Poprawa rozwiązania zadań numerycznych z zestawów 1, 2,3
Autor: Mariusz Adamczyk

Załączam fragment otrzymanego od Pana mail'a (nieco zmodyfikowany – na niebiesko zadania które wymagały korekty , na czerwono numery zadań które nadal niestety zwracają niedokładne rozwiązania, na brązowo numery zadań których poprawności wyniku nie weryfikowałem z wynikami opartymi o wbudowane funkcje Matlab) :

komentarze do projektów:

Zestaw1:

Zad10: 1/1, ok

Zad12: 2/2, ok

Zad13: 0,5/1 czy funkcja inv odwraca macierz z definicji (co jest bardzo złe) czy jakimś numerycznym algorytmem (są takie, np. algorytm Jordana)? Proszę to sprawdzić np. w dokumentacji i skomentować albo jawnie rozwiązać układy równań jednym z algorytmów omawianych na zajęciach.

Zestaw2:

Zad5a: 3/3 ok

Zad5b: 1/1 ok

Jedna uwaga: ponieważ najciekawsze rzeczy ze zbieżnością dzieją się w zakresie 0-50 lepiej byłoby przedstawić tam wyniki gęściej.

Zestaw3:

Zad8: 2/2 ok

Zad9: 0/3 chyba musi Pan jeszcze przemyśleć na czym ta diagonalizacja polega: procedura trójdiagonalizacji Householder'a jest pomocniczym etapem dla przyspieszenia obliczeń, natomiast główna część to algorytm: i) rozkładamy macierz początkową A1=Q1R1, ii) budujemy nową macierz A2=R1Q1, iii) rozkładamy macierz A2=Q2R2, iv) budujemy kolejną macierz A3=R2Q2… i tak dalej. Powtarzamy to wiele razy (np. 100) i dopiero powiedzmy R100 (która zawsze jest trójkątna, bo tak prowadzimy rozkład) może będzie miała wartości własne na diagonali. Wektory własne zaś bierzemy z akumulowanego iloczynu Q100…*Q1.*

Zad10: 1/1 ok
Zad12: 0/2 dobrze Pan zaczyna ale nie rozumiem czemu nie wykonuje Pan dalej wielokrotnych iteracji rozwiązania tego równania – przecież to metoda potęgowa (choć odwrotna) – musimy wielokrotnie iterować, jak można było zapomnieć o tym kluczowym elemencie :]?

Zad: 13N, zestaw 1

Komentarz do funkcji *inv():*
Funkcja wykorzystuje stworzoną dla języka *Fortran 77* bibliotekę *LAPACK (Linear Algebra Package)*. Macierz odwracana jest z pomocą algorytmu rozkładu macierzy do postaci SVD zaproponowanego przez Jacobi'ego. Gdyby macierz była odwracana z definicji to nie można odwrócić macierzy której wyznacznik równy jest 0, poza tym odwracanie definicyjne wymaga większej ilości czasu oraz posiada mniejszą dokładność numeryczną.

Być może lepszym pomysłem w tym zadaniu byłoby użycie zdefiniowanego w Matlab operatora „\" dedykowanego do rozwiązywania układów równań liniowych Ax = b, wtedy x = A\b. Rozwiązanie układu Ax = b znajdowane jest wtedy z wykorzystaniem eliminacji Gauss'a.

Poniżej poprawione zadanie z moją implementacją rozwiązywania układu równań liniowych:

```
%funkcja rozwiązująca układ Ax = B
%z uzyciem metody eliminecji Gauss'a
%autor: Mariusz Adamczyk,
%ostatnia modyfikacja: 14.12.2013r.
%-----------------------------------------------

function [x] = mojGauss(A, B)

x = ones(length(A));
x = x(:,1);

%moja implementacja metody Gaussa
%bez pivoting'u
% i - nr wiersza
% j - nr kolumny
for j = 1: length(A) - 1
    for i = 1 + j : length(A)
        if(A(i,j) ~=0 )
            a = A(i,j)/A(j,j);
            C = A(j, :)*a;
            A(i, :) = A(i, :) - C;
            B(i) = B(i) - B(j) * a;
        else continue
        end
    end
end

%wyliczanie wektora x
for k = length(A) : -1 : 1
    x(k) = B(k)/A(k,k);
    A(1:k-1, k) = A(1:k-1, k) * x(k);
    B(1:k-1) = B(1:k-1) - A(1:k-1, k);
end

end
%koniec funkcji, Mariusz Adamczyk
%-----------------------------------------------
```

```
%zad 13N zestaw z 06.11.2013r.
```

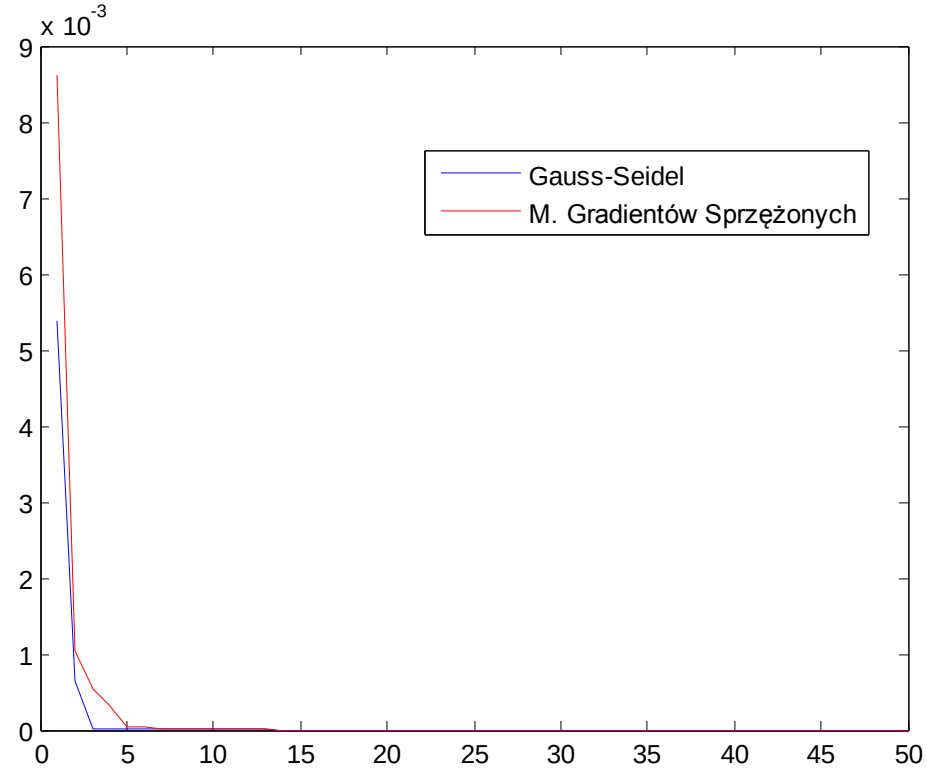
```
%autor: Mariusz Adamczyk,
%ostatnia modyfikacja: 14.12.2013r.
%-----------------------------------------------
clear all
clc

%funkcja obliczająca współczynnik uwarunkowania macierzy A,
%definiowany jako cond(A) = ||A||?||A-1||, stanowiący
%współczynnik z jakim błędy wejściowe przenoszą się na wyjście
%podczas operacji macierzowej; im większa jest wartość
%współczynnika uwarunkowania macierzy tym większa jest jej
%wra?liwość na błędy zaokrągleń podczas wykonywania operacji
%arytmetycznych

A = [-116.66654   583.33346  -333.33308  100.00012  100.00012;
      583.33346  -116.66654  -333.33308  100.00012  100.00012;
     -333.33308  -333.33308   133.33383  200.00025  200.00025;
      100.00012   100.00012   200.00025  50.000025 -649.99988;
      100.00012   100.00012   200.00025 -649.99988  50.000025];

wspUwarunkowania_A = cond(A);

b1 = [-0.33388066;  1.08033290; -0.98559856; 1.31947922; -0.09473435];
b2 = [-0.33388066;  1.08033290; -0.98559855; 1.32655028; -0.10180541];
b3 = [ 0.72677951;  0.72677951; -0.27849178; 0.96592583;  0.96592583];
b4 = [ 0.73031505;  0.73031505; -0.27142071; 0.96946136;  0.96946136];

%LAPACK
% z1 = inv(A)*b1;
% z2 = inv(A)*b2;
% z3 = inv(A)*b3;
% z4 = inv(A)*b4;

z1 = mojGauss(A, b1);
z2 = mojGauss(A, b2);
z3 = mojGauss(A, b3);
z4 = mojGauss(A, b4);

%długości wektorów
D_b1_minus_b2 = sqrt(sum((b1-b2).^2));
D_b3_minus_b4 = sqrt(sum((b3-b4).^2));

Il_1 = sqrt(sum((z1-z2).^2))/D_b1_minus_b2;
Il_2 = sqrt(sum((z3-z4).^2))/D_b3_minus_b4;

%układ źle uwarunkowany?

%koniec zad13N Mariusz Adamczyk
%-------------------------------------------------------
```

Uznaję, że głównym elementem poprawy tego zadania jest znalezienie wektorów z_i bez użycia funkcji *inv().* Poniżej porównanie wektorów znalezionych z użyciem *inv()* oraz tych obliczonych wg mojej

implementacji metody eliminacji Gauss'a. (implementacja zdaje się dawać poprawne wyniki, choć nie są one dokładnie takie same jak te z wykorzystaniem funkcji *inv()*).

inv()	*mojGauss(A, B)*
z1 =	z1 =
0,00196304996370100	0,00196304996373809
-5,72551219644168e-05	-5,72551219761944e-05
-0,000259268641844557	-0,000259268641743685
0,000220741637443567	0,000220741637493935
-0,00179956373674273	-0,00179956373669037
z2 =	z2 =
0,00196562396121500	0,00196562396128724
-5,46811244781509e-05	-5,46811244270505e-05
-0,000254120634252786	-0,000254120634145397
0,000233417158618465	0,000233417158663034
-0,00180709124689400	-0,00180709124683454
z3 =	z3 =
364,019900458131	364,019900484338
364,019900458131	364,019900484338
728,037635852779	728,037635905193
364,018105821937	364,018105848144
364,018105821937	364,018105848144
z4 =	z4 =
367,660091053431	367,660091079900
367,660091053431	367,660091079900
735,318017043367	735,318017096306
367,658295893964	367,658295920434
367,658295893965	367,658295920434

zestaw 2, wykres dokładniej ilustrujący co dzieje się między kolejnymi iteracjami



Zad: 9N, zestaw 3

Niestety nie udało mi się nadal uzyskać poprawnych wyników. Stosowany przeze mnie rozkład QR na pewno działa poprawnie (sprawdziłem). Błąd musi zatem tkwić gdzieś w „iterowaniu”, ale nie umiem go znaleźć, mimo, że to ledwie kilka linijek kodu.

```
%funkcja znajdująca rozkład QR macierzy wejściowej
%autor: Mariusz Adamczyk,
%ostatnia modyfikacja: 14.12.2013r.
%-----------------------------------------------

function [Q,R] = mojQR(A)
    for i = 1:length(A)-1
       a = A(:,i);
       n = 1;
       while n < i
          a(n) = 0;
          n = n+1;
       end
       e = zeros(length(A));
       e = e(:,1);
       e(i) = 1;
```

```
        v = a + norm(a)*e;
        b = v'*v;
        c = v*v';
        H = eye(length(A)) - 2*c/b;
        A = H*A;
        if i == 1
            Q_trans =H;              %Góra w4 slajd 8
        else
            Q_trans = H*Q_trans;
        end
    end
    %macierz A po hausholderze
    Q = Q_trans';
    R = A;
end
%czyli powyżej mamy pojedynczy rozkład QR

%koniec funkcji, Mariusz Adamczyk
%-----------------------------------------------
```

```
%zad 9N zestaw z 20.11.2013r.
%autor: Mariusz Adamczyk,
%ostatnia modyfikacja: 14.12.2013r.
%-----------------------------------------------
clear all
clc

A = [ 19/12  13/12  5/6  5/6   13/12  -17/12;
      13/12  13/12  5/6  5/6  -11/12   13/12;
       5/6    5/6   5/6  -1/6   5/6     5/6;
       5/6    5/6  -1/6   5/6   5/6     5/6;
     13/12  -11/12  5/6   5/6   13/12  13/12;
     -17/12 13/12   5/6   5/6   13/12  19/12];

 Awej = A;

% A = [1 2 -1;
%      1 4 5;
%      1 4 1 ];

[V,D] = eig(A);      %liczy w. własne i odpowiadające im wektory dla
sprawdzenia

%najpierw Householder do macierzy trójkątnej
%czyli mamy ortogonale Q
for i = 1 :1000
    [Q, R] = mojQR(Awej);
    %Awej-Q*R   działa dobrze na pewno QR bo różnice rzędu e-15
    Awej = R*Q;
    if i==1
        Qkum = Q;
    else
        Qkum = Q*Qkum;
    end
end
%Awej po ilus iter ma na diagonali wartości własne
%Qkum to odpow im wektory
T = R*Q;         %trójdiagonalna z w. włas
```

```
%koniec zad9N Mariusz Adamczyk
%------------------------------------------------------
```

Uzyskane wyniki:

T = (wartości własne macierzy A na diagonali)

4.0000	-0.0000	0.0000	0.0000	-0.0000	-0.0000
0.0000	**3.0000**	-0.0000	0.0000	-0.0000	-0.0000
-0.0000	0.0000	**0.6667**	-1.8856	-0.0000	-0.0000
-0.0000	0.0000	-1.8856	**-0.6667**	-0.0000	0.0000
-0.0000	0.0000	-0.0000	-0.0000	**1.0000**	-0.0000
0.0000	-0.0000	0.0000	0.0000	-0.0000	**-1.0000**

Qkum = (kolejne kolumny to wektory własne odpowiadające wartościom na powyższej diagonali)

0.5095	-0.5019	0.0701	0.1032	0.6048	0.3274
0.3696	0.2455	-0.3758	0.2336	-0.4488	0.6371
-0.5402	-0.5773	0.2592	0.0308	-0.3084	0.4601
0.0608	0.2418	0.6282	0.7352	-0.0246	-0.0448
0.2772	0.2533	0.6225	-0.6269	-0.1244	0.2510
-0.4811	0.4816	-0.0675	-0.0140	0.5671	0.4585

Wyniki otrzymane dzięki gotowym funkcjom:

D =

-2.0000	0	0	0	0	0
0	-1.0000	0	0	0	0
0	0	1.0000	0	0	0
0	0	0	2.0000	0	0
0	0	0	0	3.0000	0
0	0	0	0	0	4.0000

V =

0.5000	-0.2887	0.0000	-0.0000	0.7071	0.4082
-0.5000	-0.2887	0.0000	0.7071	0.0000	0.4082
0.0000	0.5774	0.7071	-0.0000	-0.0000	0.4082
0.0000	0.5774	-0.7071	0.0000	-0.0000	0.4082
-0.5000	-0.2887	-0.0000	-0.7071	-0.0000	0.4082
0.5000	-0.2887	0.0000	0.0000	-0.7071	0.4082

Wyniki uzyskane dzięki mojej implementacji nie są identyczne z wynikami uzyskanymi dzięki gotowym funkcjom **. Niestety nie udało mi się nadal uzyskać poprawnych wyników. Stosowany przeze mnie rozkład QR na pewno działa poprawnie (sprawdziłem). Błąd musi zatem tkwić gdzieś w „iterowaniu", ale nie umiem go znaleźć, mimo, że to ledwie kilka linijek kodu. Błędnie wyznaczane są dwie wartości własne, co za tym idzie kumulowanie wektorów własnych także obarczone jest błędami.**

Zad: 12N, zestaw 3

```
%zad 12N zestaw z 20.11.2013r.
%autor: Mariusz Adamczyk,
%ostatnia modyfikacja: 16.12.2013r.
%algorytm za: http://osilek.mimuw.edu.pl/index.php?title=MN13
%-----------------------------------------------
clear all
clc

A = [2 -1 0 0 1;
     -1 2 1 0 0;
     0 1 1 1 0;
     0 0 1 2 -1;
     1 0 0 -1 2];

lam = 0.38197;
B = lam*eye(length(A));

C = A - B;
x=[1;1;1;1;1];          %dowolny na początek

for i = 1:100
    y = inv(C)*x;
    %y = mojGauss(C, x);         %z13 zestaw 1, daje wektor w odwrotnej
    %kolejności
    x = y/norm(y);
end
%koniec zad12N Mariusz Adamczyk
%-------------------------------------------------------
```

Dane wejściowe:

A =

2	-1	0	0	1
-1	2	1	0	0
0	1	1	1	0
0	0	1	2	-1
1	0	0	-1	2

lam =

0.3820

Uzyskane wyniki:

x =

-0.6015

-0.3717

-0.0000

0.3717

0.6015