// Autor: Daniel Szarek
//=====================

Zawartość folderu z zadaniem:

main.cpp – Kod z rozwiązaniem zadania napisanym w języku C++
funkcje.h – Biblioteka z własnymi funkcjami wykorzystywanymi do rozwiązania zadania N7
opracowanie.pdf – Ten dokument z opracowaniem zadania N7
funkcje.cpp – Plik zawierający implementację funkcji podanych w własnej bibliotece funkcje.h
Makefile – Do uruchomienia programu main.cpp (Makefile oferuje możliwość uruchomienia programu za pomocą komendy ‘make run’, usunięcia plików po uruchomieniu programu komendą ‘make clean’ oraz możliwość utworzenia paczki .tar.gz z zawartością foldera z zadaniem za pomocą komendy ‘make tar’
wyniki.txt – Plik tekstowy z wynikami programu main.cpp

Do zrealizowania zadania wykorzystałem GSL - GNU Scientific Library, w Makefile zawarłem komendy do uruchomienia zadania na swojej maszynie w systemie Ubuntu, w celu uruchomienia zadania na swoim sprzęcie należy w prawidłowy sposób zainstalować bibliotekę GSL oraz zmienić scieżkę INCLUDEPATH1 dla swojej maszyny.

W swoich plikach źródłowych zamieszczam liczne komentarze, które na bieżąco tłumaczą działanie oraz funkcjonalności mojego kodu. W tym dokumencie przedstawię omówienie metod wykorzystanych do rozwiązania zadania.

Treść Zadania:

N7 *Zadanie numeryczne*
Zaimplementować metodę gradientów sprzężonych dla układu z zadania N6 z poprzedniego zestawu.

Układ z zadania N6

$$A = \begin{pmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix}, \qquad b = \begin{pmatrix} 2 \\ 6 \\ 2 \end{pmatrix}$$

Rozwiązanie zadania:

```
vboxuser@Ubuntu:~/Desktop/MetodyNumeryczne/Zadanie7$ make run
ar rsv libMojeFunkcje.a funkcje.o
ar: creating libMojeFunkcje.a
a - funkcje.o
mkdir -p ./lib
mv libMojeFunkcje.a ./lib
g++ -o main.x main.o -Wall -std=c++11 -L./lib -lMojeFunkcje -lgsl -lgslcblas
./main.x
x[0] = 1
x[1] = 2
x[2] = 1
```

Omówienie:

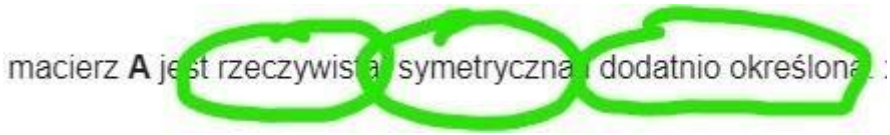
Swoje rozwiązanie oparłem na artykule z Wikipedii i zaimplementowałem wskazany tam algorytm.

Wynikowy algorytm [edytuj | edytuj kod]

Upraszczając, otrzymujemy poniższy algorytm rozwiązujący **Ax** = **b**, gdzie macierz **A** jest rzeczywista, symetryczna i dodatnio określona. $\mathbf{x}_0$ jest punktem startowym.

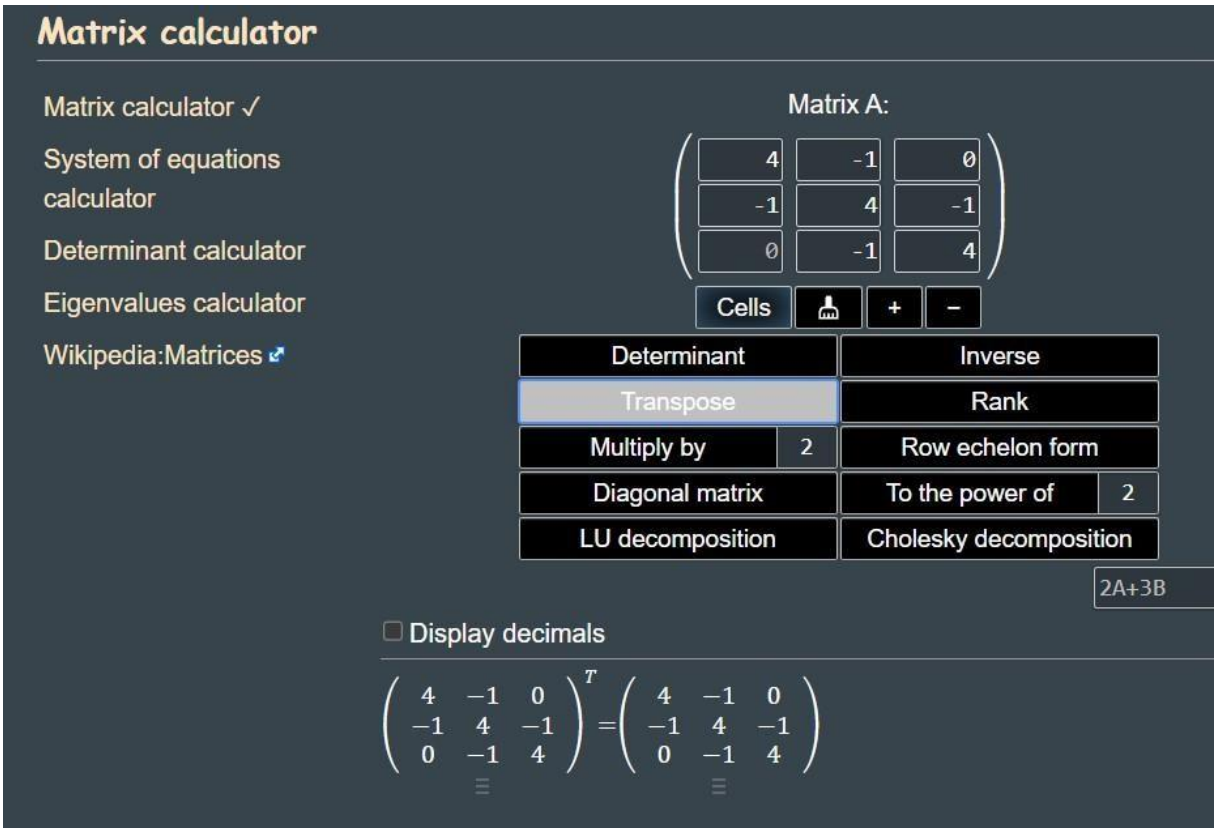
$r_0 := b - Ax_0$
$p_0 := r_0$
$k := 0$
repeat
$\alpha_k := \frac{r_k^\top r_k}{p_k^\top A p_k}$
$x_{k+1} := x_k + \alpha_k p_k$
$r_{k+1} := r_k - \alpha_k A p_k$
if r_{k+1} jest "wystarczająco mały" **then** exit loop **end if**
$\beta_k := \frac{r_{k+1}^\top r_{k+1}}{r_k^\top r_k}$
$p_{k+1} := r_{k+1} + \beta_k p_k$
$k := k + 1$
end repeat
Wynikiem jest x_{k+1}

Zwróciłem szczególną uwagę na podane powyżej wymagania odnośnie zastosowania tej metody i sprawdziłem czy nasz układ je spełnia.



- Nasza macierz A jest rzeczywista, ponieważ 0,-1 oraz 4 należą do zbioru liczb rzeczywistych. Innymi słowy wszystkie jej elementy należą do zbioru liczb rzeczywistych.
- Macierz A jest symetryczna, ponieważ równa jest ona swojej transpozycji.

Do wizualizacji tego podpunktu posłużę się wykorzystaniem strony: https://matrixcalc.org/



- Ostatni warunek jest najbardziej pracochłonny do sprawdzenia w naszym przykładzie. Podobnie jak w poprzednich podpunktach będzie on spełniony. Nasza macierz A będzie dodatnio określona. Do udowodnienia tego wykorzystam Kryterium Sylvestera.

Kryterium Sylvestera [edytuj]

10 języków

Artykuł Dyskusja Czytaj Edytuj Edytuj kod źródłowy Wyświetl historię Narzędzia

Kryterium Sylvestera – kryterium pozwalające badać dodatnią (lub ujemną) określoność symetrycznej macierzy. Nazwa pochodzi od brytyjskiego matematyka J. J. Sylvestera.

Kryterium Sylvestera [edytuj | edytuj kod]

Niech

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

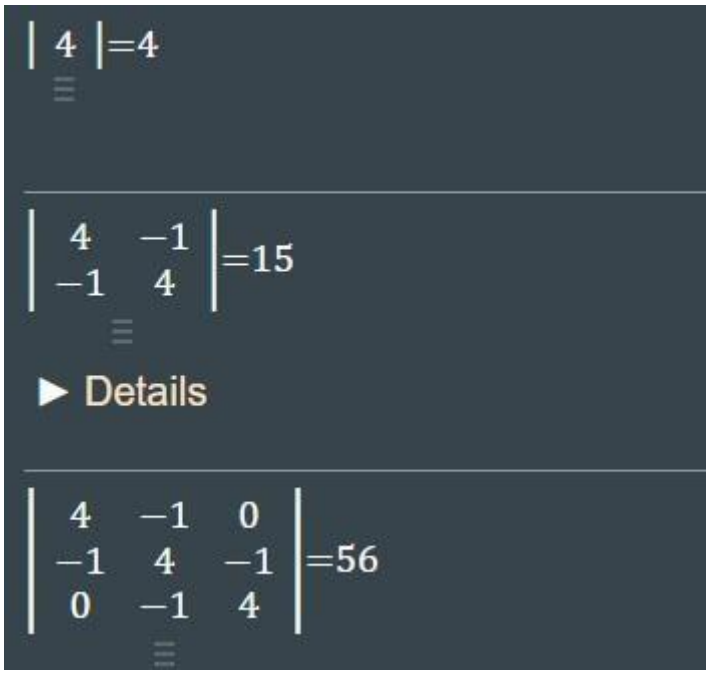
będzie macierzą symetryczną o współczynnikach rzeczywistych.

Niech ponadto

$$M_1 = a_{11}, \quad M_2 = \det \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \dots \quad M_l = \det \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1l} \\ a_{21} & a_{22} & \dots & a_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{ll} \end{bmatrix}$$

Wówczas

Oraz wcześniej wspomnianą stronę https://matrixcalc.org/



Jak widać wszystkie nasze wyznaczniki są > 0, stąd macierz A będzie dodatnio określona.

Otrzymaliśmy zgodność z tym, że do rozwiązania naszego układu równań możemy zastosować metodę gradientów sprzężonych.

W mojej implementacji gradientów sprzężonych funkcja w pierwszej kolejności tworzy wektory pomocnicze r_k oraz p_k. Następnie wypełniam wektor rezyduum (r_k) oraz wektor sprzężony (p_k). Warto zauważyć, że wektor początkowy x0 jest wypełniony zerami, więc wartości wektora r_k będą równe wartościom wektora b. Kolejno następuje obliczenie następnych wartości x. W tym celu obliczam wartości alpha_k oraz beta_k podane we wzorze. Do tego wykorzystuję funkcję z biblioteki GSL np. do obliczenia iloczynu skalarnego wektorów gsl_blas_ddot oraz własnoręcznie napisane funkcję np. do obliczenia iloczynu macierzy A oraz wektora p_k nazwaną iloczynSkalarnyWektoraIMacierzy, wynikiem tej funkcji będzie wektor Ap_k przechowujący kolejne iloczyny skalarne wierszy macierzy A oraz wektora p_k. Do obliczenia następnych iteracji x, r_k oraz p_k wykorzystałem inną własną funkcję nazwaną kolejnaIteracja. W funkcji sprawdzenieDokladnosciObliczen z każdą iteracją programu sprawdzam, czy kolejna iteracja r_k jest wystarczająco mała. Wykorzystuje do tego porównanie normy z następnej iteracji r_k z moim kryterium dokładności. Za swoje kryterium dokładności użyłem epsilona równego 10e-10, tak jak w poprzednim zadaniu numerycznym N6, zgodnie z poleceniem zadania N7. Jeżeli norma jest mniejsza niż epsilon funkcja, przerywa działanie pętli oraz na koniec zwalnia pamięć z wektorów GSL.