EPFL

Numerical Analysis and Computational Mathematics

Fall Semester 2025 – CSE Section

Prof. Laura Grigori

Assistant: Israa Fakih

Session 11 – November 26, 2025

Solutions – Linear systems & Eigenproblems

Solution I (MATLAB)

a) We assign the matrix A, visualize the truss bridge in Figure 1, and plot the pattern of A in Figure 2(left) by means of the following commands.

```
KK = 1e3;
nnodes = 29; % number of nodes of the bridge (odd)
[ A ] = bridge_stiffness_matrix( nnodes, KK );
plot_bridge(nnodes);
spy(A);
```

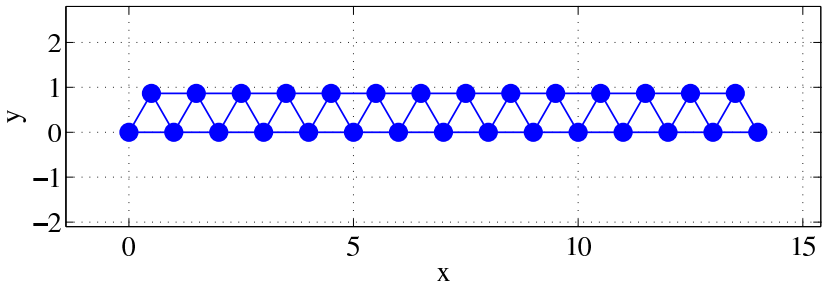

Figure 1: Truss bridge with $N_{nodes} = 29$.

We observe that the matrix A is sparse and banded. Then, we verify that the sparse matrix A is symmetric by plotting the pattern of the matrix $A - A'$ in Figure 2(right); we observe that each element of the matrix $A - A'$ is identically zero and thus A is symmetric.

```
spy(A'-A);
```

We verify that the matrix A is singular. Instead of computing the determinant of the matrix A, we estimate its conditioning number by means of the MATLAB command `condest` which returns the result `Inf`.

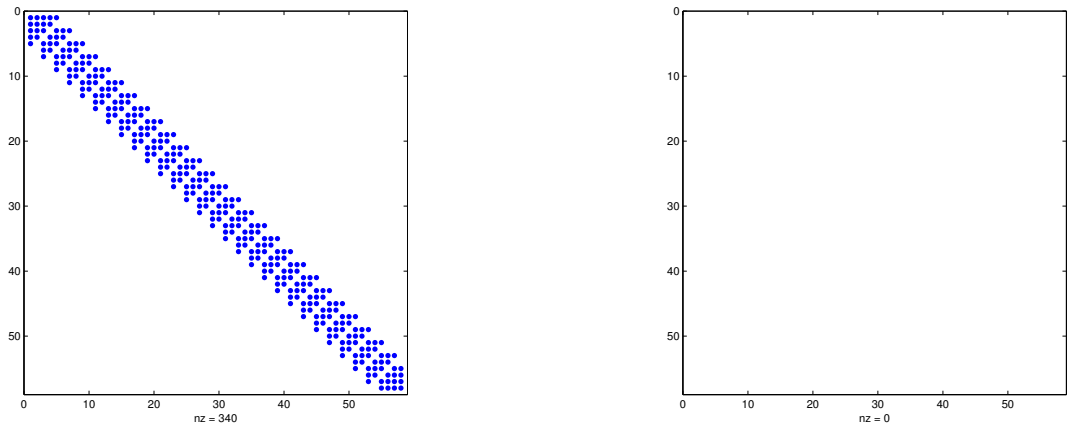

Figure 2: Patterns of the sparse matrices A (left) and $A - A'$ (right); the nonzero elements are indicated by dots.

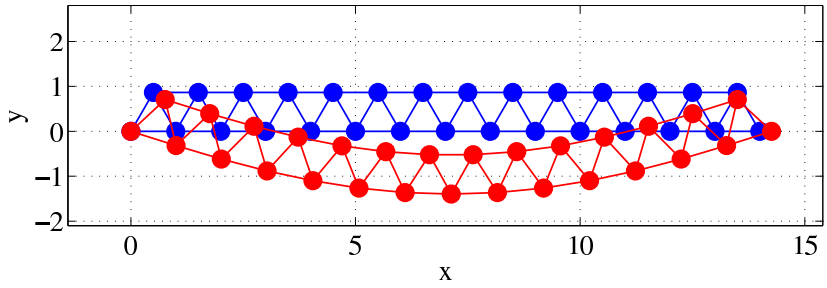

Figure 3: Truss bridge; deformed configuration (red) for the load $\mathbf{f}_i^{ext}$, $i = 1, 3, \ldots, \ldots, N_{nodes}$.

```
K_A = condest( A )
%    K_A =
%       Inf
```

b) We impose the constraints to obtain the matrix $\widetilde{A}$ from A and we verify that $\widetilde{A}$ is nonsingular by means of the following commands. `Ar` corresponds to the matrix $\widetilde{A}$.

```
n = size( A, 2 );
nr = n - 3;
ir = [ 3 : n - 1 ];
Ar = A( ir, ir );
K_Ar = condest( Ar )
%    K_Ar =
%       1.8838e+04
```

c) We use the following MATLAB commands to obtain the result of Figure 3 representing the deformed configuration of the bridge; this is obtained by applying the displacement vector $\mathbf{d}$ to the nodes of the undeformed truss bridge of Figure 1.

```
node_force = 1 : 2 : nnodes;
b = zeros( n, 1 );
b( 2 * node_force, 1 ) = -1;
br = b( ir, 1 );
dr = Ar \ br;
d = zeros( n, 1 );
d( ir, 1 ) = dr;
[ fig_h ] = plot_bridge( nnodes, d );
```

`br` and `dr` correspond to the vectors $\widetilde{\mathbf{b}}$ and $\widetilde{\mathbf{d}}$, respectively.

d) Since the LU factorization of the matrix $\widetilde{A}$ is independent of the loads (the vector $\widetilde{\mathbf{b}}$), we use the following MATLAB commands, with the factorization of the matrix $\widetilde{A}$ performed only once.

```
[ Lr, Ur, Pr ] = lu( Ar );
for l = 1 : 10
    b = 2 * ( rand( n, 1 ) - 0.5 );  % random force in [-1,1]
    br = b( ir, 1 );
    [ yr ] = forward_substitutions( Lr, Pr * br );
    [ dr ] = backward_substitutions( Ur, yr );
    d = zeros( n, 1 );
    d( ir, 1 ) = dr;
    [ fig_h ] = plot_bridge( nnodes, d );
    pause( 1 );
end
```

e) We use the following MATLAB commands to obtain that the GMRES and conjugate gradient methods converge to the solution, for the prescribed tolerance, in 54 and 56 iterations, respectively.

```
node_force = 1 : 2 : nnodes;
b = zeros( n, 1 );
b( 2 * node_force, 1 ) = -1;
br = b( ir, 1 );
dr_gmres = gmres( Ar, br, [], 5e-5, nr );
%   gmres converged at iteration 54 to a solution with relative residual 1.8e-05.
dr_cg = pcg( Ar, br, 5e-5, 1000, [] );
%   pcg converged at iteration 56 to a solution with relative residual 9.8e-06.
```

Solution II (MATLAB)

a) We consider the following implementation of the MATLAB function `power_method.m`.

```
function [ lambda, x, k ] = power_method( A, x0, tol, kmax );
% POWER_METHOD power method for the computation of the largest eigenvalue
% (in modulus) of the matrix A (\lambda_1). We assume that A is square,
% |\lambda_1| > |lambda_i| for i=2,...n, and \lambda_1 non zero
% Stopping criterion based on the relative difference of successive
% iterates of the eigenvalue.
```

```
%  [ lambda, x ] = power_method( A, x0, tol, kmax )
%  Inputs: A      = matrix (n x n)
%          x0     = initial vector (n x 1)
%          tol    = tolerance for the stopping criterion
%          kmax   = maximum number of iterations
%  Output: lambda = computed (largest) eigenvalue
%          x      = computed eigenvector correspoding to lambda
%          k      = number of iterations
%

x = x0;
y = x / norm( x );
lambda = y' * A * y;
err = tol + 1;
k = 0;
lambda_old = lambda;

while ( err ≥ tol && k ≤ kmax )

    x = A * y;
    y = x / norm( x );
    lambda = y' * A * y;

    err = abs( lambda - lambda_old ) / abs( lambda );

    lambda_old = lambda;
    k = k + 1;

end

return
```

b) We consider the following MATLAB commands for which we obtain that $\lambda_1^{(k_c)} = 7.2589$ and $k_c = 23$.

```
n = 4;
A = [ 5 -2 -1 0; -2 5 -1 -1; -1 -1 4 -1; 0 -1 -1 5 ];
x0 = ones( n, 1 );
[ lambda1_c, x1, k_c ] = power_method( A, x0, 1e-6,100 );
lambda1_c
%   lambda1_c =
%       7.2589
k_c
%   k_c =
%       23
```

We compute the relative error with respect to the exact value of λ_1 (computed by means of MATLAB). We obtain that $e^{(k_c)} = 7.9711 \cdot 10^{-7}$.

```
lambda1_ex = max( eig( A ) );
rel_err = abs( lambda1_c - lambda1_ex ) / abs( lambda1_ex )
%   rel_err =
%      7.9711e-07
```

c) We consider the following MATLAB commands for which we obtain that $\lambda_n^{(k_c)} = 1.5919$ and $k_c = 6$.

```
invA = inv( A );
[ lambda1_inv_c, x1_inv, k_c ] = power_method( invA, x0, 1e-6, 100 );
lambdan_c = 1 / lambda1_inv_c
%   lambdan_c =
%       1.5919
k_c
%   k_c =
%        6
```

d) We verify that we can obtain the same result of point c) by means of the following commands. We select a shift value $s = 5$ for which we can calculate $\lambda_n^{(k_c)}$ by firstly computing $\lambda_{s,1}^{(k_c)}$.

```
shift = 5;
[ lambda_s, x, k_c ] = power_method( A - shift * eye( n ), x0, 1e-6, 100 );
lambdan_c = lambda_s + shift
%   lambdan_c =
%    1.5919
k_c
%      k_c =
%      13
```

Solution III (MATLAB)

a) We consider the following MATLAB commands similarly to Exercise 1, point a). The matrices M and $\widetilde{M}$ are nonsingular by construction.

```
KK = 4e2;  m = 1;
nnodes = 29; % number of nodes of the bridge
[ A ] = bridge_stiffness_matrix( nnodes, KK );
n = size( A, 2 );
M = m * speye( n, n );
nr = n - 3;
ir = [ 3 : n - 1 ];
Ar = A( ir, ir );
Mr = M( ir, ir );
K_Ar = condest( Ar )
%   K_Ar =
%      1.8838e+04
K_Mr = condest( Mr )
%   K_Mr =
%      1
```

b) We compute the 10 smallest eigenvalues $\widetilde{\lambda}_i$ and eigenmodes $\mathbf{x}_i$ (from $\widetilde{\mathbf{x}}_i$), for $i = 1, \ldots, n$, as follows.

```
neig = 10;
```

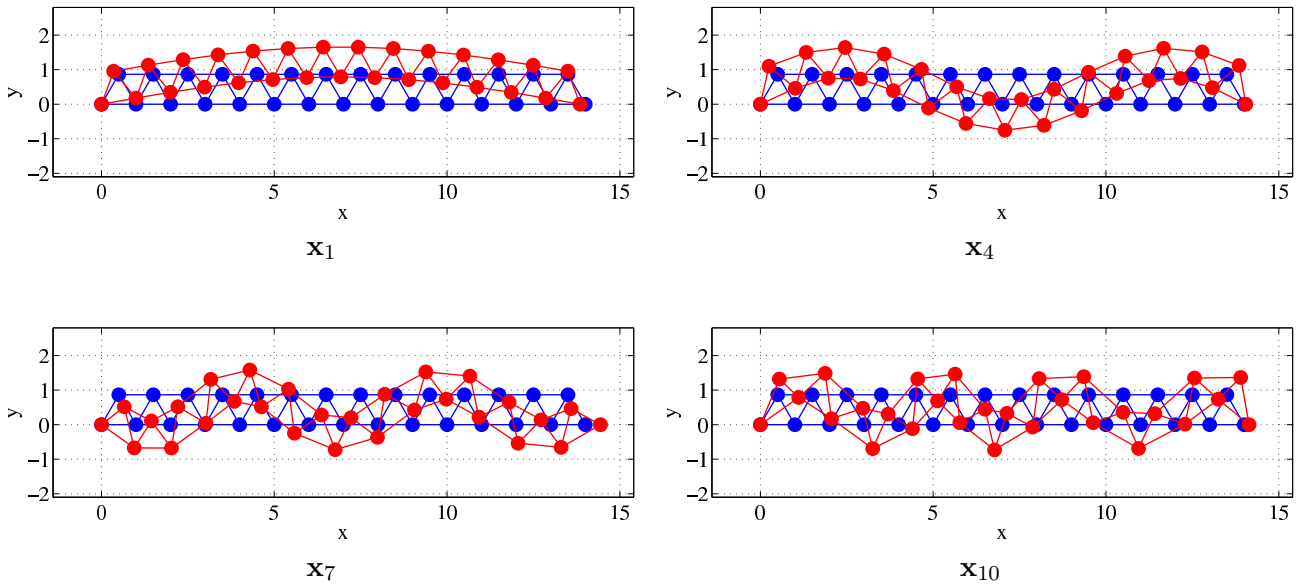

Figure 4: Eigenmodes $\mathbf{x}_1$, $\mathbf{x}_4$, $\mathbf{x}_7$, and $\mathbf{x}_{10}$.

```
[ Xr, Lr ] = eigs( Ar, Mr, neig, 'SM' );
lambda_r = diag( Lr )'
%   lambda_r =
%      0.1758    2.2183    4.3588   10.3889   24.5227   38.0159
%     49.2831   79.0215  102.4560  120.2148
X = zeros( n, neig );
X( ir, : ) = Xr( :, : );
```

We observe that `Lr` is a diagonal sparse matrix of size 10×10 containing the eigenvalues on the diagonal. Similarly, `Xr` is the matrix containing in each column the eigenvector $\widetilde{\mathbf{x}}_i$; the eigenmodes $\mathbf{x}_i$ are obtained from $\widetilde{\mathbf{x}}_i$ by taking into account the displacement constraints.

By considering a multiplicative factor to enhance the visualization of the eigenmodes and taking into account their ordering from the variable `lambda_r`, we obtain the results reported in Figure 4 by means of the following commands.

```
viz_fact = 3;  % rescale facror for visualization of the eigenmodes
% first eigenmode, lambda = 0.1758
[ fig_h ] = plot_bridge( nnodes, viz_fact * X( :, 1 ) )
% fourth eigenmode, lambda = 10.3839
[ fig_h ] = plot_bridge( nnodes, viz_fact * X( :, 4 ) )
% seventh eigenmode, lambda = 49.2831
[ fig_h ] = plot_bridge( nnodes, viz_fact * X( :, 7 ) )
% tenth eigenmode, lambda = 120.2148
[ fig_h ] = plot_bridge( nnodes, viz_fact * X( :, 10 ) )
```