków stwierdzono, że najszybszą z testowanych metod jest CGS. Zastosowanie kwadratowych wielomianów macierzowych skutkowało znacznym wzrostem wydajności, jednak kosztem dokładności algorytmu. Najwolniej zbieżną metodą okazał się GMRES. Przypuszcza się, że przeprowadzenie optymalizacji polegającej na doborze najkorzystniejszej liczby wektorów bazowych (parametr m) mogłoby w znacznym stopniu poprawić wydajność metody. Pozostałe dwie metody (BICGSTAB i TFQMR) charakteryzowały się średnią wydajnością. Analizując dane z Tabeli 1 zauważyć można, że liczba iteracji wzrasta wraz z rozmiarem macierzy i jest ona determinowana zarówno liczbą wierszy jak i elementów niezerowych.

Wykresy 4 oraz 5 obrazują działanie algorytmów na macierzach o elementach zespolonych. Zaimplementowane metody, za wyjątkiem BICGSTAB, okazały się równie skuteczne co w przypadku macierzy rzeczywistych. BICGSTAB nie doprowadził do osiągnięcia zbieżności.

Stworzona biblioteka jest intuicyjna w użyciu. Znaczna część zaimplementowanych mechanizmów jest transparentna dla użytkownika. Parametry przyjmowane przez każdą z metod są niemal identyczne (wyjątek stanowi GMRES), ustandaryzowane zostały też zwracane przez

procedury kody błędów. Biblioteka nie wymaga instalacji dodatkowych rozszerzeń, jest w pełni gotowa do użytku w programach napisanych w języku C.

Bibliografia

[1] Garcia, Alejandro L. *Numerical methods for physics*. Englewood Cliffs, NJ: Prentice Hall, 2000.

[2] Saad, Yousef. *Iterative methods for sparse linear systems*. Vol. 82. siam, 2003.

[3] Hestenes, Magnus R. „The conjugate gradient method for solving linear systems.*"Proc. Symp. Appl. Math VI, American Mathematical Society.* 1956.

[4] Golub, Gene H., and Dianne P. O'Leary. "Some history of the conjugate gradient and Lanczos algorithms: 1948–1976." *SIAM review* 31.1 (1989): 50-102.

[5] Fokkema, Diederik R., Gerard LG Sleijpen, and Henk A. Van der Vorst. "Generalized conjugate gradient squared." *Journal of Computational and Applied Mathematics* 71.1 (1996): 125-146.

[6] Widlund, Olof. "A Lanczos method for a class of nonsymmetric systems of linear equations." *SIAM Journal on Numerical Analysis* 15.4 (1978): 801-812.

[7] Van Der Vorst, Henk A. "Efficient and reliable iterative methods for linear systems." *Journal of Computational and Applied Mathematics* 149.1 (2002): 251-265.

[8] Saad, Youcef, and Martin H. Schultz. "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems." *SIAM Journal on scientific and statistical computing* 7.3 (1986): 856-869.

[9] https://math.nist.gov/MatrixMarket/ [dostęp z dnia 21.12.2019].

Dodatek

Poniżej zamieszczono definicje zaimplementowanych procedur.

```
int CG(const int * const rows, const int * const columns, const double * const values, const int ntot, const int nnz, double * const b, double * const x, const double err, const int maxIter, int * const mess);

int CGS(const int * const rows, const int * const columns, const double * const values, const int ntot, const int nnz, const double * const b, double * const x, const double err, const int maxIter, int * const mess);

int BICGSTAB(const int * const rows, const int * const columns, const double * const values, const int ntot, const int nnz, const double * const b, double * const x, const double err, const int maxIter, int * const mess);

int TFQMR(const int * const rows, const int * const columns, const double * const values, const int ntot, const int nnz, const double * const b, double * const x, const double err, const int maxIter, int * const mess);

int GMRES(const int * const rows, const int * const columns, const double * const values, const int ntot, const int nnz, const double * const b, double * const x, const double err, const int m, const int maxIter, int * const mess);


int CGComplex(const int * const rows, const int * const columns, const complex double * const values, const int ntot, const int nnz, const double complex const * b, double complex * const x, const double err, const int maxIter, int * const mess);

int CGSComplex(const int * const rows, const int * const columns, const complex double * const values, const int ntot, const int nnz, const double complex const * b, double complex * const x, const double err, const int maxIter, int * const mess);

int BICGSTABComplex(const int * const rows, const int * const columns, const complex double * const values, const int ntot, const int nnz, const double complex const * b, double complex * const x, const double err, const int maxIter, int * const mess);
```

```
int TFQMRComplex(const int * const rows, const int * const columns, const complex double * const values, const int ntot, const int nnz, const double complex const * b, double complex * const x, const double err, const int maxIter, int * const mess);

int GMRESComplex(const int * const rows, const int * const columns, const complex double * const values, const int ntot, const int nnz, const double complex const * b, double complex * const x, const double err, const int m, const int maxIter, int * const mess);
```

Gdzie:

rows – tablica wierszy w formacie CSR,

columns – tablica kolumn w formacie CSR,

values – tablica wartości w formacie CSR,

ntot – rozmiar macierzy (liczba wierszy/kolumn),

nnz – liczba niezerowych elementów,

b – wektor wyrazów wolnych,

x – szukany wektor,

err – wartość normy określająca zakończenie procedury,

maxIter – maksymalna liczba iteracji, po przekroczeniu tej wartości procedura kończy działanie,

mess – komunikat zwrotny,

m – liczba wektorów bazy w algorytmie GMRES.

Znaczenie komunikatów zwracanych poprzez parametr **mess**:

0 – wywołanie prawidłowe,

1 – nieprawidłowe wartości w tablicy wierszy,

2 – nieprawidłowe wartosci w tabeli kolumn,

3 – nieprawidłowe elementy w tabeli wartości (`NAN` lub `INFINITY`),

4 – nieprawidłowy parametr **maxIter**,

5 – przekroczono maksymalną liczbe iteracji,

6 – obliczana norma otrzymała wartość `NAN` lub `INFINITY`.