System of Linear Equations

Systems of linear equations are very important in applied and numerical mathematics. Since engineers are dealing mostly with linear models, this requires solving systems of linear equations in engineering problems.

Consider a problem in statics. Let us analyze the forces on a truss with equilateral triangles as shown on the figure below.

Compute member forces by the method of joints. Make the following approximations:

\[
\cos(60^\circ) = 0.5 \quad \text{and} \quad \sin(60^\circ) = \frac{\sqrt{3}}{2} \approx 0.8660 \quad (f_1 = 1000 \quad \text{N} \quad \text{and} \quad f_2 = 5000 \quad \text{N}).
\]

Equations of equilibrium of rectangular components of forces and moments are used to determine reactions at supports \((R_1, R_2, R_3)\). The following system of linear equations can be formulated by considering forces at nodes, where \(T_i\) represent tension in members:

\[
\begin{align*}
0.5 T_1 + T_2 &= R_1 = f_1 \\
0.866 T_1 &= -R_2 = 0.433 f_1 - 0.5 f_2 \\
-0.5 T_1 + 0.5 T_3 + T_4 &= -f_1 \\
0.866 T_1 + 0.866 T_3 &= 0 \\
-T_2 - 0.5 T_3 + 0.5 T_5 + T_6 &= 0 \\
0.866 T_3 + 0.866 T_5 &= f_2 \\
-T_4 - 0.5 T_5 + 0.5 T_7 &= 0
\end{align*}
\]

The above linear system is equivalent to the matrix equation \(A x = b\).

In general form:
Where:

\[ A = \begin{pmatrix}
0.5 & 1 & 0 & 0 & 0 & 0 & 0 \\
0.866 & 0 & 0 & 0 & 0 & 0 & 0 \\
-0.5 & 0 & 0.5 & 1 & 0 & 0 & 0 \\
0 & -1 & -0.5 & 0 & 0.5 & 1 & 0 \\
0 & 0 & 0.866 & 0 & 0.866 & 0 & 0 \\
0 & 0 & 0 & -1 & -0.5 & 0 & 0.5 \\
\end{pmatrix}, \quad 
\begin{pmatrix}
f_1 \\
0.433f_1 - 0.5f_2 \\
-f_1 \\
0 \\
f_2 \\
0 \\
\end{pmatrix} \]

Existence and uniqueness of solution

The system of equations can be inhomogeneous \((b \neq 0)\) or homogeneous \((b = 0)\). When the system is homogeneous, we have a nontrivial solution \((x \neq 0)\) only if the determinant of \(A\) is zero: \(\text{det}(A) = 0\).

Discussion of solutions when the system is inhomogeneous:

**Inhomogeneous systems of linear equations, \(A\times x = b\)**

- **I. Solution exists** \(r(A) = r(A|b)\)
  - 1) Unique solution, \(r(A) = n, \text{det}(A) \neq 0\)
  - 2) Infinite number of solutions, \(r(A) < n\)

- **II. No solution** \(r(A) < r(A|b)\)
  - 1) \(m = n\) (overdetermined)
  - 2) \(m > n\) (underdetermined)

Solution exists and is unique

**Solution exists**, if rank of matrix \(A\) (number of its linearly independent columns, \(r(A)\)) is the same as the rank of the augmented matrix \(A|b\), i.e. \(r(A) = r(A|b)\). The augmented matrix is obtained by appending column vector \(b\) to matrix \(A\).
The solution is unique, if \( r(A) = r([A \mid b]) = n \) or \( \det(A) \neq 0 \), that is rank of matrix \( A \) is equal to the rank of the augmented matrix and this rank is full (equals to the number of columns).

Load \( A \) and \( b \) from file `truss.txt`.

\[
\begin{align*}
A &= \text{load('truss.txt')}
\end{align*}
\]

Check whether we have a unique solution

\[
\text{rank}(A), \text{rank}([A \mid b]) \text{ } 7=7, \text{ solution exists and unique: } n=7
\]

Check whether the determinant is non-zero:

\[
\text{det}(A) \text{ } \% \text{ } -0.3247 \text{ } - \text{ we have a unique solution}
\]

**Solution methods in matlab if there is a unique solution**

Both direct and iterative methods may be used for solving systems of linear equations. Let us consider now direct methods. Consider four possibilities:

\[
\begin{align*}
x1 &= \text{inv}(A) \times b; \text{ } \% \text{ computation of inverse (time consuming)} \\
x2 &= A \backslash b; \text{ } \% \text{ nxn: Cholesky or LU factorization; mxn: QR factorization} \\
x2 &= \text{mldivide}(A,b); \text{ } \% \text{ the same as above} \\
x3 &= \text{pinv}(A) \times b; \text{ } \% \text{ pseudoinverse by SVD factorization} \\
x4 &= \text{linsolve}(A,b); \text{ } \% \text{ LU or QR factorization can be specified} \\
x1 \text{ } x2 \text{ } x3 \text{ } x4 \text{ } \% \text{ results are shown side by side}
\end{align*}
\]

Check our solutions by computing the length (\text{norm} - \text{function \textbf{norm}}) of the residual vector if it is zero or is within the required tolerance.

\[
\begin{align*}
\text{norm}(A \times x1-b) \text{ } \% \text{ } 1.0904e-12 \\
\text{norm}(A \times x2-b) \text{ } \% \text{ } 5.5695e-13 \\
\text{norm}(A \times x3-b) \text{ } \% \text{ } 5.1348e-12 \\
\text{norm}(A \times x4-b) \text{ } \% \text{ } 6.0157e-13
\end{align*}
\]

The most precise method is \( x2 = A \backslash b \).

**Matrix decompositions**

**Solution of linear equations by LU decomposition**

LU decomposition for \( A \) consists of three matrices \((L, U, P=\text{permutation matrix})\) such that
\[ P \cdot A = L \cdot U. \]

Matlab will produce an LU decomposition with pivoting for a matrix \( A \) with the following command:

\[
[L \ U \ P] = lu(A)
\]

To solve \( Ax = b \) we first multiply both sides with the permutation (pivot) matrix:

\[ P \cdot A \cdot x = P \cdot b = d. \]

Substituting \( L \cdot U \) for \( P \cdot A \) we get

\[ L \cdot U \cdot x = d \]

Then we define \( y = U \cdot x \), which is unknown since \( x \) is unknown. Using forward substitution, we can (easily) solve

- \( L \cdot y = d \) (for \( y \), where \( L \) is lower triangular and \( d = P \cdot b \))

and then using back substitution we can (easily) solve

- \( U \cdot x = y \) (for \( x \), where \( U \) is upper triangular)

Solve the equation system of the truss problem by LU decomposition

\[
[L \ U \ P] = lu(A)
\]

\[
d = P*b;
\]

\[
\text{opt1.LT}=true
\]

\[
y = \text{linsolve}(L,d,\text{opt1});
\]

\[
\text{opt2.UT}=true
\]

\[
x = \text{linsolve}(U,y,\text{opt2})
\]

\[
\text{residual} = \text{norm}(A*x - b) \% \text{check}
\]

**Cholesky decomposition**

Cholesky decomposition is very similar to LU decomposition: \( A = L^T \cdot L \).

Steps of the solution using \( A \cdot x = L^T \cdot L \cdot x = L^T \cdot y = b \):

- \( L^T \cdot y = b \) (where \( L^T \) is lower triangular)
- \( L \cdot x = y \) (where \( L \) is upper triangular)

Conditions:

- Matrix \( A \) is symmetric, i.e. \( A^T = A \) and
- positive definite, i.e. it has got only positive eigenvalues

Matrix \( A \) of the truss problem is symmetric and positive definite:

\[
\text{norm}(A-A') \% 1.5946
\]

\[
\text{real(eig(A))} \% [0.5000; -0.7136; -0.7136; -0.7136; 1.2136; 1.2136; 1.2136]
\]

\[
\text{%chol(A)} \% \text{issues an error message: Matrix must be positive definite}
\]
Instead consider the following symmetric and positive definite matrix:

\[ A = \text{pascal}(4) \]  
\[ b = \text{rand}(4,1) \]  
\[ \text{norm}(A-A') \] (symmetric because \( A = A' \))  
\[ [V D] = \text{eig}(A) \] (compute eigenvectors and eigenvalues)  
\[ \text{min}(	ext{diag}(D)) \] (positive definite since even the smallest eigenvalue >0)  
\[ L = \text{chol}(A) \] (Cholesky decomposition is allowed)  
\[ \text{norm}(A-L'\cdot L) \] (check)

Solution of a linear system is similar to LU decomposition (there is no permutation, hence instead of \( d \) we put vector \( b \), put \( L' \) in place of \( L \) and put \( L \) in place of \( U \)):

\[
\begin{align*}
\text{opt1.LT} &= \text{true} \\
\text{y} &= \text{linsolve}(L', b, \text{opt1}); \\
\text{opt2.UT} &= \text{true} \\
\text{x} &= \text{linsolve}(L, y, \text{opt2}) \\
\text{residual} &= \text{norm}(A\cdot x - b) \quad \text{(check)}
\end{align*}
\]

Infinite number of solutions

This is possible when the ranks of the matrix and that of the augmented matrix are equal but it is less than the number of columns, i.e. \( r(A)=r(A\mid b) \) ès \( r(A)<n \). This may happen for a rank deficient square matrix \((m = n)\) or for an undetermined system \((m < n)\) with less equations than unknowns. One is selected from the infinite possible solutions, generally the one having smallest norm using the formula

\[ x = A^\top \cdot (A \cdot A^\top)^{-1} \cdot b. \]

Two Matlab built-in functions can be used for the solution: \texttt{pinv} and \texttt{\backslash} (or \texttt{mldivide}).

Consider the following system of equations:

\[
\begin{align*}
7 \cdot x_1 + 2 \cdot x_2 + 2 \cdot x_4 &= 1 \\
x_1 + 8 \cdot x_2 + x_3 + 8 \cdot x_4 &= 2
\end{align*}
\]

Check the number of solutions

\[
\begin{align*}
A &= \begin{bmatrix} 7 & 2 & 0 & 2; & 1 & 8 & 1 & 8 \end{bmatrix}, \quad b = \begin{bmatrix} 1; & 2 \end{bmatrix} \\
\text{rank}(A) &= \text{rank}([A \ b]), \quad \text{rank}(A)
\end{align*}
\]

Rank of matrix \( A \) is the same as rank of the augmented matrix, hence there is a solution, even an infinite number of solutions exist since \( r(A)=2 < n=4 \).

Solve the system in three different ways and check:

\[
\begin{align*}
xa &= A' \cdot \text{inv}(A \cdot (A')) \cdot b \\
xb &= \text{pinv}(A) \cdot b \\
xc &= A\backslash b \\
\text{norm}(xa), \text{norm}(xb), \text{norm}(xc)
\end{align*}
\]

\% Results are: \( 0.1852, \ 0.1852, \ 0.2519 \)
QR decomposition

Matlab's built-in \( \text{mldivide} \) and \text{lin.solve} \) commands apply QR decomposition for non-square matrices. QR method is based on the fact that every matrix \( A \) can be decomposed into the product of matrices \( Q \) and \( R \) (\( A = QR \)), where \( Q \) is orthonormal and \( R \) is upper triangular.

Steps of the solution of the system \( A \cdot x = b \) is:

- compute \( B = Q^T \cdot b \)
- solve \( R \cdot x = B \)

Let us make a solution by QR decomposition and check it.

```matlab
[Q R] = qr(A)
norm(A-Q*R) % Check decomposition
norm(Q'*Q) % Is Q orthonormal? Yes, Q'*Q is an identity matrix
B=Q'*b % Right side
% Solution
opt.UT=true;
x=lin.solve(R,B,opt)
norm(A*x-b)
```

SVD (Singular Value Decomposition)

Every \((m \times n)\) matrix \( A \) has the following decompositon:

\[
A = U \cdot S \cdot V^T
\]

where matrix \( U \) is \( m \times m \) and its column vectors are eigenvectors of \( A \cdot A^T \), matrix \( V \) is \( n \times n \) and its columns are eigenvectors of \( A^T \cdot A \). Matrix \( S \) is \( m \times n \) containing along its main diagonal positive square roots of eigenvalues of \( A \cdot A^T \), the so called singular values. All of its other elements are zero. Matrices \( U \) and \( V \) are orthonormal, i.e. their inverses are equal to their transposes, \( U^{-1} = U^T \), \( V^{-1} = V^T \).

Due to the fact that \( U \) and \( V \) are orthonormal and \( S \) is diagonal it is easy to obtain inverse/pseudoinverse of \( A \):

\[
A^{-1} = V \cdot S^{-1} \cdot U
\]

Let us solve the adjustment problem of leveling network by SVD decomposition.

```matlab
[U,S,V] = svd(A)
% Checks
norm(A-U*S*V') % check decomposition
norm(U'*U-eye(size(A,1))) % U is mxm orthonormal, U'=inv(U)
norm(V'*V-eye(size(A,2))) % V is nxn orthonormal, V'=inv(V)
diag(S), sqrt(eig(A'*A)) % singular values are roots of eigenvalues of A'A
% Compute pseudoinverse
invS=(1./S)' % inverse of S
invS(invS==Inf)=0 % put 0 where there is an Inf (infinity) element
invS*S % check
invA=V*invS*U' % pseudoinverse of A
```
Function \texttt{pinv}(A) uses SVD decomposition as well.

**No solution (solution with minimum error)**

When rank of the matrix A is smaller than rank of the augmented matrix A|b, i.e. r(A)<r(A|b) we do not have a solution. Vector b does not belong to the space of column vectors of A. Either we have a square matrix m=n or an undetermined system m>n, i.e. we have more equations than unknowns.

Although we have no exact solution, the solution with minimum residuals will be the following:

\[ x = (A^T \cdot A)^{-1} \cdot A^T \cdot b \]

The same two Matlab built-in functions may be used as before: \texttt{pinv} and \texttt{\backslash} (or \texttt{mldivide}).

Let us now solve a problem in surveying. Heights of benchmarks are to be determined in a leveling network with redundancy. Consider adjustment of the leveling network on figure below.

Height differences of lines 1–7 are measured, heights of 3 benchmarks (A,B,C) above mean sea level are known and we have to solve for heights of 3 additional benchmarks (D,E,F). Lengths of leveling lines are nearly the same, hence we may assign the same weights to the measurements.

Heights of benchmarks: \(HA=183.506\)m, \(HB=192.353\)m, \(HC=191.880\)m.

Measurements:
\[
H_D - H_A = +6.135 \\
H_E - H_D = +8.343 \\
H_E - H_B = +5.614 \\
H_F - H_D = +1.394 \\
H_F - H_E = -6.969 \\
H_F - H_C = -0.930 \\
H_E - H_C = +6.078
\]

Matrix formulation of the system of equations:

\[
\begin{bmatrix}
1 & 0 & 0 \\
-1 & 1 & 0 \\
0 & 1 & 0 \\
-1 & 0 & 1 \\
0 & -1 & 1 \\
0 & 0 & 1 \\
0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
H_D \\
H_E \\
H_F
\end{bmatrix}
=
\begin{bmatrix}
6.135+183.506 \\
8.343 \\
5.614+192.353 \\
1.394 \\
-6.969 \\
-0.930+191.880 \\
6.078+191.880
\end{bmatrix}
=
\begin{bmatrix}
189.641 \\
8.343 \\
197.967 \\
1.394 \\
-6.969 \\
190.950 \\
197.958
\end{bmatrix}
\]

There are 7 equations and 3 unknowns (m>n), hence we have an overdetermined system. Solve it in Matlab. Either we may type in our matrices or load from the file leveling.txt.

\[
A = [1 \ 0 \ 0; -1 \ 1 \ 0; 0 \ 1 \ 0; -1 \ 0 \ 1; 0 \ -1 \ 1; 0 \ 0 \ 1; 0 \ 1 \ 0] \\
b = [189.641; 8.343; 197.967; 1.394; -6.969; 190.950; 197.958] \\
\% Ab = load('leveling.txt') \\
\% A = Ab(:,1:3) \\
\% b = Ab(:,4)
\]

Check if there is any solution

\[
\text{rank}(A) \quad \% 3 \\
\text{rank}([A \ b]) \quad \% 4
\]

We found r(A)<r(A|b), hence there is no exact solution to this system, only a minimum norm approximation. Let us find this solution by using Matlab's built-in functions.

\[
x1 = \text{inv}(A'*A)*A'*b \quad \% 189.6153, 197.9588, 190.9830 \\
x2 = A\backslash b \quad \% 189.6153, 197.9588, 190.9830 \\
x3 = \text{linsolve}(A,b) \quad \% 189.6153, 197.9588, 190.9830 \\
x4 = \text{pinv}(A)*b \quad \% 189.6153, 197.9588, 190.9830
\]

Check:

\[
\text{norm}(A*x1-b) \quad \% 0.0506
\]

**Iterative methods (Jacobi, Gauss-Seidel)**

Determine the concentration of the water plant based on this figure. Consider perfect amalgamation (i.e., the concentration is the same at each node).
The equations:

\[ 6c_1 - c_3 = 50 \]
\[ -3c_1 + 3c_2 = 0 \]
\[ -c_2 + 9c_3 = 160 \]
\[ -c_2 - 8c_3 + 11c_4 - 2c_5 = 0 \]
\[ -3c_1 - c_2 + 4c_5 = 0 \]

The matrix form of the problem:

\[
A = \begin{pmatrix}
6 & 0 & -1 & 0 & 0 \\
-3 & 3 & 0 & 0 & 0 \\
0 & -1 & 9 & 0 & 0 \\
0 & -1 & -8 & 11 & -2 \\
-3 & -1 & 0 & 0 & 4
\end{pmatrix}, \quad b = \begin{pmatrix}
50 \\
0 \\
160 \\
0 \\
0
\end{pmatrix}
\]

Check whether there is a solution and it is unique. Load A and b from file *waterplant.txt*:

```matlab
Ab = load('waterplant.txt');
% Ab=[6,0,-1,0,0,50;-3,3,0,0,0,0;-1,9,0,0,160;0,-1,-8,11,-2;0,-3,1,0,4,0];
A = Ab(:,1:end-1); b = Ab(:,end);
rank(A), rank(Ab) % 5, 5
```

Letting \( AI = (I - B^{-1} \cdot A) \) and \( bi = B^{-1} \cdot b \), we get the following iteration equation

\[ x_{k+1} = AI \cdot x + bi \]

This is a general iteration equation including both Jacobi and Gauss-Seidel methods. The only difference is in matrix \( B \). In Jacobi method matrix \( B \) contains the main diagonal of \( A \). In Gauss-Seidel method matrix \( B \) matrix is the lower triangular part of matrix \( A \).

First see how to implement the iteration formula in Matlab with known matrix \( AI \) and vector \( bi \.

Look at the code in file iterativ.m:
function X=iterativ(Ai,bi,x0,e,imax)
    X=x0; i=0;
    x1= Ai*x0+bi; % first iteration
    while i<=imax & & norm(x1-x0)>e
        x0=x1;
        x1= Ai*x0+bi;
        X=[X x1];
        i=i+1;
    end
end

Let us solve equations of the water plant problem by Jacobi and Gauss-Seidel iteration. Matrices $A$ and vectors $b$ have to be specified for both cases first. Remember that matrix $B$ in Jacobi iteration method is the main diagonal of $A$, whereas in Gauss-Seidel iteration $B$ is the lower diagonal part of $A$. For initial value a vector with components all 1 is given.

First we use Jacobi method:

```matlab
x0=ones(5,1)
% compute Ai and bi for Jacobi method
B = diag(diag(A)) % main diagonal of A
Ai=eye(5)-inv(B)*A
bi = inv(B)*b
% iterative solution
X=iterativ(Ai,bi,x0,1E-6,100)
% result is in the last column
x=X(:,end)
% number of iterations
iter = size(X,2) % 15
% check (relative error)
norm(A*x-b)
% make plot
figure(1);
plot(x,'*-')
title('Convergence of Jacobi method')
```
Now consider Gauss-Seidel method. Mainly a copy of the above, only with a different matrix $B$

```matlab
% compute AI and bi for Gauss-Seidel method
B=tril(A) % lower triangular part of A
Ai=eye(5)-inv(B)*A
bi = inv(B)*b
% iterative solution
X=iterativ(Ai,bi,x0,1E-6,100)
% result is in the last column
x=X(:,end)
% number of iterations
iter=size(X,2) % 7
% check (relative error)
norm(A*x-b)/norm(b)
% make plot
figure(2);
plot(X,'*-*')
title('Convergence of Gauss-Seidel method')
```
Comparing the two solutions we notice that Jacobi method required 15 iterations whereas 7 iterations was enough for the Gauss-Seidel method in this particular case.

Matlab has its own built-in method for iterative solution called `gmres`. For instance:

\[
[x, \text{flags}, \text{relres}, \text{iter}, \text{resvec}] = \text{gmres}(A,b) \\
\text{figure(3); plot(resvec,'b*-')} \% \text{plot iteration residuals}
\]

For effective storage of sparse matrices Matlab has its own format that is useful for storing large sparse matrices: `A5=sparse(A)`.

**System of linear equations continued**

In engineering practice sometimes we face with homogeneous systems and also with inhomogeneous systems having no or infinite number of solutions. It may happen as well that seemingly there is a unique solution for a problem, but on close inspection we find something strange.

**Matrix condition number**

Consider the following problem. We have a unique solution, however, there are doubts whether it is really correct or not. Let us solve the following two linear systems with the same matrix `A` and with vectors `b` that only have a small difference in one of their components.

\[
A = \begin{bmatrix} 1 & 1/2 \\ 1 & 1/3 \end{bmatrix} ; \quad b_1 = \begin{bmatrix} 3/2 \\ 1 \end{bmatrix} ; \quad b_2 = \begin{bmatrix} 3/2 \\ 5/6 \end{bmatrix}
\]

Solutions by Gaussian elimination:
\[
\begin{bmatrix}
1 & 1/2 & 3/2 \\
0 & 1/12 & 1/4
\end{bmatrix}
\quad \text{and} \quad
\begin{bmatrix}
1 & 1/2 & 3/2 \\
0 & 1/12 & 1/12
\end{bmatrix}
\]

Proper solution of the first is \(x_1 = (0; 3)\), and of the second is \(x_2 = (1, 1)\).

With Matlab:

\[
A = \begin{bmatrix} 1 & 1/2; & 1/2 & 1/3 \\ b1 = [3/2; 1] \\ b2 = [3/2; 5/6] \\ x1 = A\b1 \% -0.0000; 3.0000 \\ x2 = A\b2 \% 1.0000; 1.0000
\]

The two inputs are close to each other, but the two solutions are not. Why?

Certain matrices are very sensitive to a small change of their input. Condition number (\(k\)) scales the relative error of output in terms of the relative error of input. The greater this number, the greater the change of output in response to a small change in input.

The condition number of \(A\):

\[
\text{cond}(A) \% \text{result is: 19.2815}
\]

This is of serious concern in engineering calculations since input measurements or constants are mostly approximate quantities.

**Homogeneous system of linear equations**

When right side of the system is zero (\(b=0\)) it is called a homogeneous system of linear equations. In this case there is a nontrivial solution \(x \neq 0\) only when the determinant of matrix \(A\) is zero: \(\det(A)=0\).

Let us consider an example from engineering (mechanics), the Cauchy stress tensor \(F\) (see figure).
Determination of principal stresses and axes of principal stresses is an eigenvalue-eigenvector problem which is the following homogeneous system

\[(F - \sigma_e \cdot I) \cdot e = 0\]

where \(\sigma_e\) is a principal stress, \(I\) is the identity matrix and \(e\) is unit vector along the principal stress axis. We seek a nontrivial solution \(e \neq 0\). Eigenvalues are solutions of the characteristic equation \(\det(F - \sigma_e \cdot I) = 0\).

Consider an example in Matlab. Elements of stress tensor \(F\) at point P of the solid body are: \(\sigma_x = 50\text{ MPa}, \sigma_y = 80\text{ MPa}, \sigma_z = -20\text{ MPa}, \tau_{xy} = \tau_{yx} = 20\text{ MPa}, \tau_{xz} = \tau_{zx} = -40\text{ MPa}, \tau_{yz} = \tau_{zy} = -30\text{ MPa}\).

Let us determine both principal stresses and axes.

We can write characteristic equation symbolically by expanding the determinant

```matlab
syms de
F = [50 20 -40; 20 80 -30; -40 -30 -20]
eq = det(F-eye(3)*de)
% Result:
eq = - de^3 + 110*de^2 + 1500*de - 197000
```

Roots of the equation are:

```matlab
c=sym2poly(eq) % c = c = -1 110 1500 -197000
de = roots(c) % [106.7674 -41.3691 44.6017]
```

Shortcut in Matlab to obtain eigenvalues by the `eig` function:

```matlab
eig(F) % -41.3691 44.6017 106.7674
```

Check whether matrix is really singular:

```matlab
A1 = F-eye(3)*de(1)
% Singular? YES
det(A1)
3-rank(A1) % Number of basis vectors: 1
```

The eigenvector is a basis vector of the null space of matrix A1

```matlab
v1=null(A1)
% v1 =
% -0.5203
% -0.7795
% 0.3487
```

Check our solution by substituting into the original homogeneous system of equations

```matlab
% substitution into the homogeneous system
(F-de(1)*eye(3))*v1
% 1.0e-13 *
% -0.2487
```
Matlab built-in function `eig` with two outputs gives all eigenvalues and eigenvectors as columns of \( V \):

% Eigenvalue and eigenvector computation by the built-in function
[V D]=eig(F)

% V =
% 0.3647  0.7722  0.5203
% 0.1664 -0.6038  0.7795
% 0.9162 -0.1977 -0.3487
% D =
% -41.3691  0   0
% 0  44.6017  0
% 0   0  106.7674

```matlab
function X=iterativ(Ai,bi,x0,e,imax)
X=x0; i=0;
x1=Ai*x0+bi; % first iteration
while i<=imax && norm(x1-x0)>e
    x0=x1;
x1=Ai*x0+bi;
X=[X x1];
i=i+1;
end
end
```