WROCLAW UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF ELECTRONICS

FIELD: Electronics
SPECIALITY: Advanced Applied Electronics

Numerical Methods: Eigenproblems

AUTHOR:
Jaroslaw M. Szumega

SUPERVISOR:
Rafal Zdunek, D.Sc, K-4/W4

GRADE:

WROCŁAW 20.03.2017

Chapter 1

Solution to the given problems

(Problems 1, 3, 4 and 7 are solved analytically, without using any of selected algorithms. Result are checked with built-in Octave/Matlab function.)

Problem 1 - Compute the eigenpairs of the matrices. Verify that trace equals to eigenvalues sum and the determinant to their product. Which matrix is singular?

To find eigenvalues, the following calculations will be used:
$Ax = \lambda x$
$Ax - \lambda x = 0$
$(A - \lambda I)x = 0$
$det(A - \lambda I) = 0$

Then the characteristic polynomial can be determined. It's roots are the eigenvalues.

$$A_1 = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

$$det \begin{bmatrix} 1-\lambda & 0 & 0 \\ 2 & 1-\lambda & 0 \\ 0 & 0 & 3-\lambda \end{bmatrix} = (1-\lambda)(1-\lambda)(3-\lambda)$$

$For \lambda = 1:$

$$\begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0 \;=>\; \begin{matrix} 2x_1 = 0 \\ 2x_3 = 0 \\ (no\ x_2\ formula) \end{matrix} \;=>\; x = \begin{bmatrix} 0 \\ t \\ 0 \end{bmatrix}$$

$For \lambda = 3:$

$$\begin{bmatrix} -2 & 0 & 0 \\ 2 & -2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0 \;=>\; \begin{matrix} -2x_1 = 0 \\ 2x_1 - 2x_2 = 0 \\ (no\ x_3\ formula) \end{matrix} \;=>\; x = \begin{bmatrix} 0 \\ 0 \\ t \end{bmatrix}$$

$tr(A) = 1 + 1 + 3 = 5$
$\sum \lambda = 1 + 1 + 3 = 5$

$det(A) = 1 \cdot 1 \cdot 3 = 3$
$\prod \lambda = 1 \cdot 1 \cdot 3 = 3$
Matrix determinant is non–zero, so the matrix is not singular.

$$A_2 = \begin{bmatrix} 0 & -2 & 1 \\ 1 & 3 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$det \begin{bmatrix} 0-\lambda & -2 & 1 \\ 1 & 3-\lambda & -1 \\ 0 & 0 & 1-\lambda \end{bmatrix} = (Sarrus\ theorem =>)(1-\lambda)(1-\lambda)(3-\lambda) =$$

$$= (-\lambda)(3-\lambda)(1-\lambda) - (-2)(1-\lambda) = (1-\lambda)(\lambda^2 - 3\lambda + 2)$$

$For \lambda = 1:$

$$\begin{bmatrix} -1 & -2 & 1 \\ 1 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0 \ => \begin{matrix} -x_1 - 2x_2 + x_3 = 0 \\ x_1 + 2x_2 - x_3 = 0 \end{matrix} => x = \begin{bmatrix} -2v_2 + v_3 \\ v_2 \\ v_3 \end{bmatrix}$$

$For \lambda = 2:$

$$\begin{bmatrix} -2 & -2 & 1 \\ 1 & 1 & -1 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0 \ => \begin{matrix} -2x_1 - 2x_2 + x_3 = 0 \\ x_1 + x_2 - x_3 = 0 \\ (-x_3 = 0) \end{matrix} => x = \begin{bmatrix} v \\ -v \\ 0 \end{bmatrix}$$

$tr(A) = 3 + 1 = 4$

$\sum \lambda = 1 + 1 + 2 = 4$

$det(A) = 2$

$\prod \lambda = 1 \cdot 1 \cdot 2 = 2$

Matrix determinant is non–zero, so the matrix is not singular.

Chapter 2

Listings of solutions and algorithms

2.1 Octave files with problems solutions

Problem no. 1

```
A = [2 -1 0 0; -1 2 -1 0; 0 -1 2 -1; 0 0 -1 2];
b = [0; 0; 0; 5];
C = [A, b];

# basic operations to achieve upper triangular form
C
disp(['R1 <-> R2']);
C = exchange(C, 1, 2);

disp(['R2 = R2 + 2R1']);
C(2,:) = C(2,:) + 2*C(1,:);

disp(['R2 <-> R3']);
C = exchange(C,2,3);
disp(['R3 = R3 + 3R2']);
C(3,:) = C(3,:) + 3*C(2,:);

disp(['R3 <-> R4']);
C = exchange(C,3,4);
disp(['R4 = R4 + 4R3']);
C(4,:) = C(4,:) + 4*C(3,:);


#hand-made backsubstitution
x4 = z = C(4,5)/C(4,4);
x3 = w = (C(3,5) - C(3,4)*x4)/C(3,3);
x2 = v = (C(2,5) - C(2,3)*x3 - C(2,4)*x4)/C(2,2);
x1 = u = (C(1,5) - C(1,2)*x2 - C(1,3)*x3 - C(1,4)*x4)/C(1,1);

#and check with written gaussian elimination and backsubstitution
disp(['Check result']);
D = gaussian_elim([A,b]);
x = backsub(D)
```

Problem no. 2

```
disp(['Equations in of matrix form'])
A = [1 1 1; 1 1 2; 1 2 2]
b = [1;2;1]

disp(['Conacatenation of A and B'])
C = [A, b]

# piwot at 1,1
disp(['R2 = R2 - R1'])
C(2,:) = C(2,:) - C(1,:)
disp(['R3 = R3 - R1'])
C(3,:) = C(3,:) - C(1,:)

#element at 2,2 is zero, row interchange
disp(['R3 <-> R2'])
C = exchange(C,2,3)

x3 = C(3,4)/C(3,3)
x2 = (C(2,4) - C(2,3)*x3)/C(2,2)
x1 = (C(1,4) - C(1,2)*x2 - C(1,3)*x3)/C(1,1)
```

Problem no. 3

```
A = [0.0001 1; 1 1 ]
b = [1;2]
P = [A, b]
NP = [A, b];
WP = [A, b];

disp(['Calculation without pivoting'])
disp([' '])
disp(['R2 - 1e-4*R1'])
NP(2,:) = NP(2,:) - 1e4*NP(1,:)

x2 = round(NP(2,3)/NP(2,2))
x1 = (NP(1,3) - NP(1,2))/NP(1,1)

disp([' '])
disp([' '])
disp(['And with pivoting'])
disp([' '])
disp(['R1<->R2'])
WP = exchange(WP,1,2)
disp(['R2 - 1e-4*R1'])
WP(2,:) = WP(2,:) - 1e-4*WP(1,:)


x2 = round(WP(2,3)/WP(2,2))
x1 = (WP(1,3) - WP(1,2))/WP(1,1)
```

Problem no. 4

```
A = [0.835 0.667; 0.333 0.266]
b = [0.168; 0.067]

disp(['System matrix:'])
C = [A b]

C(2,:) = C(2,:) - C(1,:) * (C(2,1)./C(1,1));
Cx2 = C(2,3)/C(2,2)
Cx1 = (C(1,3) - C(1,2)* Cx2)/C(1,1)

Ap = [0.835 0.667; 0.333 0.266];
b = [0.168; 0.066];

disp(['Small perturbation'])
D = [A b]

D(2,:) = D(2,:) - D(1,:) * (D(2,1)./D(1,1));
Dx2 = D(2,3)/D(2,2)
Dx1 = (D(1,3) - D(1,2)* Dx2)/D(1,1)

disp([' '])
disp(['Condition number of matrix'])
cond(C)
```

Problem no. 5

```
A = [2 1 2; 1 2 3; 4 1 2]
I = [1 0 0; 0 1 0; 0 0 1]
disp(['System matrix:'])
C = [A I]

disp(['R2 - (a21)/(a11) R1'])
C(2,:) = C(2,:) - (C(2,1)/C(1,1)) * C(1,:)

disp(['R3 - (a31)/(a11) R1'])
C(3,:) = C(3,:) - (C(3,1)/C(1,1)) * C(1,:)

disp(['R1 - (a12)/(a22) R2'])
C(1,:) = C(1,:) - (C(1,2)/C(2,2)) * C(2,:)

disp(['R3 - (a32)/(a22) R2'])
C(3,:) = C(3,:) - (C(3,2)/C(2,2)) * C(2,:)

disp(['R1 - (a13)/(a33) R3'])
C(1,:) = C(1,:) - (C(1,3)/C(3,3)) * C(3,:)

disp(['R2 - (a23)/(a33) R3'])
C(2,:) = C(2,:) - (C(2,3)/C(3,3)) * C(3,:)

disp(['R1 / a11'])
C(1,:) = C(1,:)/C(1,1)
disp(['R2 / a22'])
C(2,:) = C(2,:)/C(2,2)
disp(['R3 / a33'])
C(3,:) = C(3,:)/C(3,3)
invC = C(:,4:6)
check = inv(A)
```

Problem no.7

```
disp(['Init of Hilbert Matrix'])
H = hilb(5)

disp(['Programmed algorithm LU'])
[L, U] = lu_fac(H)

[n,n] = size(H);

disp(['Own determinant calculus'])
detH = 1;
for i = 1:n
detH = detH * U(i,i);
end
detH
disp(['Check with embedded function'])
det(H)

#+timing measurement
```

Problem no.8

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


#is positive definite?
positivedefinite = all(eig(A) > 0)
#in this case G is upper triangular matrix
disp(['Cholesky factorization'])
G = cholesky_fac(A)

disp(['Inversion using coded Cholesky factorization'])
inv(G)*inv(G)'
disp(['Embedded inversion'])
inv(A)

#+timing measurement
```

Problem no.9

```
A = pascal(100);
cholesky_fac(A);

B = pascal(5)
cholesky_fac(B)

C = pascal(10)
cholesky_fac(C)
```

Problem no.10

```
A = [1 2 2 3 1; 2 4 4 6 2; 3 6 6 9 6; 1 2 4 5 3;]
RREF = alg_gjrref(A)
```

Problem no.11

```
A = [1 3 3 2; 2 6 9 5; -1 -3 3 0]
B = lu_fac_pivot(A)
```

Problem no.12

```
A = [0 -1 -3; 0 0 -2; 0 -2 1]

[Q, R] = QRgivens_lecture(A)

disp(['check by Q*R'])
Q*R



for i = 1:1000
[Q, R] = QRgivens_lecture(A);
end
toc
disp([' '])
disp([' '])
disp(['Embedded algorithm'])
tic
for i = 1:1000
[Q, R] = qr(A);
end
toc
```

Problem no.15

```
A = [1 0 -1 -1 0 0 0;
0 1 1 0 -1 0 0;
-1 0 0 0 1 1 0;
0 0 -10 10 0   0  10;
0 0 0  10  0   10 20;
0 0  0  0  10 -10 10]

disp(['Using gauss-jordan elimination'])
alg_gjrref(A)
```

2.2 Coded selected algorithms

Algorithm 1 - Gaussian elimination

```
function [A] = gaussian_elim(A)

[n,m] = size(A);

for k = 1:n-1
#discussed on the lectre -> to avoid second loop, the 'rows' are used
rows = k+1:n;
A(rows, k) = A(rows,k)/A(k,k);
A(rows,rows) = A(rows,rows) - A(rows,k)*A(k,rows);
end
```

Algorithm 3 - Forward substitution

```
function [b] = forwardsub(C)

[n,m] = size(C);

L = C(:,1:m-1);
b = C(:,m);

b(1) = b(1)/U(1,1);

for i = 2:n
b(i) = (b(i) - L(i, 1:i-1)*b(1:i-1))/L(i,i);

end
end
```

Algorithm 4 - Back substitution

```
function [b] = backsub(C)

[n,m] = size(C);

U = C(:,1:m-1);
b = C(:,m);

b(n) = b(n)/U(n,n);

for i = n-1:-1:1
b(i) = (b(i) - U(i, i+1:n)*b(i+1:n))/U(i,i);

end
end
```

Algorithm 5, 6 – Gauss–Jordan, used also as RREF

```
function [A] = alg_gjrref(A)

[n,m] = size(A);

j = 1;
for i = 1:n
#a
while(A(i:n,j) == 0)
j = j+1;

if(j>m)
return
endif
endwhile

#b
x = i;
while(A(i,j)==0)
x = x + 1;
if(x>n)
break;
endif
if(A(x,j)!=0)
temp = A(i,:);
A(i,:) = A(x, :);
A(x, :) =  temp;
break;
endif
endwhile

if(A(i,j)==0)
continue;
endif

#c
A(i,:) = A(i,:)/A(i,j);

#d
for k = 1:n
if( k == i)
continue;
endif
A(k,:) = A(k,:) - A(i,:)*A(k,j);
end

end
end
```

Algorithm 7 LU factorization

```
function [L, U] = lu_fac(A)

[n, m] = size(A);

L = eye(n);
U = zeros(n,m);

for j = 1:n
if (j == 1)
v(j:n) = A(j:n,j);

else
#the elimination below won't work
#z = (A(1:j-1,j)) / (L(1:j-1, 1:j-1));

z = inv((L(1:j-1, 1:j-1))) * (A(1:j-1,j));
U(1:j-1, j) = z;

v(j:n) = A(j:n, j) - L(j:n, 1:j-1)*z;
endif

if(j<n)
L(j+1:n, j) = v(j+1:n) / v(j);
end
U(j,j) = v(j);

end
```

Algorithm 8 – LU factorization with pivoting

```
function [L, U, p] = lu_fac_pivot(A)

[n, m] = size(A);

L = eye(n);
U = zeros(n,m);
p = zeros(n,n)

for j = 1:n
if (j == 1)
v(j:n) = A(j:n,j);

else
#the elimination below won't work??
#z = (A(1:j-1,j)) / (L(1:j-1, 1:j-1));

z = inv((L(1:j-1, 1:j-1))) * (A(1:j-1,j));
U(1:j-1, j) = z;

v(j:n) = A(j:n, j) - L(j:n, 1:j-1)*z;
endif

if(j<n)
[val, index] = max(v(j:n));
p(j) = index;
tempV = v(j);
v(j) = v(index);
v(index) = tempV;

tempA =  A(j,j+1:n)
A(j,j+1:n) = A(index, j+1:n);
A(index, j+1:n) = tempA;

L(j+1:n, j) = v(j+1:n) / v(j);

if(j>1)
tempL = L(j,1:j-1);
L(j,1:j-1) = L(index, 1:j-1);
L(index, 1:j-1) = tempL;
endif
endif
U(j,j) = v(j);

end
```

Algorithm 10 – Cholesky factorization

```
function [G] = cholesky_fac(A)

G = A;
[n,k] = size(G);

for j = 1:n
if (j>1)
G(j:n,j) = G(j:n,j) - G(j:n,1:j-1)*G(j,1:j-1)';
end
G(j:n,j) = G(j:n,j)/sqrt(G(j,j));
end

#at the end -> eliminate what has left from A matrix when i>j
for i = 1:n
for j = 1:k

if(i>j)
G(i,j) = 0;

end
end


end
```

Algorithm 12 – QR by Givens rotation

```
function [ Q,R ] = QRgivens_lecture( A )

[n, n] =size(A);
Q=eye(n);
R=A;
for j=1:n
for i=n:(-1):j+1
x=R(:,j);
if norm([x(i-1),x(i)])>0

# calculate givens c and s
c=x(i-1)/norm([x(i-1),x(i)]);
s=-x(i)/norm([x(i-1),x(i)]);

G=eye(n);

#update
G([i-1,i],[i-1,i])=[c,s;-s,c];

R=G'*R;
Q=Q*G;
end

end
end
```

Bibliography

[1] Björck, Åke. Numerical methods for least squares problems. Society for Industrial and Applied Mathematics, 1996.

[2] Golub, Gene H., and Charles F. Van Loan. "Matrix computations, 3rd." (1996).

[3] Transforming a matrix to reduced row echelon form, http://www.di-mgt.com.au/matrixtransform.html

[4] Zdunek R., Numerical Methods - lecture slides.