NATIONAL TECHNICAL UNIVERSITY OF ATHENS

GRADUATE COURSE IN COMPUTATIONAL MECHANICS

GRADUATE COURSE THESIS

Metaheuristic algorithms for damage identification in real sized structures

GRADUATE STUDENT
STAVROS CHATZIELEFHERIOU

SUPERVISING PROFESSORS

MANOLIS PAPADRAKAKIS
PROFESSOR
SCHOOL OF CIVIL ENGINEERING

NIKOS LAGAROS
ASSISTANT PROFESSOR
SCHOOL OF CIVIL ENGINEERING

ABSTRACT

The scope of this thesis is to apply metaheuristic algorithms for damage identification in realistic regarding size, member response and eigenvalue approximation (a case of a two-storey steel frame building is examined approximating the eigenvalues via substructuring) civil engineer structures as well as reviewing some of the basic theories and assumptions made.

Two techniques for damage identification are proposed. The problem of damage identification is an inverse problem where one may expect multiple solutions. A discrete value algorithm is proposed in order to control the maximum number of damaged elements for the search. When size and/or number of damages increases the existing methods (mainly sensitivity methods derived from first order perturbation theory) produce more damages then the ones alleged. A technique using the null space of the sensitivity matrix (which is considered a function of the damage factors) is proposed so one can track the multiple solutions finding cases with fewer damaged elements.

INTRODUCTION

Assessment of structural integrity and therefore the determination of the severity and number of damages in structures has always been of great importance to the engineering community, in order to evaluate maintenance, repair and replacement issues. These issues are even more demanding today because of the need to cut down source expenses and adopt a more preserving, greener attitude.

Several non destructive testing tools for structural health monitoring have been developed and the research on the subject is very active. Especially appealing and promising are damage identification techniques utilizing modal data e.g. eigenfrequencies modal shapes or damping ratios. Among the first two, identification using only eigenfrequencies is even more appealing as they have a global nature and can be easily measured at few points. On the other hand the problem is easiest to deal with when mode shapes are included but experimental modal shapes demand more measurements and are in general noisy data (Parameter Identification of M & S, Mroz-Stavroulakis).

Numerous papers dealt with the problem, (e.g. damage identification using variation of eigenfrequencies between damage and undamaged model) Cawley and Adams (1979), Bicanic and Chen (1997), Hassiotis and Jeong (1993,1995) for single or multiple damages ,Hu & Liang (1992,1993) , Bing,Chen & Zhengjia (2012). The problem can be set as a minimization problem and state of the art metaheuristic algorithms can be applied with satisfactory results such as genetic algorithms (Ruotolo & Surace 1997) simulated annealing (He & Wang 2006), particle swarm optimization technique (Saada,Arafa & Nassef 2012) and others.

Thesis order

The thesis is organized in six chapters:

- In the first chapter some theoretical background regarding the derivatives of eigenvalues and eigenvectors and the sensitivity methods for damage identification using eigenfrequencies is given.

- In the second chapter the test examples for analysis and algorithm application are set as well as the physical meaning of the damage factors used for identification.

- In the third chapter the algebraic equations for simple cases are derived in order to demonstrate the technique for tracking multiple solutions finding cases with fewer damaged elements.

- In the fourth chapter a general description of metaheuristic algorithms is given and a review on particle swarm optimization algorithm used for damage identification.

- In the fifth chapter a review on techniques for approximating eigenvalues for large structures via substructuring, in particular component mode synthesis and Rayleigh-Ritz analysis, is given.

- In the sixth chapter numerical results of code implemented in Matlab are given, specifically

    a. For 11 damaged beam cases using Quadratic programming, PSO algorithm and tracking solutions of the same error as Quadprogramming solutions but with fewer damaged elements.

    b. For 11 cases of a damaged small two storey steel frame building using the proposed Discrete value algorithm

    c. For the same cases with single or double damage of the beam and building using the Discrete value algorithm and calculating the eigenfrequencies via component mode synthesis method.

CONTEXT

Chapter 1
Theoretical background

In this chapter we give some theoretical background of the problem, in particular, methods for calculating the derivatives of eigenvalues and eigenvectors and sensitivity methods for damage identification.

1.1 General formulation of the problem

The simulation of the structure is through Finite Element Method and the global stiffness, mass and damping matrices are assembled from the local matrices of each element. The local matrix of every element is considered a function of a parameter (damage factor) Si.

So the global matrices can be considered as:

$$K = \sum_{i=1}^{N} K_i(s_i) \ , \ M = \sum_{i=1}^{N} M_i(s_i) \ , \ C = \sum_{i=1}^{N} C_i(s_i)$$

Where summation means matrices assembly and K_i , M_i , C_i are part of zone matrices corresponding to each element. In the present paper only the stiffness matrix is considered a function of damage factors.

The formulation of the generalized eigenproblem is as follows:

$$\mathrm{K}u+\lambda Cu+\lambda^2 Mu=0 \qquad (1)$$

K,M,C *nxn matrices*

u *nx*1 *vector*

which can be set to standard form

$$A\varphi=\lambda\varphi \qquad (2)$$

where

$$\mathrm{A}=\begin{bmatrix} 0 & \mathrm{I} \\ -\mathrm{M}^{-1}\mathrm{K} & -\mathrm{M}^{-1}C \end{bmatrix} \quad , \quad \varphi=\begin{bmatrix} u \\ \lambda u \end{bmatrix}$$

and for the left eigenvector Ψ we have $\mathrm{A}^T\psi=\lambda\psi$

If there exist matrix Γ such that $\mathrm{A}^T=\Gamma\mathrm{A}\Gamma^{-1}$

then for each of the r=2n, left Ψκ and right Φκ eigenvectors

$$\psi_\kappa=\Gamma\varphi_\kappa \qquad (3)$$

and because Ψκ, Φκ are orthogonal by definition we can normalize the vectors Φκ

$$\varphi_i^{\mathrm{T}} \Gamma \varphi_j = \delta_{ij} \qquad (4)$$

Depending on the form of matrix C there exist real or complex eigenvalues and eigenvectors.

In the present study free undamped vibrations of the structure are considered therefore C=0 and K,M are symmetric positive definite (or semi definite) matrices. The generalized eigenproblem is

$$\mathrm{K}u = \lambda M u \qquad (5)$$

Assuming M is positive definite we consider the decomposition $M = SS^T$, it can be set as the standard eigenproblem

$$\tilde{\mathrm{K}}\tilde{u} = \lambda \tilde{u} \qquad (6)$$
$$\tilde{\mathrm{K}} = S^{-1}\mathrm{K}S^{-T}, \quad \tilde{u} = S^T u$$

1.2 Sensitivity of eigenvalues and eigenvectors

We consider the problem

$$(\mathrm{K} - \lambda_\kappa M)\Phi_\kappa = 0, \quad \kappa = 1,2,...n \qquad (7)$$
$$\Phi_i^{\mathrm{T}} \mathrm{K} \Phi_j = \lambda_j \delta_{ij} \quad , \quad \Phi_i^{\mathrm{T}} \mathrm{M} \Phi_j = \delta_{ij}$$

We can find the sensitivity derivatives of the eigenvalues and eigenvectors with respect to a factor S by differentiating the above equations

$$(\mathrm{K} - \lambda_\kappa \mathrm{M}) \frac{d\Phi_\kappa}{ds} - \frac{d\lambda_\kappa}{ds} M \Phi_\kappa = -(\frac{dK}{ds} - \lambda_\kappa \frac{dM}{ds})\Phi_\kappa \qquad (8)$$

$$\Phi_\kappa^{\mathrm{T}} \mathrm{M} \frac{d\Phi_\kappa}{ds} = -\frac{1}{2} \Phi_\kappa{}^{\mathrm{T}} \frac{dM}{ds} \Phi_\kappa \qquad (9)$$

after pre-multiplying (8) with Φ_κ^T, we obtain the first sensitivity derivative of the eigenvalue

$$\frac{d\lambda_\kappa}{ds} = \frac{-\Phi_\kappa^{\mathrm{T}}(\frac{dK}{ds} - \lambda_\kappa \frac{dM}{ds})\Phi_\kappa}{\Phi_\kappa^{\mathrm{T}} \mathrm{M} \Phi_\kappa} = -\Phi_\kappa{}^{\mathrm{T}}(\frac{dK}{ds} - \lambda_\kappa \frac{dM}{ds})\Phi_\kappa \qquad (10)$$

in order to obtain the eigenvector derivative we consider them to be linear combination of the original eigenvectors that is

$$\frac{d\Phi_\kappa}{ds} = \sum_{j=1}^{n} c_{kj} \Phi_j \qquad (11)$$

And pre-multiplying (8) by Φ_j^T we get

$$c_{kj} = \frac{\Phi_j^T (\frac{dK}{ds} - \lambda_k \frac{dM}{ds}) \Phi_k}{(\lambda_k - \lambda_j) \Phi_j^T \mathrm{M} \Phi_j} , \qquad k \neq j \qquad (12)$$

There are several methods for computing the sensitivity derivatives even without using the whole set of eigenpairs, suitable for larger scale systems where only a few of eigenvalues and eigenvectors are approximated (Nelson 1976). Also from the above equations iterative procedures can be formulated to compute the variations of eigenvalues and eigenvectors.

A way of dealing with the problem is using first order theory and in order to compute the variations of the eigenvalue, to neglect the variations of the eigenvectors, however as we can see from equation (12) that's not always the case especially when two or more eigenvalues are close to each other so the coefficients c_{kj} become large in magnitude.

1.3 Damage identification based on frequency measurements set as a minimization problem

We consider again the problem for the undamaged structure

$$(\mathrm{K}_o - \lambda_{oi} M_o) \Phi_{oi} = 0 , \quad i = 1,2,...N \qquad (13)$$

The eigenvalues are expressed with the Rayleigh quotient

$$\lambda_{oi} = \frac{\Phi_{oi}^T K_o \Phi_{oi}}{\Phi_{oi}^T \mathrm{M}_o \Phi_{oi}} \qquad (14)$$

or

$$\Phi_{oi}^{\mathrm{T}} \mathrm{K}_o \Phi_{oj} = \lambda_{oi} \delta_{ij} \quad , \quad \Phi_{oi}^{\mathrm{T}} \mathrm{M}_o \Phi_{oj} = \delta_{ij}$$

For the damaged structure

$$(\mathrm{K} - \lambda_i M) \Phi_i = 0 , \quad i = 1,2,...N \qquad (15)$$

where

$\mathrm{K} = K_o + \Delta K$, $M = M_o + \Delta M$, $\Phi_i = \Phi_{oi} + \Delta \Phi_i$, $\lambda_i = \lambda_{oi} + \Delta \lambda_i$

We assume $\Delta M = 0$, $M = M_o$ and we have

$$\lambda_i = \Phi_i^{\mathrm{T}} \mathrm{K} \Phi_j \quad , \quad \Phi_i^{\mathrm{T}} \mathrm{M} \Phi_j = \delta_{ij} \qquad (16)$$

We can write

$$(\mathrm{K}+\Delta K)(\Phi_{oi}+\Delta\Phi_i)-(\lambda_{oi}+\Delta\lambda_i)\mathrm{M}(\Phi_{oi}+\Delta\Phi_i)=0$$

Expanding and neglecting the higher order terms

$$(\mathrm{K}-\lambda_{oi}\mathrm{M})\Delta\Phi_i=-\Delta K\Phi_{oi}+\Delta\lambda_i\mathrm{M}\Phi_{oi} \qquad (17)$$

Pre-multiplying by Φ_{oi}^T and because $\Phi_{oi}^T(K-\lambda_{oi}M)=0$ we get the sensitivity of the eigenvalue to changes in the stiffness matrix

$$\Delta\lambda_i=\Phi_{oi}^T\Delta K\Phi_{oi} \qquad (18)$$

From the formulation of the stiffness matrix we can write

$$\mathrm{K}+\Delta K=\sum_{j=1}^{NumElem}K_j^e(1+\delta\kappa_j) \qquad (19)$$

Where summation means matrices assembly. δκ a NumElem-vector being the reduction of the element stiffness multiplying each matrix K_j^e which can be considered as matrix with equal size to K with zeros everywhere but the degrees of freedom of the corresponding element at the global system. Substituting (19) to (18) we get

$$\Delta\lambda_i=\Phi_{oi}^T\Big(\sum_{j=1}^{NumElem}K_j^e\delta\kappa\Big)\Phi_{oi}$$

$$=\begin{bmatrix}\Phi_{oi}^TK_1^e\Phi_{oi} & \dots & \Phi_{oi}^TK_j^e\Phi_{oi}\end{bmatrix}\begin{bmatrix}\delta\kappa_1\\ \dots\\ \delta\kappa_j\end{bmatrix}$$

Considering m changes of eigenvalues we get the linear system

$$D\,\delta\kappa=\Delta\lambda \qquad (20)$$

Where D is a m x NumElem matrix and Δλ a m-vector.

This is the linear system produced from first order perturbation theory and is in general underdetermined (as the number of measured eigenfrequencies is usually less than the number of elements). The solution space is bounded as $-1<\delta\kappa_j<0$, we consider reduction of stiffness up to 100%. In order to solve the equations above we can set them as a minimization problem, requiring that the "damaged" eigenproblem is not far from the initial.

Assuming that the damaged eigenvectors are close to the initial ones we write

$$(\mathrm{K}+\Delta K)\Phi_{oi}-(\lambda_{oi}+\Delta\lambda_i)\mathrm{M}\Phi_{oi}=R_i \qquad (21)$$

Where R_i is the vector of residuals. Simplifying we have

$$R_i=\Delta K\Phi_{oi}-\Delta\lambda_i\mathrm{M}\Phi_{oi} \qquad (22)$$

we can express it's magnitude by the square

$$|| R_i ||^2 = R_i^T R_i = \Phi_{oi}^{\mathrm{T}} \Delta K^2 \Phi_{oi} - 2\Delta\lambda_i \Phi_{oi}^{\mathrm{T}} \Delta \mathrm{K} \mathrm{M} \Phi_{oi} + \Delta\lambda_i^2 \Phi_{oi}^{\mathrm{T}} M^2 \Phi_{oi} \qquad (23)$$

And summing over m changes of eigenvalues

$$g = \sum_{i=1}^{m} || R_i ||^2 = \delta\kappa^{\mathrm{T}} Q \delta\kappa + 2\delta\kappa^{\mathrm{T}} C \qquad (24)$$

Where Q is NumElem x NumElem matrix , C a NumElem-vector with

$$q_{ij} = \sum_{k=1}^{m} \Phi_k^T K_i^e K_j^e \Phi_k \quad , \quad c_i = \sum_{k=1}^{m} \Phi_k^T K_i^e M \Phi_k$$

Where the last term of (6) has been dropped as it does not affect the minimization.

So the problem can be posed as

$$\min\ g = \delta\kappa^{\mathrm{T}} Q \delta\kappa + 2\delta\kappa^{\mathrm{T}} C \qquad (25)$$
$$s.t.\ \ D\delta\kappa = \Delta\lambda \ , \ \ -1 \le \delta\kappa \le 0$$

Which is a quadratic programming problem with linear equality and inequality constraints. It is possible to modify the cost function g in order to account for the eigenvector variation. However, solving the problem and satisfying the equality constraints, implies that a linear correlation stands, which is the case for a few small damages but as the damage size and/or number increases the non linearity effect grows significantly. Alternative methods for solving the above linear system have been proposed such us pseudo inverse techniques were one finds the min-norm solution. The reduction of stiffness can be expressed with the stiffness of an appropriate rotational spring in the assumed crack position in order to derive the size of a crack in beam structures. Finally as proposed in recent work the reduction of stiffness in order to represent damaged elements is justified and closed form solutions for multiple cracked beams are available.

References

[1] Z.Mroz ,G. E. Stavroulakis,Parameter Identification of materials and structures, CISM courses and lectures no 469

[2] S.Hassiotis, G.D.Jeong, Assesement of structural damage from natural frequency measurements, Computers and structures, 49 (1993) 671-691

[3] LI Bing, C.Xuefeng, HE Zhengjia, Three steps meshing based multiple crack identification for structures and its experimental studies, Chinese journal of mechanical engineering 26 (2013) 1

[4] S.Moradi, M.H.Kargozarfard, On multiple crack detection in beam structures,Journal of mechanical science and technology, 27 (1)(2013)47-55

[5] S.Caddemi, A.Morassi, Multi-cracked Euler-Bernoulli beams: mathematical modeling and exact solutions, International Journal of Solids and Structures 50 (2013) 944–956

[6] Wikipedia,Pertrubation theory
[7] R.B.Nelson,Simplified calculation of eigenvector derivatives, AIAA Journal 14 9(1976)

Chapter 2
Test examples

In this chapter we introduce the examples that we used for damage identification techniques and algorithms along with the damage factors which are used for quantification of damage.

2.1 Setting the examples

In the present paper we are applying damage identification techniques for three cases. Starting we examine a simple free-free rod composed of N 1-d finite elements (P1 truss element) regarding the equations that govern the problem and ways to perform identification. Secondly we examine a thin free-free steel beam of rectangular solid cross-section divided in 22 beam elements. Last we examine a small two-storey steel frame building composed by 63 finite elements (S2 elements), the columns of the building are steel members of rectangular hollow cross-section (400 x 400 x 10) while the beams are of type IPB 450.

The element stiffness and mass matrices for each of the three case are:

1 .

```
Kloc= [w1 -w1
       -w1  w1]
```

with `w1 = E*A/L`

The mass matrix is assembled assuming a single mass of magnitude m at each node resulting in a N x N unit mass matrix.

2.

```
Kloc=[w2  w3 -w2  w3
      w3  w4 -w3  w5
     -w2 -w3  w2 -w3
      w3  w5 -w3  w4]
with
w2 = 12*E*I/(L*L*L)
w3 = 6*E*I/(L*L)
w4 = 4*E*I/L
w5 = 2*E*I/L
```

For the beam case we have a consistent local mass matrix

```
Mloc=(mf/420)*[156     22*L      54     -13*L
              22*L   4*(L^2)   13*L   -3*(L^2)
               54      13*L     156      -22*L
              -13*L  -3*(L^2)  -22*L   4*(L^2)]
```

where

```
E=200*(10^9)Pa , Steel's modulus of elasticity
L=Ltot/Nelem ,  Element's length
Nelem=22  ,  Number of elements
Ltot=0.55m , Total length of the beam
dens=7850 Kg/m^3  ,  Steel's density
```

```
b=7.86 mm , Width of the cross section
h=7.86 mm , Height of the cross section
mf=dens*(b*h)*L
I=(b*(h^3))/12 , Moment of inertia of the cross section
```

The beam case was derived from an experimental case study on PSO algorithm (Saada, Arafa & Nassef 2012), a small mass on the 10^{th} element was added in order to simulate the mass of the accelerometer in the original paper (maccel=0.03 Kgr).

3.

```
Kloc=[w1 0   0   0    0   0   -w1 0   0    0    0    0
      0  w2  0   0    0   w3  0   -w2 0    0    0    w3
      0  0   w6  0   -w7  0   0    0 -w6   0   -w7   0
      0  0   0   w10  0   0   0    0   0 -w10   0    0
      0  0  -w7  0    w8  0   0    0   w7   0    w9   0
      0  w3  0   0    0   w4  0   -w3  0   0    0    w5
     -w1 0   0   0    0   0   w1   0   0   0    0    0
      0 -w2  0   0    0  -w3  0    w2  0   0    0   -w3
      0  0  -w6  0    w7  0   0    0   w6   0    w7   0
      0  0   0  -w10  0   0   0    0   0   w10  0    0
      0  0  -w7  0    w9  0   0    0   w7   0    w8   0
      0  w3  0   0    0   w5  0   -w3  0   0    0    w4 ]
```

with

```
w1 = E*A/L
w2 = 12*E*Iz/(L*L*L)
w3 = 6*E*Iz/(L*L)
w4 = 4*E*Iz/L
w5 = 2*E*Iz/L
w6 = 12*E*Iy/(L*L*L)
w7 = 6*E*Iy/(L*L)
w8 = 4*E*Iy/L
w9 = 2*E*Iy/L
w10 = G*J/L
```

where

```
E=2.1e8 KPa , Steel's modulus of elasticity
G=0.8e8 Kpa, Steel's shear modulus
A=0.0156 m^2 cross section area for the columns
A=0.0218 m^2 cross section area for the beams
Iy,Iz,G = 0.0003935 , 0.000395 , 0.00059319  m4
          Moments of inertia along y,z and x local axis
          For the columns
Iy,Iz,G = 0.0007989 , 0.0001172 , 3.88e-06  m4
          Moments of inertia along y,z and x local axis
          For the beams (IPB 450)
L , Element's length
```

For the stiffness matrix the number of nodes is 58, each node has 6 degrees of freedom (three displacements and three rotations) resulting in a 312 x 312 global stiffness matrix. The mass matrix is assembled globally, 10 KN/m^2 load at each floor is assumed and the mass is distributed accordingly to the area of influence of each node of the model. The significant degrees of freedom of the nodes at the edges of the floors are assumed to be x,y displacements and z rotation while those at the midspan of the beams are x,y,z displacements and z rotation. Also 0,1 T (or 0.1 Tm) mass is assumed at the remaining DOFs resulting in a 312 x 312 diagonal mass matrix.

Next, two photos of the structure are shown, from the static analysis program used for verification analysis (3dr Strad) and from the program used to perform the analysis for the damage identification (Matlab).

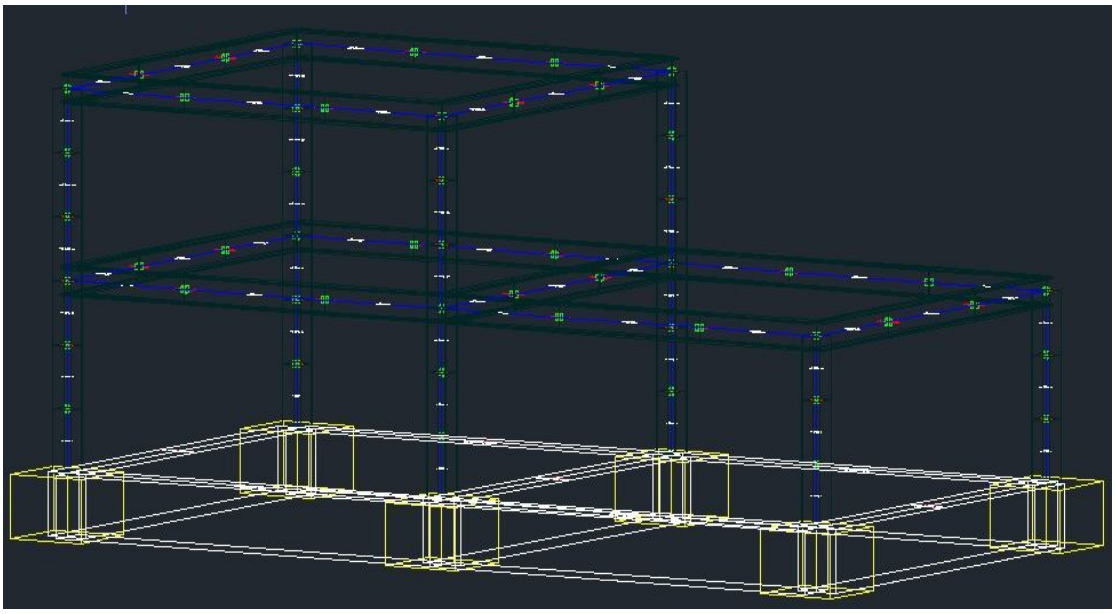


Figure 2.1 Picture of the model from the analysis program 3DR Strad

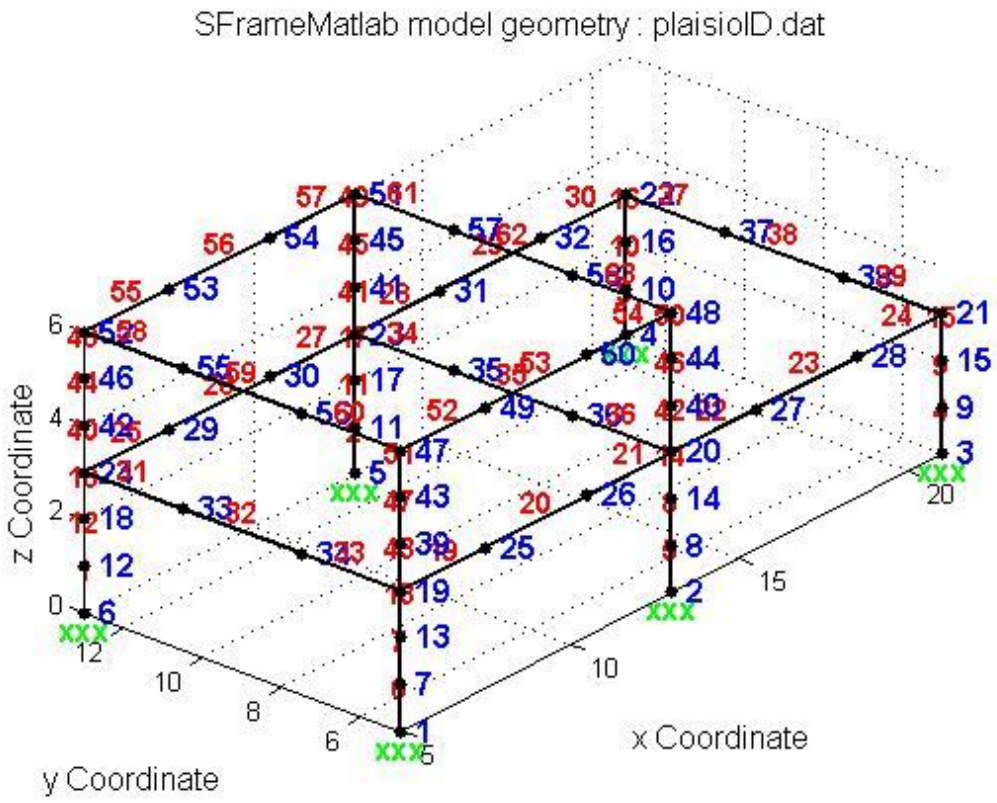


Figure 2.2 Frame building picture from the Matlab model

2.2 Interpretation of damage factors

From equations (7), damage factors can be seen as a percentage reduction of stiffness (-1<δκ<0) or as the percentage of the remaining stiffness S_i.

$$S_i = 1 + \delta\kappa_i \quad , \quad 0 \le S_i \le 1 \qquad (26)$$

The global stiffness matrix for the damaged structure is then formulated from the element stiffness matrices multiplied by the element damaged factor S_i . Magnitude of the factor equal to 1 suggests that the element is intact and on the other hand equal to 0 suggests that the element's stiffness is zero.

The damage factor can relate to geometric characteristics of the element cross section. For each of the three cases we will derive to the usage of the damage factor described above.

For the first case (truss element) it can be expressed as the ratio $S_i = \frac{A_i}{A_{oi}}$, A_{oi} being the initial area of the cross section , A_i the cross section of the damaged element. Damaged can be seen as a reduction of the element's cross section.

For the second case the damage factor can be expressed as the ratio $S_i = \frac{I_i}{I_{oi}} = \frac{b h_i^3 / 12}{b h_{oi}^3 / 12} = (\frac{h_i}{h_{oi}})^3$, so damage is expressed as a reduction from I_{oi} to I_i of the moment of inertia, or as a symmetric reduction of the height's cube of the cross section as pictured below.

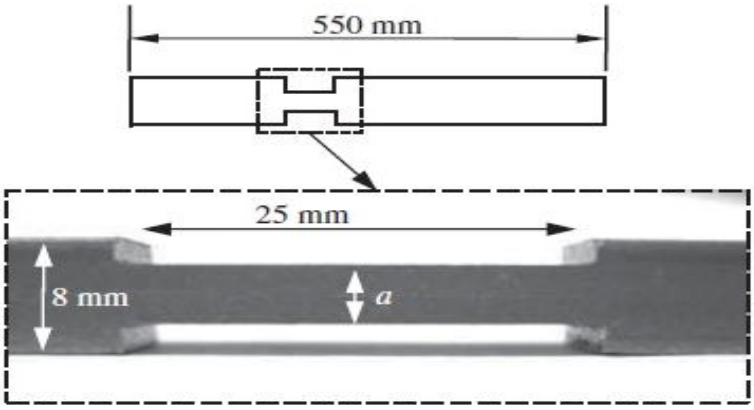


Figure 2.3 Height reduction of element's cross-section

We should note that in the present analysis 22 elements are used. However, using a more dense discretization one can simulate a crack like defect but always assuming a linear behavior which means that the edges of the crack do not close. Finally for the case of the two-storey building we can come up with one damage factor multiplying each element's stiffness matrix using the ratio $S_i = \frac{dA_d}{dA}$, where dA is the infinitesimall area of the undamaged cross section and dA_d of the damaged one and assuming that the reduction of the area is distributed uniformly

along both x and y axis. We illustrate this at the picture below assuming $S_i = \frac{dA_d}{dA} = \frac{A_d}{A} = 0.333 = 1/3$

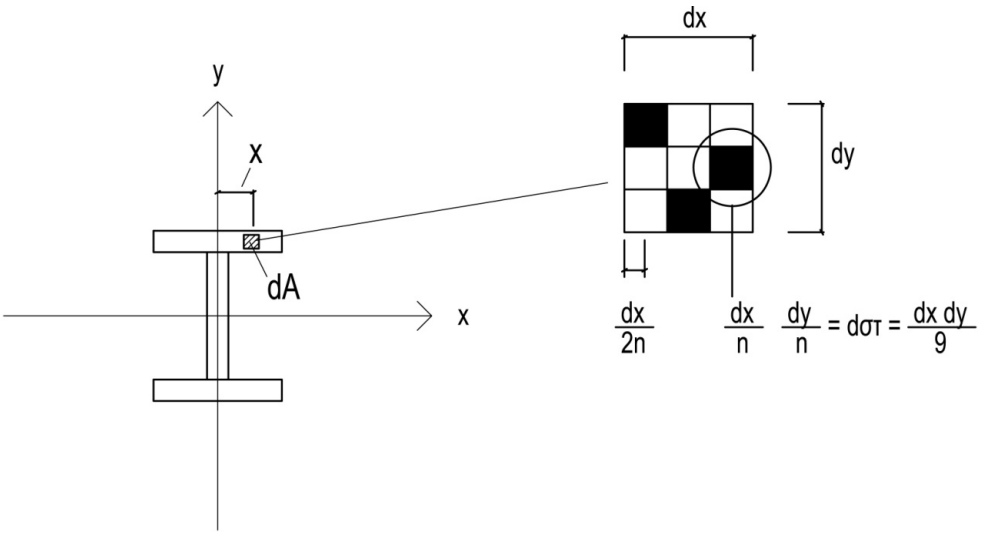


Figure 2.4 Uniform reduction of cross-section's area

In this example n=3 and we have

$\frac{dA_d}{dA} = \frac{n\frac{dx}{n}\frac{dy}{n}}{n^2\frac{dx}{n}\frac{dy}{n}} = \frac{1}{3}$, for the infinitesimall moment of inertia over y axis

$$dI_{ydam} = (x - \frac{2dx}{2n})^2 d\sigma\tau + x^2 d\sigma\tau + (x + \frac{2dx}{2n})^2 d\sigma\tau$$
$$= 3x^2 d\sigma\tau + 2\frac{dx^2}{n^2} d\sigma\tau$$
$$= \frac{1}{3}x^2 dxdy + \frac{2dx^2 dxdy}{3^4}$$

$$\frac{dI_{ydam}}{dI_y} = \frac{1}{3} + \frac{2dx^2}{3^4 x^2} \qquad (27)$$

as dx tends to zero we have $\frac{dI_{ydam}}{dI_y} = \frac{1}{3}$, so

$$I_{ydam} = \int\int dI_{ydam} = \frac{1}{3}\int\int dI_y = \frac{1}{3}I_y \qquad (28)$$

The same is true for I_x and $I_r=I_x+I_y$. We note that we can express damage in different forms coming up with more than one damage factor for each element.

References

[1] M.Papadrakakis, Contemporary analysis of structures

[2] M.M.Saada, M.H.Arafa, A.O.Nashef, Finite element model updating approach to damage identification in beams using particle swarm optimization, Engineering optimization, 2013 45 6 677-696

Chapter 3
Uniqueness issues

In this chapter we give the algebraic equations governing the simple rod problem and present the method of tracking multiple solutions on the intersection of hypersurfaces that represent fixed eigenvalues for the system.

3.1 Algebraic equations governing the problem

Let us consider the first case of the truss element rod, each node having one degree of freedom along the rod axis. The local stiffness matrix for the possibly damaged element is $K_i = \frac{EA}{L}\begin{bmatrix} S_i & -S_i \\ -S_i & S_i \end{bmatrix}$ the mass matrix is assembled globally with a mass m at each node. If we consider, initially only two elements of same length L and if λa_j is the a priori known eigenvalue (derived from the measured eigenfrequency) the eigeproblem can be written as

$$Ku - \lambda Mu = 0$$

$$\frac{EA}{L}\begin{bmatrix} S_1 & -S_1 & 0 \\ -S_1 & (S_1+S_2) & -S_2 \\ & -S_2 & S_2 \end{bmatrix} u - m\lambda a_j \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} u = 0$$

If the normalized eigenvalue is $\lambda_j = \frac{Lm}{EA}\lambda\alpha_j$ and in order to be such for the damaged system we must have

$$Det\begin{vmatrix} S_1-\lambda_j & -S_1 & 0 \\ -S_1 & (S_1+S_2)-\lambda_j & -S_2 \\ 0 & -S_2 & S_2-\lambda_j \end{vmatrix} = 0$$

expanding we have

$$Det_{n=2} = -3\lambda_j S_1 S_2 + 2\lambda_j^2 (S_1+S_2) - \lambda_j^3 = 0 \qquad (29)$$

Let $Y_1 = S_1 S_2$, $Y_2 = (S_1+S_2)$ and assuming two eigenvalues are known (the first eigenvalue is always zero as the rod has free-free supports expressing the unstressed movement of the system as a rigid body) then we have the linear system

$$\begin{bmatrix} -3\lambda_2 & 2\lambda_2^2 \\ -3\lambda_3 & 2\lambda_3^2 \end{bmatrix}\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} \lambda_2^3 \\ \lambda_3^3 \end{bmatrix} \qquad (30)$$

Solving the above system we get the sum and the product of the damage factors.

For 3 elements we have

$$Det\begin{vmatrix} S_1-\lambda_j & -S_1 & 0 & 0 \\ -S_1 & (S_1+S_2)-\lambda_j & -S_2 & 0 \\ 0 & -S_2 & (S_2+S_3)-\lambda_j & -S_3 \\ 0 & 0 & -S_3 & S_2-\lambda_j \end{vmatrix}=0$$

And expanding

$$Det_{n=3}=-4\lambda_j S_1S_2S_3+\lambda_j^2(3S_1S_2+4S_1S_3+3S_2S_3)-2\lambda_j^3(S_1+S_2+S_3)+\lambda_j^4 \\ =0 \qquad (31)$$

we can set

$$Y_1=S_1S_2S_3 \quad , \quad Y_2=3S_1S_2+4S_1S_3+3S_2S_3 \quad , \quad Y_3=(S_1+S_2+S_3)$$

And solve the linear system

$$\begin{bmatrix} -4\lambda_2 & \lambda_2^2 & -2\lambda_2^3 \\ -4\lambda_3 & \lambda_3^2 & -2\lambda_3^3 \\ -4\lambda_4 & \lambda_4^2 & -2\lambda_4^3 \end{bmatrix}\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \end{bmatrix}=-\begin{bmatrix} \lambda_2^4 \\ \lambda_3^4 \\ \lambda_4^4 \end{bmatrix} \qquad (32)$$

For 4 elements expanding the determinant yields

$$Det_{n=4}=-5\lambda_j S_1S_2S_3S_4+\lambda_j^2(4S_2S_3S_4+6S_1S_3S_4+6S_1S_2S_4+4S_1S_2S_3) \\ -\lambda_j^3(4S_1S_4+4S_2S_4+3S_3S_4+3S_2S_3+4S_1S_3+3S_1S_2) \\ +2\lambda_j^4(S_1+S_2+S_3+S_4)-\lambda_j^5=0 \qquad (33)$$

Setting

$$Y_1=S_1S_2S_3S_4 \quad , \quad Y_2=4S_2S_3S_4+6S_1S_3S_4+6S_1S_2S_4+4S_1S_2S_3$$
$$Y_3=-(4S_1S_4+4S_2S_4+3S_3S_4+3S_2S_3+4S_1S_3+3S_1S_2)$$
$$Y_4=(S_1+S_2+S_3+S_4)$$

The linear system is

$$\begin{bmatrix} -5\lambda_2 & \lambda_2^2 & \lambda_2^3 & 2\lambda_2^4 \\ -5\lambda_3 & \lambda_3^2 & \lambda_3^3 & 2\lambda_3^4 \\ -5\lambda_4 & \lambda_4^2 & \lambda_4^3 & 2\lambda_4^4 \\ -5\lambda_5 & \lambda_5^2 & \lambda_5^3 & 2\lambda_5^4 \end{bmatrix}\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{bmatrix}=\begin{bmatrix} \lambda_2^5 \\ \lambda_3^5 \\ \lambda_4^5 \\ \lambda_5^5 \end{bmatrix} \qquad (34)$$

For the specific problem, from the properties of the determinant one can show that the following recursive formula stands

$$Det_n = (S_n - \lambda) Det_{n-1} - \lambda S_n Det_{n-2} - \lambda S_n S_{n-1} Det_{n-3} - \ldots$$
$$- \lambda S_n S_{n-1} S_{n-2} \ldots S_{\kappa+2} Det_{\kappa} - \ldots - \lambda S_n S_{n-1} S_{n-2} \ldots S_3 Det_1$$
$$- \lambda S_n S_{n-1} \ldots S_3 S_2 (S_1 - \lambda) \qquad (35)$$

If we expand we can see that each monomial is a product of a coefficient and n+1 terms of the eigenvalue and the damage factors. The monomials with λ^{n+1} come up from the first term, the ones with λ^n and one damage factor come up from the two first terms of the above equation while the monomials with λ and n distinct damage factors come up from the first and last term, thus it can be proved by induction that we have the terms

$$-(n+1)\lambda S_1 S_2 \ldots S_n \quad , \quad (-1)^n 2\lambda^n (S_1 + S_2 + \ldots + S_n) \quad , \quad (-1)^{n-1} \lambda^{n+1}$$

the rest of the terms have common factor λ^k with k=2,3…n-1 so

Although finding the values for each damage factor is a hard problem

setting $Y_1 = S_1 S_2 S_3 \ldots S_n$, $Y_n = (S_1 + S_2 + S_3 + \ldots S_n)$

we can solve the following linear system and find the sum and product of the damage factors without having explicit formulae for Y_2 , Y_2 , … , Y_{n-1}

$$\begin{bmatrix} -(n+1)\lambda_2 & \ldots & \lambda_2^m & \ldots & (-1)^n 2\lambda_2^n \\ .. & & .. & & .. \\ -(n+1)\lambda_k & \ldots & \lambda_k^m & \ldots & (-1)^n 2\lambda_k^n \\ .. & & .. & & .. \\ -(n+1)\lambda_{n+1} & \ldots & \lambda_{n+1}^m & \ldots & (-1)^n 2\lambda_{n+1}^n \end{bmatrix} \begin{bmatrix} Y_1 \\ .. \\ Y_m \\ .. \\ Y_n \end{bmatrix} = (-1)^n \begin{bmatrix} \lambda_2^{n+1} \\ .. \\ \lambda_k^{n+1} \\ .. \\ \lambda_{n+1}^{n+1} \end{bmatrix} \qquad (36)$$

The values of the damage factors are between 0 and 1, so solving the above system we get valuable information for the total damage of the rod as well as it's distribution. If we have for example 10 elements, for the intact the sum is 10 and the product is 1, suppose that solving the system we get 9.50 for the sum if this is due to 1 damage of 0.5 factor the product is 0.5, if we have two damages of 0.75 factor for each the product will be 0.5625 and so forth. However a set of n eigenvalues is needed and as the number of elements increases, because of the large powers of λ, the system becames ill-conditioned and difficult to solve.

3.2 Possible solutions uniqueness issues – Tracking the intersection of hypersurfaces for fixed eigenvalues

One important aspect that should be noted when considering damage identification using only eigenfrequencies should be the expectation one has from it. We usually have a small number of eigenvalues at our disposal, much smaller than the number of elements who suggest possible damage positions. The number of the unknown damage factors is larger than that of the equations and we can expect that the system will have an infinite number of solutions. If we consider a limited number of damages (e.g. equal or less than the number of the measured eigenfrequencies) we may expect a finite number of solutions and therefore search for solutions using one or more of the available algorithms.

On the other hand a variation of the eigenfrequencies suggest that if the mass of the structure is not altered, there certainly is a change in the stiffness. Smaller eigenfrequencies suggest less rigid structures, the measured eigenfrequencies give infinite number of solutions who lie on a subspace of all possible configurations of our system. The question now can be posed as: which of these configurations are most dangerous for our structure? In the case of a statically determinate structure one plastic hinge would suggest collapse so we would, certainly, be interested in the element with the greatest amount of damage. However, it is possible that a scattering of the damages (more damages, less magnitude) could make the structure prone to other phenomena.

There are certain combinations of damage factors that produce the same set of eigenfrequencies, for example damages at symmetric elements. In the building examined, in various occasions, apart from the damage scenario, which most of the times is detected there are some other combinations that produce almost identical first ten eigenfrequencies, sometimes at approximate locations of the structure and sometimes not. The scope of this chapter is to delve deeper in these subjects.

Let's consider again the case of the 1-d rod of 3 elements. Also we consider elements 1 and 3 to be undamaged and element 2 having damage resulting in 70% decrease in stiffness. The damage factor vector is $S=[1\ 0.3\ 1]^T$. We calculate the eigenvalues of the system and assume that only the second eigenvalue is known and we wish to find all the possible combinations of damage factors that produce this eigenvalue. From equation (8) and creating a mesh grid of values from 0 to 1 for the damage factors S1 and S2 and solve each time for S3, we get the following surface plot.

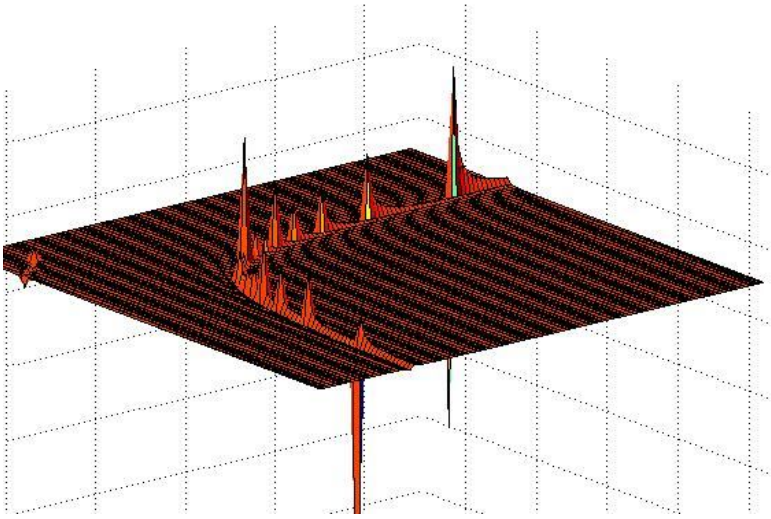


Figure 3.1 Plot of equation (31) solved for S3

Because S3 takes singular values and we are only interested in the [1,1,1] cube we cut the values of S3 where abs(S3)>1.5 and we have the following plot.

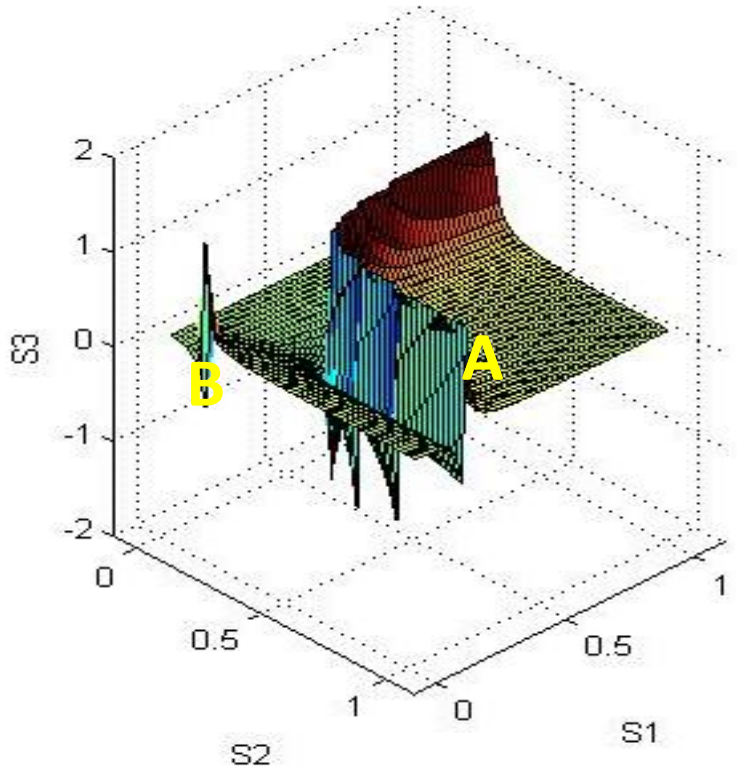


Figure 3.2 Plot of equation (31) solved for S3

We can see two surfaces with a singular "zone" as a border between them. Equation (31) holds both for the known eigenvalue being the second and the third of the system and at the plot we can see exactly that (actually there is a third smaller area and a smaller "zone" where the known eigenvalue becomes the fourth one of the system). Area A is the surface between the singular "zone" and surfaces S1=1 , S2=1. The triplets of S1,S2 and S3 on this area satisfy equation (31) and give the eigenvalue as the second for the system. The values of the damage factors corresponding at area B produce the fixed eigenvalue as the third for the system, here our system changes and a new near zero or negative eigenvalue appears as the fixed eigenvalue descents from the second to the third place which means that matrix K is no longer positive definite. We are interested in the combinations at area A.

If we have some number of measured eigenfrequencies our solutions exist on the intersection of the hypersurfaces of each fixed eigenvalue. If we find (from an optimization algorithm for example) one solution we could "move" along the intersection, finding other solutions that produce the same eigenvalues for the system. One way to do this is through the derivatives of the eigenvalues. If $dS=[dS1,dS2,\ldots,dSn]^T$ (n is the number of elements) are the infinitesimal changes of the damage factors, we have:

$$
\begin{aligned}
D_v\lambda_i &= (grad(\lambda_i(S_1,S_2,...,S_n))\cdot d\vec{S} \\
&= \frac{\partial\lambda_i}{\partial S_1}dS_1 + \frac{\partial\lambda_i}{\partial S_2}dS_2 + \ldots + \frac{\partial\lambda_i}{\partial S_n}dS_n \\
&= (\Phi_i^T\frac{dK}{dS_1}\Phi_i)dS_1 + (\Phi_i^T\frac{dK}{dS_2}\Phi_i)dS_2 + \ldots + (\Phi_i^T\frac{dK}{dS_n}\Phi_i)dS_n \\
&= (\Phi_i^T K_1^e\Phi_i)dS_1 + (\Phi_i^T K_2^e\Phi_i)dS_2 + \ldots + (\Phi_i^T K_n^e\Phi_i)dS_n \\
&= [(\Phi_i^T K_1^e\Phi_i)\ \ (\Phi_i^T K_2^e\Phi_i)\ \ \ldots\ \ (\Phi_i^T K_n^e\Phi_i)]\cdot d\vec{S} \qquad (37)
\end{aligned}
$$

Where $D_v\lambda_i$ stands for the directional derivative of the function which shows how the eigenvalue changes along some direction dS, K is the global stiffness matrix , K_j^e the element stiffness matrix for the global system and $\Phi i(S1,S2,\ldots,Sn)$ the i-th eigenvector. If we consider m eigenvalues we can define the vector valued function $\Lambda=[\lambda_1,\lambda_2,...,\lambda_m]^T$ than the jacobian matrix of Λ is

$$J(\Lambda)=\begin{pmatrix} \Phi_1^T K_1^e \Phi_1 & \cdots & \Phi_1^T K_n^e \Phi_1 \\ \vdots & \ddots & \vdots \\ \Phi_m^T K_1^e \Phi_m & \cdots & \Phi_m^T K_n^e \Phi_m \end{pmatrix} \qquad (38)$$

This matrix is essentially almost the same as matrix D of equation (20) only that this time is considered as a function of S. If we are on a point (S1,S2,…,Sn) and we want to move at the intersection of the hypersurfaces of the fixed eigenvalues all we have to do is move along the null space of J(Λ). We calculate the null space (using for example singular value decomposition) and move along one combination of the null vectors multiplied by a small scalar. At the new point, we calculate again J(Λ) and it's null space and move again.

We implemented this for the beam example as shown below. Initially we put S1-S11 =0.85 and S12-S22=1 (figure 3.3), m=5 (first five non zero eigenfrequencies) and used the above procedure to move along the first null vector. When some Si has value of 1 or approached, we zeroed the corresponding column of J(Λ). We finally had the damage factors of figure 3.4. As the step size became smaller we approached the hypersurfaces with linear convergence.

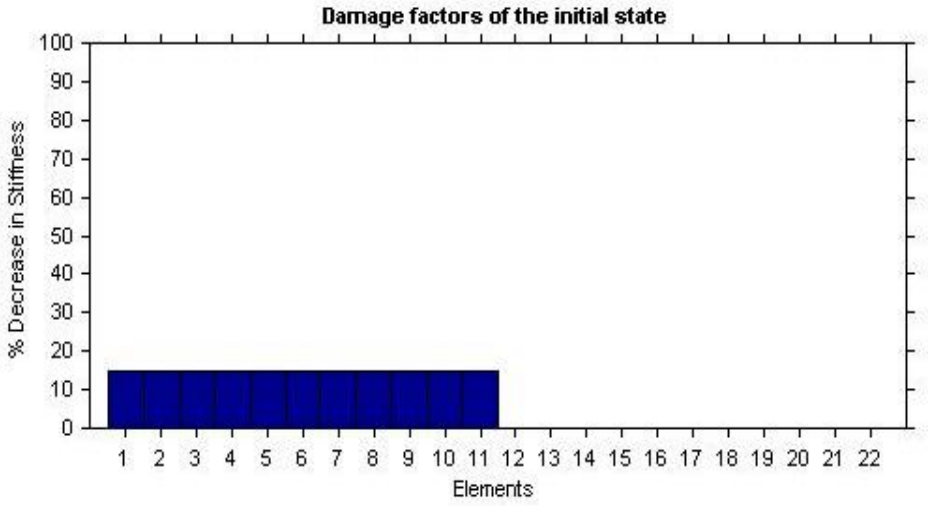


Figure 3.3 Damage factors before moving along the intersection

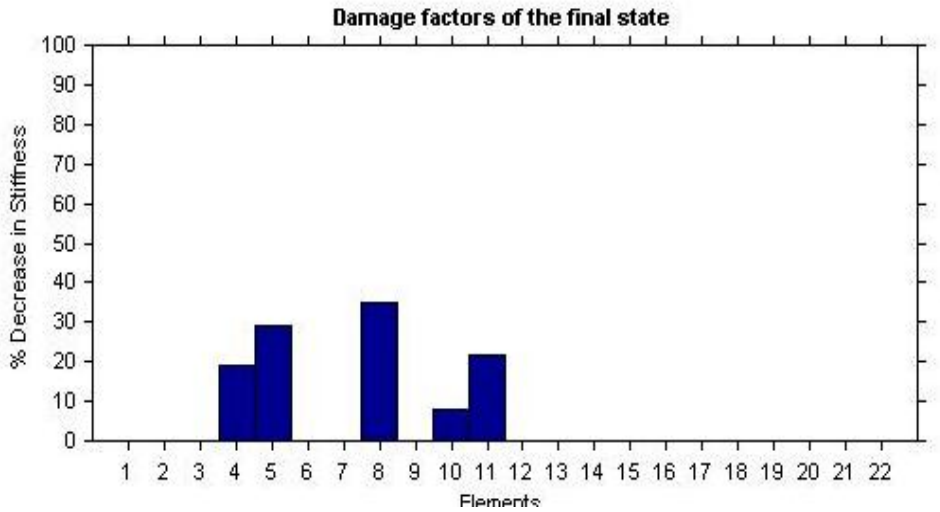


Figure 3.4 Damage factors after moving along the intersection

Below the eigenfrequencies for the initial and the final state are presented along with the error $(\omega_{init}-\omega_{final})/\omega_{init}$

Initial eigenfrequencies (Hz)	Final eigenfrequencies (Hz)	$(\omega_{init}-\omega_{final})/\omega_{init}$
121,2920	121,2867	4,36122E-05
349,9626	349,9479	4,19365E-05
661,5075	661,4799	4,17733E-05
1108,1162	1108,0842	2,88949E-05
1674,6198	1674,5533	3,96861E-05

Figure 3.5 Initial and final eigenfrequencies

In the beam example before applying PSO algorithm as we will describe later, we applied quadratic programming optimization as described in chapter 1.3 . If the number of damages are three and more (or large in magnitude) the optimization tends to give more than five damage elements with small error whatsoever, we used the above mentioned procedure to move on the intersection and find the minimum damaged element case with the same error.

Some other similar techniques could be applied but have not been tested in the present paper. We could calculate the difference for the eigenvalues of the initial (undamaged) and the final state $\Delta\Lambda=\Lambda init-\Lambda final$, divide by k and solve at each step $J(\Lambda)dS=(1/k)\Delta\Lambda$, if the number of damaged elements produced is greater than the size of Λ we can use the null space of $J(\Lambda)$ "move" along the intersection and track a case with fewer damaged elements but the same eigenvalues.

An alternative way to derive the same matrix without solving the eigenvalue problem at each step is with the use of Jacobi formula for the derivative of the determinant

$$\frac{d}{dt}\det A(t) = \operatorname{tr}(\operatorname{adj}(A(t))\frac{dA(t)}{dt}). \quad (39)$$

Equivalently, if *dA* stands for the differential of *A*, the formula is

$$d\det(A) = \operatorname{tr}(\operatorname{adj}(A)\,dA). \quad (40)$$

where $A = K - \lambda \mathrm{M}$ and demanding that $ddet(A) = 0$, $d\lambda_i = 0$ here we have to calculate the adjugate of A at each step.

Chapter 4
Optimization

In this chapter we give some definitions and introductory information regarding the global optimization problem using metaheuristics and some background on PSO algorithm in general as well as for damage identification.

4.1 Metaheuristics

In the last 20 years, a new kind of approximate algorithm has emerged which basically tries to combine basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring a search space. These methods are nowadays commonly called *metaheuristics*. The term *metaheuristic*, derives from the composition of two Greek words. *Heuristic* derives from the verb *heuriskein* (ευρισκειν) which means "to find", while the suffix *meta* means "beyond, in an upper level". Before this term was widely adopted, metaheuristics were often called *modern heuristics*.

This class of algorithms includes, but is not restricted to , Ant Colony Optimization (ACO), Evolutionary Computation (EC) including Genetic Algorithms (GA), Iterated Local Search (ILS), Simulated Annealing (SA), and Tabu Search (TS). In the following we quote some definitions of the term metaheuristic proposed by some researchers:

a. A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions.

b. A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just construction method.

c. Metaheuristics are typically high-level strategies which guide an underlying, more problem specific heuristic, to increase their performance. The main goal is to avoid the disadvantages of iterative improvement and, in particular, multiple descent by allowing the local search to escape from local optima. This is achieved by either allowing worsening moves or generating new starting solutions for the local search in a more "intelligent" way than just providing random initial solutions. Many of the methods can be interpreted as introducing a bias such that high quality solutions are produced quickly. This bias can be of various forms and can be cast as descent bias (based on the objective function), memory bias (based on previously made decisions) or experience bias (based on prior performance). Many of the metaheuristic approaches rely on probabilistic decisions made during the search. But, the main difference to pure random search is that in metaheuristic algorithms randomness is not used blindly but in an intelligent, biased form.

d. A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem.

Summarizing, we outline fundamental properties which characterize metaheuristics:

- Metaheuristics are strategies that "guide" the search process.
- The goal is to efficiently explore the search space in order to find (near)optimal solutions.
- Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.
- Metaheuristic algorithms are approximate and usually non-deterministic.
- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- The basic concepts of metaheuristics permit an abstract level description.
- Metaheuristics are not problem-specific.
- Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.
- Todays more advanced metaheuristics use search experience (embodied in some form of memory) to guide the search.

In short we could say that metaheuristics are high level strategies for exploring search spaces by using different methods. Of great importance hereby is that a dynamic balance is given between *diversification* and *intensification*. The term diversification generally refers to the exploration of the search space, whereas the term intensification refers to the exploitation of the accumulated search experience. These terms stem from the Tabu Search field and it is important to clarify that the terms *exploration* and *exploitation* are sometimes used instead, for example in the Evolutionary Computation, with a more restricted meaning. In fact, the notions of exploitation and exploration often refer to rather short term strategies tied to randomness, whereas intensification and diversification also refer to medium and long term strategies based on the usage of memory. The balance between diversification and intensification as mentioned above is important, on one side to quickly identify regions in the search space with high quality solutions and on the other side not to waste too much time in regions of the search space which are either already explored or which do not provide high quality solutions.

The search strategies of different metaheuristics are highly dependent on the philosophy of the metaheuristic itself. There are several different philosophies apparent in the existing metaheuristics. Some of them can be seen as "intelligent" extensions of local search algorithms. The goal of this kind of metaheuristic is to escape from local minima in order to proceed in the exploration of the search space and to move on to find other hopefully better local minima. This is for example the case in Tabu Search, Iterated Local Search, Variable Neighborhood Search, GRASP and Simulated

Annealing. These metaheuristics (also called trajectory methods) work on one or several neighborhood structure(s) imposed on the members (the solutions) of the search space.

We can find a different philosophy in algorithms like Ant Colony Optimization and Evolutionary Computation. They incorporate a learning component in the sense that they implicitly or explicitly try to learn correlations between decision variables to identify high quality areas in the search space. This kind of metaheuristic performs, in a sense, a biased sampling of the search space. For instance, in Evolutionary Computation this is achieved by recombination of solutions and in Ant Colony Optimization by sampling the search space in every iteration according to a probability distribution.

There are different ways to classify and describe metaheuristic algorithms. Depending on the characteristics selected to differentiate among them, several classifications are possible, each of them being the result of a specific viewpoint. We briefly summarize the most important ways of classifying metaheuristics.

e. Nature-inspired vs. non-nature inspired. Perhaps, the most intuitive way of classifying metaheuristics is based on the origins of the algorithm. There are nature-inspired algorithms, like Genetic Algorithms and Ant Algorithms, and non nature-inspired ones such as Tabu Search and Iterated Local Search. This classification is not very meaningful for the following two reasons. First, many recent hybrid algorithms do not fit either class (or, in a sense, they fit both at the same time). Second, it is sometimes difficult to clearly attribute an algorithm to one of the two classes.

f. Population-based vs. single point search**.** Another characteristic that can be used for the classification of metaheuristics is the number of solutions used at the same time: Does the algorithmwork on a population or on a single solution at any time? Algorithms working on single solutions are called *trajectory methods* and encompass local search-based metaheuristics, like Tabu Search, Iterated Local Search and Variable Neighborhood Search. They all share the property of describing a trajectory in the search space during the search process. Population-based metaheuristics, on the contrary, perform search processes which describe the evolution of a set of points in the search space.

g. Dynamic vs. static objective function. Metaheuristics can also be classified according to the way they make use of the objective function. While some algorithms keep the objective function given in the problem representation "as it is", some others, like Guided Local Search (GLS), modify it during the search. The idea behind this approach is to escape from local minima by modifying the search landscape. Accordingly, during the search the objective function is altered by trying to incorporate information collected during the search process.

h. One vs. various neighborhood structures. Most metaheuristic algorithms work on one single neighborhood structure. In other words, the fitness landscape topology does not change in the course of the algorithm. Other metaheuristics, such as Variable Neighborhood Search (VNS), use a set of neighborhood structures which gives the possibility to diversify the search by swapping between different fitness landscapes.

i. Memory usage vs. memory-less methods. A very important feature to classify metaheuristics is the use they make of the search history, that is, whether they use memory or not. Memory-less algorithms perform a Markov process, as the information they exclusively use to determine the next action is the current state of the search process. There are several different ways of making use of memory. Usually we differentiate between the use of short term and long term memory. The first usually keeps track of recently performed moves, visited solutions or, in general, decisions taken. The second is usually an accumulation of synthetic parameters about the search. The use of memory is nowadays recognized as one of the fundamental elements of a powerful metaheuristic.

4.2 PSO algorithm

In numerous optimization problems encountered in different areas of scientific inquiry, the search for a solution is identified with the discovery of the global minimizer of a real valued objective function $f : S \rightarrow R$, i.e., finding a point $x \in S$ such that

$$f(x^*) \leq f(x), \quad \forall x \in S$$

where $S \subset R^D$ is a nonempty compact set.

As we already mentioned Global Optimization (GO) methods can be classified into two main categories: *deterministic* and *probabilistic* methods. Most of the deterministic methods involve the application of heuristics, such as modifying the trajectory (trajectory methods) or adding penalties (penalty-based methods), to escape from local minima. On the other hand, probabilistic methods rely on probabilistic judgements to determine whether or not search should depart from the neighborhood of a local minimum. In contrast with different adaptive stochastic search algorithms, Evolutionary Computation (EC) techniques exploit a set of potential solutions, named *population*, and detect the optimal problem solution through cooperation and competition among the individuals of the population. These techniques often find optima in complicated optimization problems faster than traditional optimization methods. The most commonly met population-based EC techniques, such as Evolution Strategies (ES), Genetic Algorithms (GA), Genetic Programming, Evolutionary Programming and Artificial Life methods are inspired from the evolution of nature.

The Particle Swarm Optimization (PSO) method is a member of the wide category of Swarm Intelligence methods (Kennedy and Eberhart, 2001), for solving GO problems. It was originally proposed by J. Kennedy as a simulation of social behavior, and it was initially introduced as an optimization method in 1995 (Eberhart and Kennedy, 1995; Kennedy and Eberhart, 1995). PSO is related with Artificial Life, and specifically to swarming theories, and also with EC, especially ES and GA. PSO can be easily implemented and it is computationally inexpensive, since its memory and CPU speed requirements are low. Moreover, it does not require gradient information of the objective function under consideration, but only its values, and it uses only primitive mathematical operators. PSO has been proved to be an efficient method for many GO problems and in some cases it does not suffer the difficulties encountered by other EC techniques (Eberhart and Kennedy, 1995).

Historical background

The implicit rules adhered to by the members of bird flocks and fish schools, that enable them to move synchronized, without colliding, resulting in an amazing choreography, was studied and simulated by several scientists. In simulations, the movement of the flock was an outcome of the individuals' (birds, fishes etc.) efforts to maintain an optimum distance from their neighboring individuals
The social behavior of animals, and in some cases of humans, is governed by similar rules. There is a general belief, and numerous examples coming from nature enforce the view, that social sharing of information among the individuals of a population, may provide
an evolutionary advantage. This was the core idea behind the development of PSO.

The Particle Swarm Optimization algorithm

PSO's precursor was a simulator of social behavior, that was used to visualize the movement of a birds' flock. Several versions of the simulation model were developed, incorporating concepts such as nearest-neighbor velocity matching and acceleration by distance. When it was realized that the simulation could be used as an optimizer, several parameters were omitted, through a trial and error process resulting in the first simple version of PSO (Eberhart et al., 1996). PSO is similar to EC techniques in that, a population of potential solutions to the problem under consideration, is used to probe the search space. However, in PSO, each individual of the population has an *adaptable velocity* (position change), according to which it moves in the search space. Moreover, each individual has a *memory*, remembering the best position of the search space it has ever visited. Thus, its movement is an aggregated acceleration towards its best previously visited position and towards the best individual of a topological neighborhood. Since the "acceleration" term was mainly used for particle systems in Particle Physics, the pioneers of this technique decided to use the term *particle* for each individual, and the name *swarm* for the population, thus, coming up
with the name *Particle Swarm* for their algorithm. Two variants of the PSO algorithm were developed. One with a global neighborhood, which we will be using for damage identification and one with a local neighborhood. According to the global variant, each particle moves towards its best previous position and towards the best particle in the whole swarm. On the other hand, according to the local variant, each particle moves towards its best previous position and towards the best particle in its restricted neighborhood. In the following paragraphs, the global variant is exposed (the local variant can be easily derived through minor changes).

Suppose that the search space is *D*-dimensional, then the *i*-th particle of the swarm can be represented by a *D*-dimensional vector, $Xi = (xi1, xi2, \ldots, xiD)^T$. The *velocity* (position change) of this particle, can be represented by another *D*-dimensional vector $Vi = (vi1, vi2, \ldots, viD)^T$. The best previously visited position of the *i*-th particle is denoted as $Pi = (pi1, pi2, \ldots, piD)^T$. Defining *g* as the index of the best particle in the swarm (i.e., the *g*-th particle is the best), and let the superscripts denote the iteration number, then the swarm is manipulated according to the following two equations:

$$v_{id}^{n+1} = v_{id}^{n} + cr_1^n (p_{id}^n - x_{id}^n) + cr_2^n (p_{gd}^n - x_{id}^n) \qquad (41)$$

$$x_{id}^{n+1} = x_{id}^{n} + v_{id}^{n+1} \qquad (42)$$

where $d = 1, 2, \ldots, D$; $i = 1, 2, \ldots, N$, and *N* is the size of the swarm; *c* is a positive constant, called *acceleration constant*; $r1$, $r2$ are random numbers,

uniformly distributed in [0, 1]; and n = 1, 2, . . ., determines the iteration number.

The above equations define the initial version of the PSO algorithm. Since there was no actual mechanism for controlling the velocity of a particle, it was necessary to impose a maximum value *Vmax* on it. If the velocity exceeded this threshold, it was set equal to *Vmax*. This parameter proved to be crucial, because large values could result in particles moving past good solutions, while small values could result in insufficient exploration of the search space. This lack of a control mechanism for the velocity resulted in low efficiency for PSO, compared to EC techniques. Specifically, PSO located the area of the optimum faster than EC techniques, but once in the region of the optimum, it could not adjust its velocity stepsize to continue the search at a finer grain. The aforementioned problem was addressed by incorporating a weight parameter for the previous velocity of the particle. Thus, in the latest versions of the PSO, Equations (2) and (3) are changed to the following ones:

$$v_{id}^{n+1} = \chi(wv_{id}^{n} + c_1 r_1^{n}(p_{id}^{n} - x_{id}^{n}) + c_2 r_2^{n}(p_{gd}^{n} - x_{id}^{n})) \qquad (43)$$

$$x_{id}^{n+1} = x_{id}^{n} + v_{id}^{n+1} \qquad (44)$$

where w is called *inertia weight*; $c1$, $c2$ are two positive constants, called *cognitive* and *social* parameter respectively; and χ is a *constriction factor*, which is used, alternatively to w to limit velocity.

The PSO method appears to adhere to the five basic principles of swarm intelligence:

(a) *Proximity*, i.e., the swarm must be able to perform simple space and time computations;
(b) *Quality*, i.e., the swarm should be a able to respond to quality factors in the environment;
(c) *Diverse response*, i.e., the swarm should not commit its activities along excessively narrow channels;
(d) *Stability*, i.e., the swarm should not change its behavior every time the environment alters; and finally
(e) *Adaptability*, i.e., the swarm must be able to change its behavior, when the computational cost is not prohibitive.

Indeed, the swarm in PSO performs space calculations for several time steps. It responds to the quality factors implied by each particle's best position and the best particle in the swarm, allocating the responses in a way that ensures diversity. Moreover, the swarm alters its behavior (state) only when the best particle in the swarm (or in the neighborhood, in the local variant of PSO) changes, thus, it is both adaptive and stable.

The parameters of PSO

The role of the *inertia weight w*, in Equation (4), is considered critical for the PSO's convergence behavior. The inertia weight is employed to control the impact of the previous history of velocities on the current one. Accordingly, the parameter w regulates the trade-off between the global (wide-ranging) and local (nearby) exploration abilities of the swarm. A large inertia weight facilitates global exploration (searching new areas), while a small one tends
to facilitate local exploration, i.e., fine-tuning the current search area. A suitable value for the inertia weight w usually provides balance between global and local exploration abilities and consequently results in a reduction of the number of iterations required to locate the optimum solution. Initially, the inertia weight was constant. However, experimental results indicated that it is better to initially set

the inertia to a large value, in order to promote global exploration of the search space, and gradually decrease it to get more refined solutions. Thus, an initial value around 1.2 and a gradual decline towards 0 can be considered as a good choice for *w*.

The parameters *c*1 and *c*2, in Equation (4), are not critical for PSO's convergence. However, proper fine-tuning may result in faster convergence and alleviation of local minima. As default values, *c*1 = *c*2 = 2 were proposed, but experimental results indicate that *c*1 =*c*2 = 0.5 might provide even better results. Other work reports that it might be even better to choose a larger cognitive parameter, *c*1, than a social parameter,
*c*2, but with *c*1 + *c*2 less than 4.

The parameters *r*1 and *r*2 are used to maintain the diversity of the population and they are uniformly distributed in the range [0, 1]. The constriction factor χ controls on the magnitude of the velocities, in a way similar to the *Vmax* parameter, resulting in a variant of PSO, different from the one with the inertia weight.

PSO for structural damage identification

In the present work we implement PSO algorithm in two cases as already mentioned. Initially for a beam of 22 beam elements and then for a small steel frame building. The procedure is as follows:

a) Finite element model for each structure is assembled, each element stiffness matrix is multiplied by the correspondent damage factor Si (the value is 1 for intact or a fraction for the damaged element)
b) Damage cases are generated by reducing the stiffness of certain elements and the eigenfrequencies of the system are calculated. Essentially this is done by selecting a vector S of damage factors for each case.
c) We introduce the objective function

$$E=\max\left(\left|\frac{\omega_1^{FE}-\omega_1^{EXP}}{\omega_1^{EXP}}\right|,\left|\frac{\omega_2^{FE}-\omega_2^{EXP}}{\omega_2^{EXP}}\right|,\ldots,\left|\frac{\omega_i^{FE}-\omega_i^{EXP}}{\omega_i^{EXP}}\right|,\ldots,\left|\frac{\omega_k^{FE}-\omega_k^{EXP}}{\omega_k^{EXP}}\right|\right) \quad (45)$$

Where ω_i^{FE} is the i-th eigenfrequency produced from the algorithm during the optimization procedure and ω_i^{EXP} the value of the pseudoexperimental i-th eigenfrequency produced at step b.
d) We run PSO algorithm for a predetermined number of iterations or until a convergence criterion is met. The D dimensional vector X is the vector of the damage factors, and D=22 for the beam while D=63 for the building example.

We implement a quadratic programming minimization as described in the second chapter using the fmincon Matlab function, for test purposes. We conclude that quadratic programming minimization can successfully determine the damages when we have one or a few number of small damages and because it's a gradient based method it produces the same result almost every time we apply the method. When the magnitude of damages rises quadprog tends to overestimate their size.

PSO on the other hand is an evolutionary algorithm with a random character, so we can expect to track the minimum of the objective function at some success rate. If our function has many local minima (which is the case for our problems, especially the building), sometimes it can track one of them. In our examples, PSO sometimes misses the exact damage location (compared to Quad Programming) but as the damage magnitude becomes larger (so first order

perturbation theory declines) it predicts with more precision the size and the location of the damage.

We run the algorithm using number of particles from 50-200 (When multiple damages are to be detected, the problem rises in complexicity) and for 20 time steps (iterations). As already mentioned the Vmax term is of great importance, and should be properly calibrated in order to track the solutions with the minimum error.

Implementation of a randomized discrete variable algorithm

As we already mentioned for a fixed number of known eigenvalues , provided that this number is smaller than the number of elements, we expect an infinite number of solutions. So in order to find the solution with the minimum number of damaged elements narrowing the search space, we implemented one other type of algorithm. We considered a predetermined number of damages, equal or less then the number of the measured eigenfrequencies, so the vector of the solution for a member of the population i is Xid where d=[1,2,…,2k] , k being the number of damages assumed.

If q is an odd number that $0<q<2k$ than the values of Xiq are integer numbers that $0<Xiq<$Number of Elements, while for dimension q+1 we have a real value with $0<Xiq+1<1$. The odd dimension indicates which element while the even dimension the size (e.g. the damage factor) at this location. We keep track of the best solutions (a predetermined number of them usually ten). At every iteration one part of the population is updated randomly, and another part by partially updating the dimensions of the best solutions already found. Even if we update only randomly, after enough iterations, we get a picture of which parts of the structure are possible damaged and for which combinations.

References

[1] K.E.Parsopoulos,M.N.Vrahatis,Recent approaches to global optimization problems through Particle Swarm Optimization,Natural Computing 1:235-396,2002

[2] M.M.Saada,M.H.Arafa,A.O.Nashef,Finite element model updating approach to damage identification in beams using particle swarm optimization,Engineering optimization,2013 45 6 677-696

[3] C.Blum,A.Roli Metaheuristics in Combinatorial Optimization:Overview and Conceptual Comparison

Chapter 5
Substructures

In this chapter we give some theoretical background on dynamical reduction of a system using substructuring as a method of approximating eigenvalues through Rayleigh-Ritz analysis.

5.1 Component mode synthesis

Component mode synthesis generally refers to methods of dividing a structure in substructures analyzing each of them separately and then combining the results to synthesize the larger structure. In general, models of structures may be very refined in order to represent complex geometry and to calculate static stresses but on the other hand may be too refined to represent global dynamics. So one reason for applying CMS is for dynamic reduction of the model, another is because it is in a large extent, a natural consequence of the analysis procedure followed in practice when complex and large structures are analyzed. If analyses have been performed for different parts of the structure (many times from different group of analysts) the next natural procedure to use is CMS to acquire an analysis for the whole structure.

Initially one has to assembly the structure from the smaller substructures

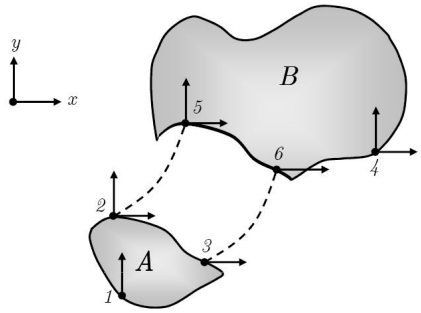


Figure 5.1 Substructures A and B

and form the equations of motion for the combined substructures

$$\begin{cases} M\ddot{u} + C\dot{u} + Ku = f + g \\ Bu = 0 \\ L^T g = 0 \end{cases}$$

Where B is a signed Boolean matrix that enforces the compatibility of the substructures, for the above example that is

$$B = \begin{matrix} u_{1y} & u_{2x} & u_{2y} & u_{3x} & u_{4x} & u_{4y} & u_{5x} & u_{5y} & u_{6x} \end{matrix}$$
$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

While L is the assembly matrix

$$\boldsymbol{u = Lq} = \begin{bmatrix} u_{1y} \\ u_{5x} = u_{2x} \\ u_{5y} = u_{2y} \\ u_{6x} = u_{3x} \\ u_{4x} \\ u_{4y} \\ u_{5x} \\ u_{5y} \\ u_{6x} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_{1y} \\ u_{4x} \\ u_{4y} \\ u_{5x} \\ u_{5y} \\ u_{6x} \end{bmatrix}$$

At the interface of the substructures the force equilibrium must be satisfied

$$\boldsymbol{L^T g} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ g_{2x} \\ g_{2y} \\ g_{3x} \\ 0 \\ 0 \\ g_{5x} \\ g_{5y} \\ g_{6x} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ g_{2x} + g_{5x} \\ g_{2y} + g_{5y} \\ g_{3x} + g_{6x} \end{bmatrix} = \boldsymbol{0}$$

If we assume a unique set of degrees of freedom at the interface, u=Lq , than we have

$$\begin{cases} \boldsymbol{ML\ddot{q} + CL\dot{q} + KLq = f + g} \\ \boldsymbol{L^T g = 0} \end{cases}$$

and premultiplying the first by L^T we have the primal assembly

$$\tilde{M}\ddot{q} + \tilde{C}\dot{q} + \tilde{K}q = \tilde{f} \qquad (46)$$

If we assume equilibrium of forces at the interface $g=-B^T\lambda$ we get the dual assembly

$$\begin{cases} M\ddot{u} + C\dot{u} + Ku + B^T\lambda = f \\ Bu = 0 \end{cases} \qquad (47)$$

The end result should be the same but the dual assembly works even for non matching meshes at the interface.

The most frequently used method (although there are many variations) is the Craig-Bampton method, where the displacements are considered as a linear combination of the modal superposition of the eigenmodes of each substructure for a fixed interface and the static solution when considering displacements on the interface. So for free undamped vibrations for each substructure we have

$$\begin{bmatrix} M_{ii} & M_{ib} \\ M_{bi} & M_{bb} \end{bmatrix}\begin{bmatrix} \ddot{q}_i \\ \ddot{q}_b \end{bmatrix} + \begin{bmatrix} K_{ii} & K_{ib} \\ K_{bi} & K_{bb} \end{bmatrix}\begin{bmatrix} q_i \\ q_b \end{bmatrix} = \begin{bmatrix} o \\ g_b \end{bmatrix}$$

$$M_{ii}\ddot{q}_i + K_{ii}q_i = -M_{ib}\ddot{q}_b - K_{ib}q_b$$

and

$$q_i = -K_{ii}^{-1}K_{ib}q_b + \Phi n$$

So assembling

$$q = \begin{bmatrix} q_b \\ q^{(1)} \\ .. \\ q^{(N_S)} \end{bmatrix} \simeq R_{CB}\begin{bmatrix} q_b \\ n^{(1)} \\ .. \\ n^{(N_S)} \end{bmatrix} = \begin{bmatrix} I & 0 & ... & 0 \\ \Psi^{(1)} & \Phi^{(1)} & & 0 \\ .. & & .. & \\ \Psi^{(N_S)} & 0 & & \Phi^{(N_S)} \end{bmatrix}\begin{bmatrix} q_b \\ n^{(1)} \\ .. \\ n^{(N_S)} \end{bmatrix} \qquad (48)$$

$$\Psi^{(S)} = -K_{ii}^{(S)-1}K_{ib}^{(S)}$$

$$(K_{ii}^{(S)} - \omega_\kappa^{(s)^2}M_{ii}^{(S)})\varphi_{(\kappa)}^{(S)} = 0 \quad , \quad \kappa < N$$

We can form the global stiffness and Mass matrices

$$\tilde{K}_{CB} = R_{CB}^{T} K R_{CB} = \begin{bmatrix} S_{bb} & & & 0 \\ & \Omega^{(1)^2} & & \\ & & \cdots & \\ 0 & & & \Omega^{(N_S)^2} \end{bmatrix} \qquad (49)$$

$$\tilde{M}_{CB} = R_{CB}^{T} M R_{CB} = \begin{bmatrix} M_{bb}^{*} & M_{b\varphi}^{(1)} & \cdots & \mathrm{M}_{b\varphi}^{(\mathrm{N}_S)} \\ M_{\varphi b}^{(1)} & I & & 0 \\ .. & & \cdots & \\ \mathrm{M}_{\varphi b}^{(\mathrm{N}_S)} & 0 & & I \end{bmatrix} \qquad (50)$$

Where S_{bb} , M_{bb}^{*} are assembly of statically condensed matrices.

In practice a small number of modes is used, so we derive to an approximation of the system's eigenvalues and eigenvectors, that is projecting the solution of the original system to a subspace with fewer dimensions and solving the problem there. In general this procedure is known as Ritz analysis.

5.2 Rayleigh-Ritz Analysis

The eigenvalue problem under consideration is $K\varphi = \lambda \mathrm{M} \varphi$ we consider the Rayleigh quotient $\rho(\varphi) = \dfrac{\varphi^{\mathrm{T}} \mathrm{K} \varphi}{\varphi^{\mathrm{T}} \mathrm{M} \varphi}$, the Rayleigh minimum principle states that $\lambda_1 = \min \rho(\varphi)$ and $\lambda_r = \min \rho(\varphi)$ where the minimum is taken over all possible vectors that satisfy the orthogonality condition $\varphi^{\mathrm{T}} \mathrm{M} \varphi_j$, $j = 1, 2, ..., r-1$. In Ritz analysis we consider a set of vectors $\overline{\Phi}$ which are considered a linear combination of the Ritz basis vectors ψ_i , $i = 1, 2, ...q$ so a typical vector is given by

$$\overline{\varphi} = \sum_{i=1}^{q} x_i \psi_i \qquad (51)$$

x_i are the Ritz coordinates. The typical vector here lies in subspace spanned from the Ritz basis vectors (they should be linearly independent) Vq with dimension q while the whole space containing Vq is Vn corresponding to the n dimensional matrices K and M.

We evaluate the Rayleigh quotient and invoke the Rayleigh minimum principle

$$\rho(\bar{\varphi}) = \frac{\sum_{j=1}^{q}\sum_{i=1}^{q} x_i x_j \bar{k}_{ij}}{\sum_{j=1}^{q}\sum_{i=1}^{q} x_i x_j \bar{m}_{ij}} = \frac{\bar{k}}{\bar{m}} \qquad (52)$$

where $\bar{k}_{ij} = \psi_i K \psi_j$

$\bar{m}_{ij} = \psi_i M \psi_j$

the necessary condition for a minimum of $\rho(\bar{\varphi})$ is $\frac{\partial\ \rho(\bar{\varphi})}{\partial x_i} = 0$, for i=1,2,…,q

$$\frac{\partial\ \rho(\bar{\varphi})}{\partial x_i} = \frac{2\bar{m}\sum_{j=1}^{q} x_j \bar{k}_{ij} - 2\bar{k}\sum_{j=1}^{q} x_j \bar{m}_{ij}}{\bar{m}^2} = 0 \qquad (53)$$

Using $\rho = \bar{k} / \bar{m}$ the condition can be stated as

$$\sum_{j=1}^{q} (\bar{k}_{ij} - \rho \bar{m}_{ij}) x_j = 0 \quad , \quad i = 1, 2, ..., q \qquad (54)$$

or in eigenvalue problem form

$$\tilde{K}x = \rho \tilde{\mathrm{M}} x \qquad (55)$$

In practical analysis we can obtain the Ritz basis vectors from a static solution of q load patterns defined in R that is

$$K\Psi = R \quad , \quad \Psi = [\psi_1, ..., \psi_q] \qquad (56)$$

so $$\tilde{K} = \Psi^{\mathrm{T}} \mathrm{K} \Psi \quad , \quad \tilde{\mathrm{M}} = \Psi^{\mathrm{T}} \mathrm{M} \Psi \qquad (57)$$

the solution of the eigenproblem can be written

$$\tilde{K}X = \tilde{\mathrm{M}} X P \qquad (58)$$

where X is the matrix storing the eigenvectors $x_1, x_2, ..., x_q$ of the Ritz coordinates and P the diagonal matrix storing the eigenvalue approximations. The original problem eigenvectors can be approximated by

$$\bar{\Phi} = \Psi \mathrm{X} \qquad (59)$$

Component mode synthesis can be understood as Ritz analysis. In our examples of the beam and the steel frame building we implemented CMS in the calculation of the eigenvalues in order to apply metaheuristic algorithms for damage identification, we considered each time the structure divided in two substructures. Two parts of 11 elements each for the beam and for the frame building as pictured below

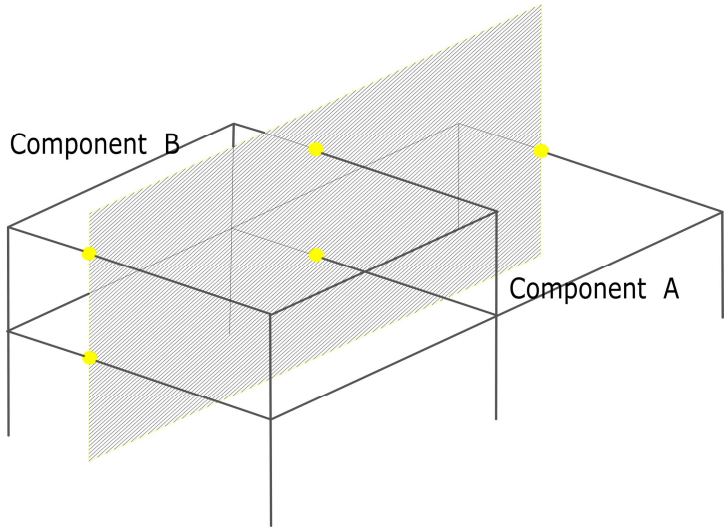


Figure 5.2 Substructures A and B for the frame building example

We considered the substructures fixed at the interface and formed the stiffness K_1,K_2 and mass M_1,M_2 matrices for the component substructures A and B respectively fixed at the interface, the matrices for the whole structure are

$$K = \begin{bmatrix} K_1 & . & \\ . & . & . \\ & . & K_2 \end{bmatrix} \quad , \quad M = \begin{bmatrix} M_1 & . & \\ . & . & . \\ & . & M_2 \end{bmatrix}$$

We solved the eigenvalue problem for each of the substructures

$$K_1\Phi_1 = \Phi_1\Lambda_1 \quad , \quad K_2\Phi_2 = \Phi_2\Lambda_2 \qquad (60)$$

and formed the matrix of load pattern R in order to calculate the Ritz basis vectors

$$R = \begin{bmatrix} \Phi_1 & 0 \\ 0 & \mathrm{I}_{\mathrm{A-B}} \\ \Phi_2 & 0 \end{bmatrix} \qquad (61)$$

where $\mathrm{I}_{\mathrm{A-B}}$ is the unit matrix of order equal to the connection degrees of freedom between component structure A and B. Because the substructures are fixed at the interface, the above mentioned unit matrix releases the fixed degrees of freedom.

The above procedure, as any Ritz analysis, produces an approximation of the sought eigenvalues. The problem of damage identification using eigenfrequencies is actually based on the difference between the damaged and undamaged

eigenvalues and in many cases is very sensitive to this difference, therefore is very important to have good approximations. One way of doing this is by ensuring that at the computation of the Ritz basis vectors all the significant mass degrees of freedom are excited, in our problem for acquiring good approximations and perform damage identification, for the building case, we used some extra load patterns at matrix R. At the end 80-100 degrees of freedom for the building and 50% of them for the beam, could be condensed in order to apply damage identification.

References

[1] K.J.Bathe,Finite Element Procedures
[2] D.Rixen,Dynamic substructuring concepts,Delft University

Chapter 6
Numerical Results

In this chapter we present the numerical results for damage identification of the code implemented in Matlab specifically for:

a. 11 damaged beam cases using Quadratic programming, PSO algorithm and tracking solutions of the same error as Quadprogramming solutions but with fewer damaged elements.

b. 11 cases of a damaged small two storey steel frame building using the proposed Discrete value algorithm

c. The same cases with single or double damage of the beam and building using the Discrete value algorithm and calculating the eigenfrequencies via component mode synthesis method.

6.1 Numerical Results-Beam

We present the results of several damage cases for the 22-element beam. We used as input data the first five (non zero) eigenfrequencies. All the code was implemented in Matlab, for the quadratic programming cases Matlab function fmincon (SQP,Interior Point algorithms) was used. For the PSO algorithm 50-200 particles were used and no more than 30 iterations, the purpose was to identify the damage at a reasonable time (3-9 sec max) at a simple personal computer. The success rate for the PSO was most of the times very high (especially for a small number of damages) but not always (exact success rates were not calculated). We tried initiating PSO with some particle's initial point being the quadratic programming solution , some time the algorithm produced this as the best solution (as it was the one with the smallest error), other than that no other change was detected.

The error for the best solutions ranged from very small order of 1e-8 to order of 1e-02. The order of the error is important as the problem proves to be sensitive in some occasions especially for light damage. In the quadratic programming cases when we had a large amount and/or number of damages, the algorithm tended to give many damaged elements, in these cases we used the procedure described in chapter 3 to track solutions with the same error, the number of damaged elements was decreased.

We should note that because our beam is symmetric, damages at symmetric places produce the same eigenfrequencies. Also, damage at the edges of the beam has little effect on the first five eigenfrequencies.

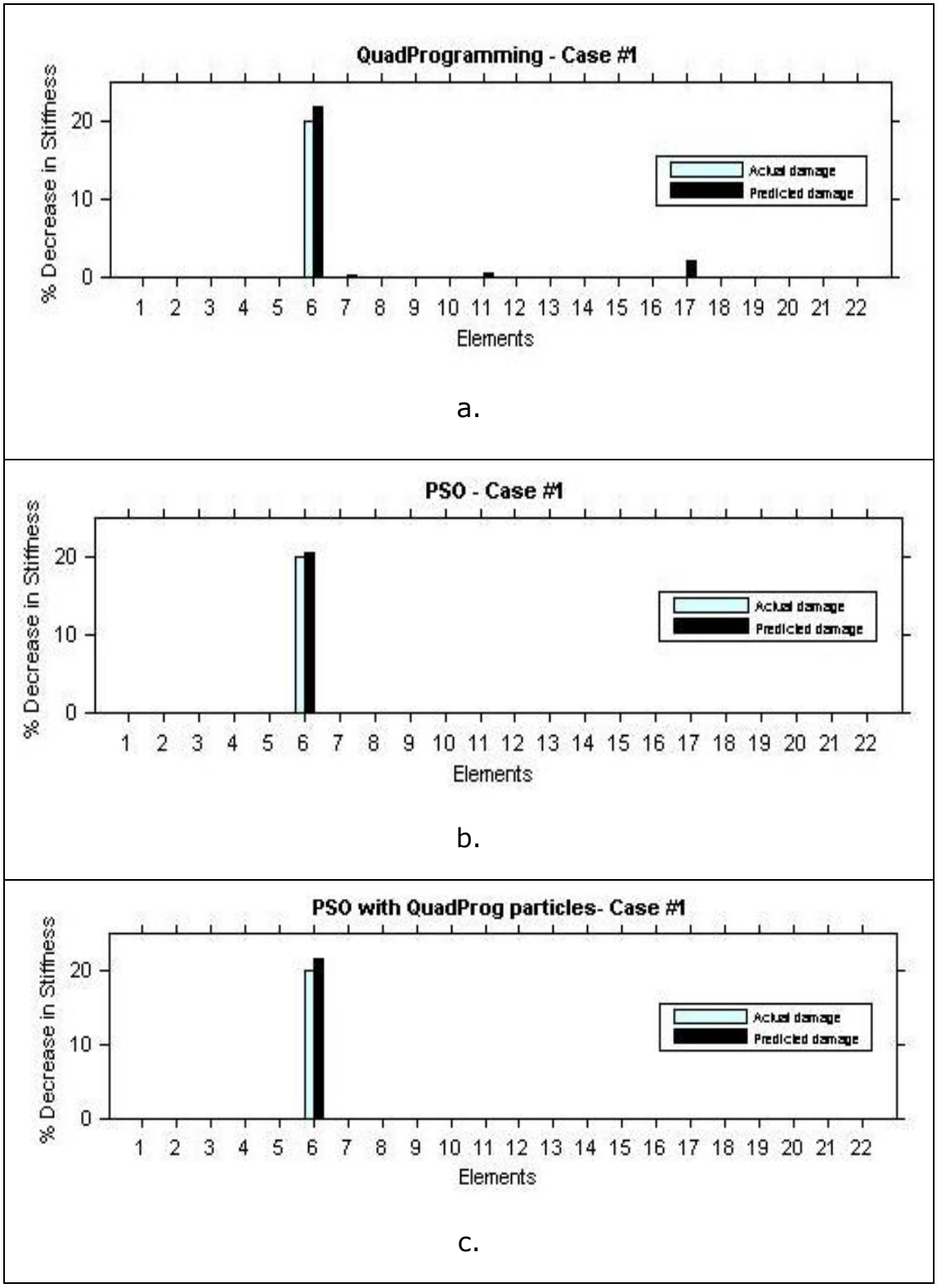


Figure 6.1 Damage identification of case#1 implementing a.Quadratic Programming b. PSO c. PSO with QuadProgramming particles

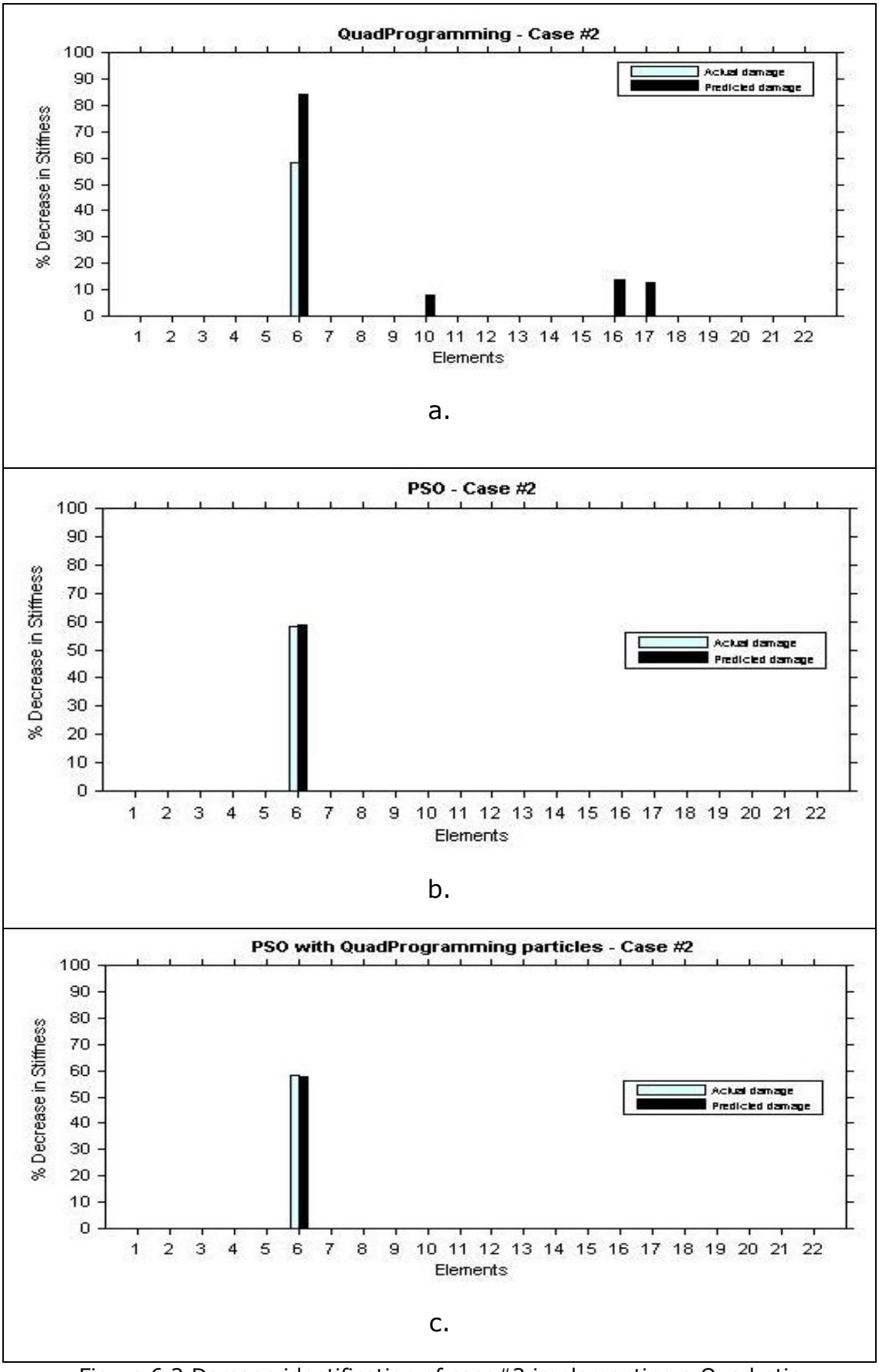


Figure 6.2 Damage identification of case#2 implementing a.Quadratic Programming b. PSO c. PSO with QuadProgramming particles

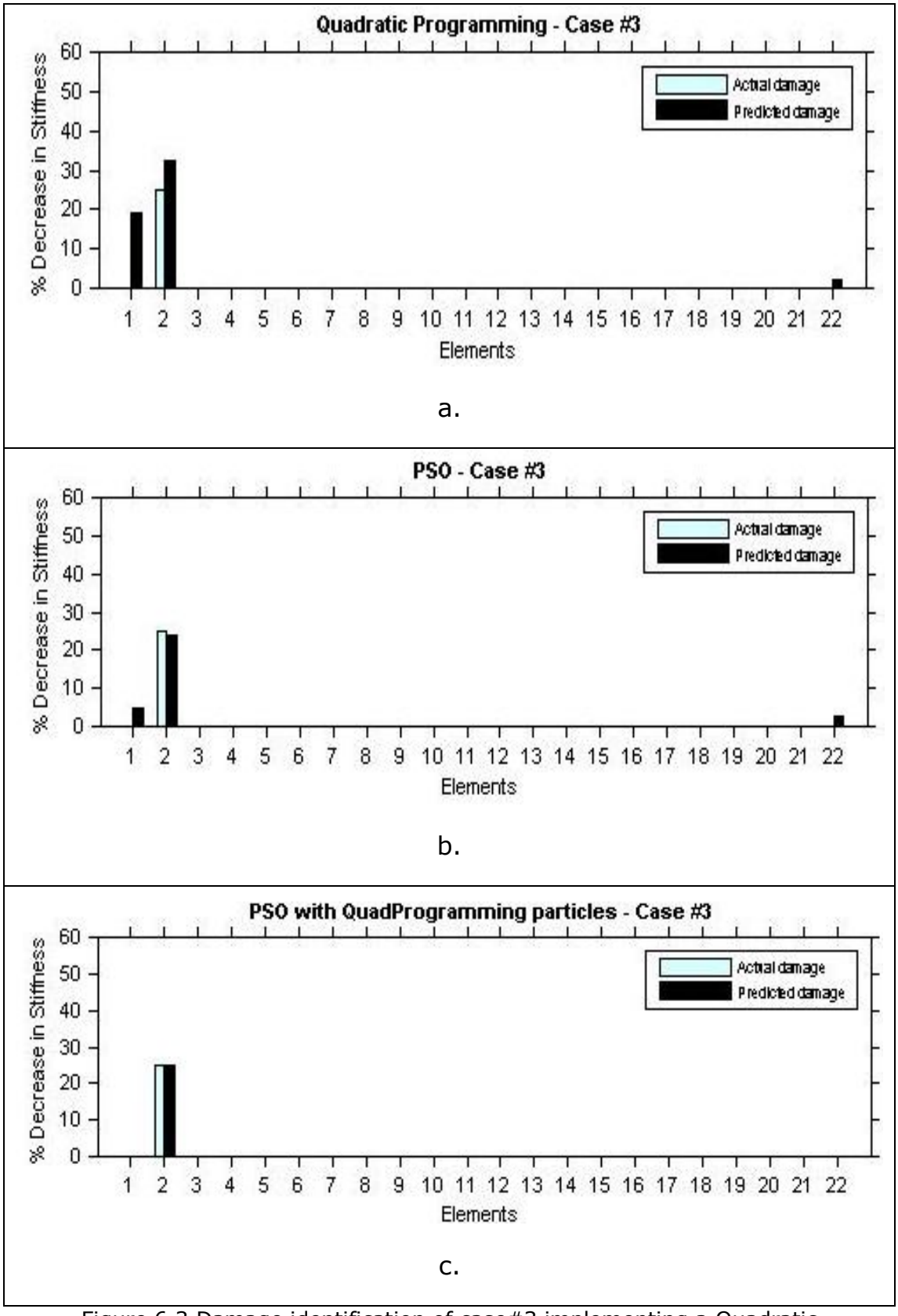


Figure 6.3 Damage identification of case#3 implementing a.Quadratic Programming b. PSO c. PSO with QuadProgramming particles

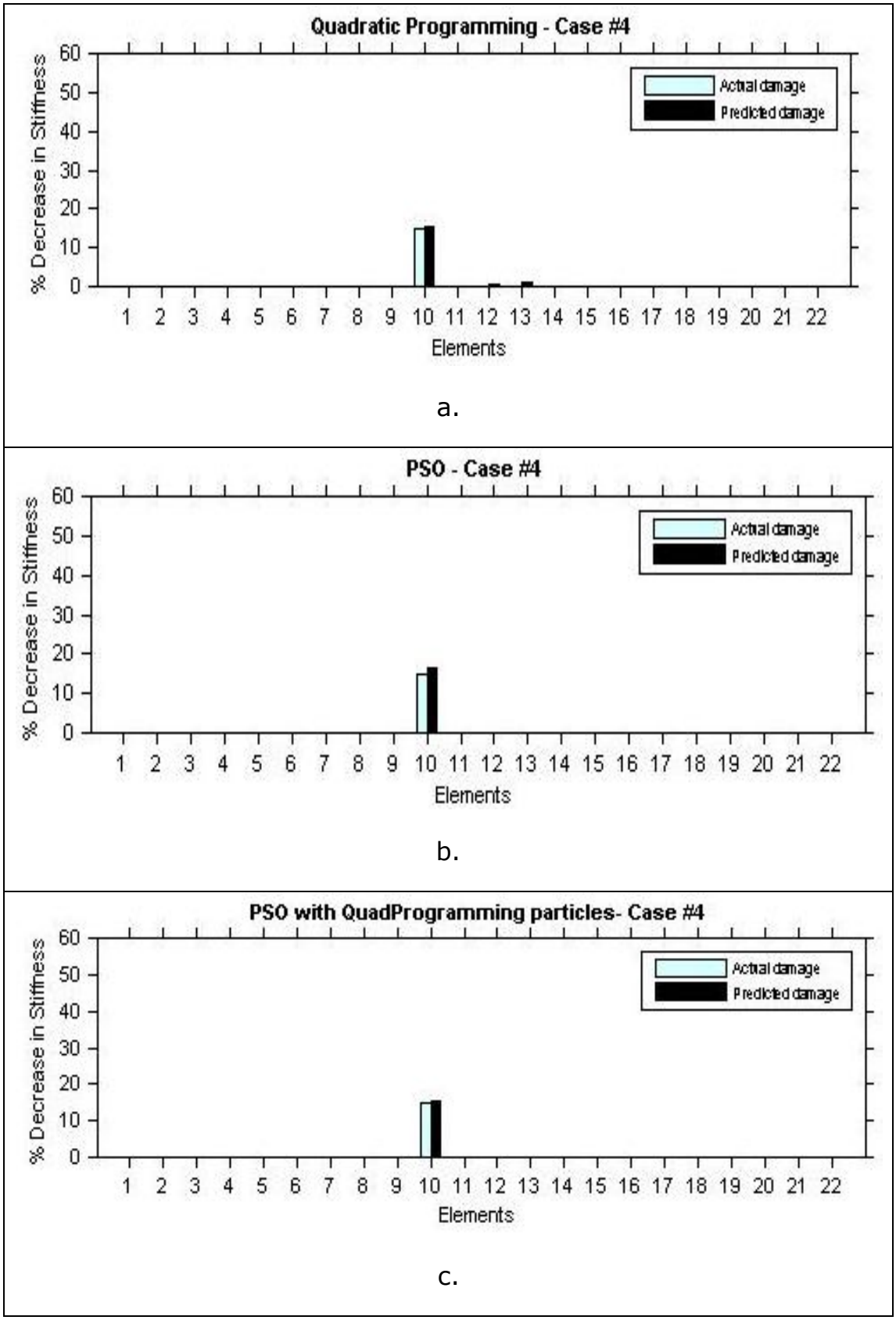


Figure 6.4 Damage identification of case#4 implementing a.Quadratic Programming b. PSO c. PSO with QuadProgramming particles

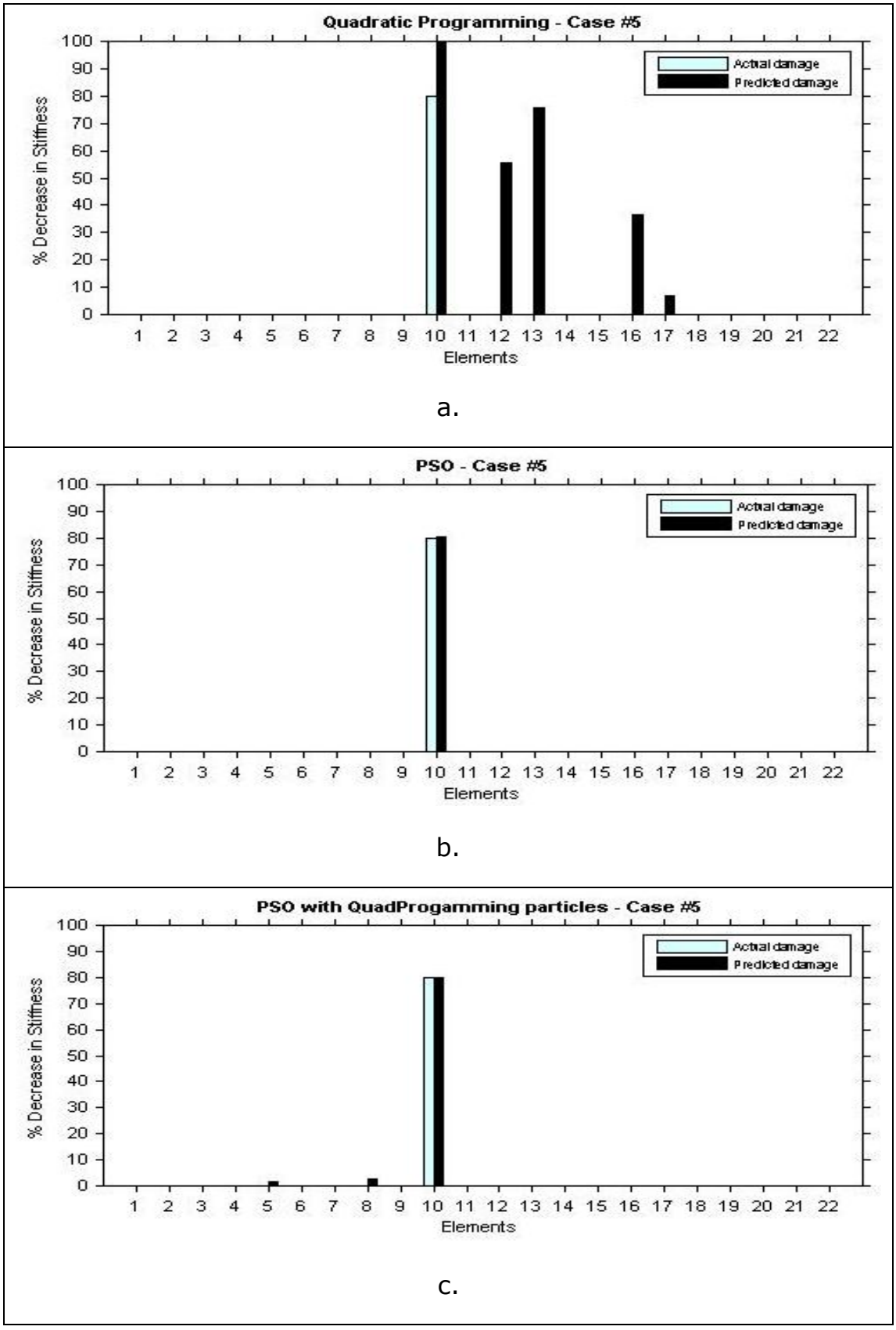


Figure 6.5 Damage identification of case#5 implementing a.Quadratic Programming b. PSO c. PSO with QuadProgramming particles

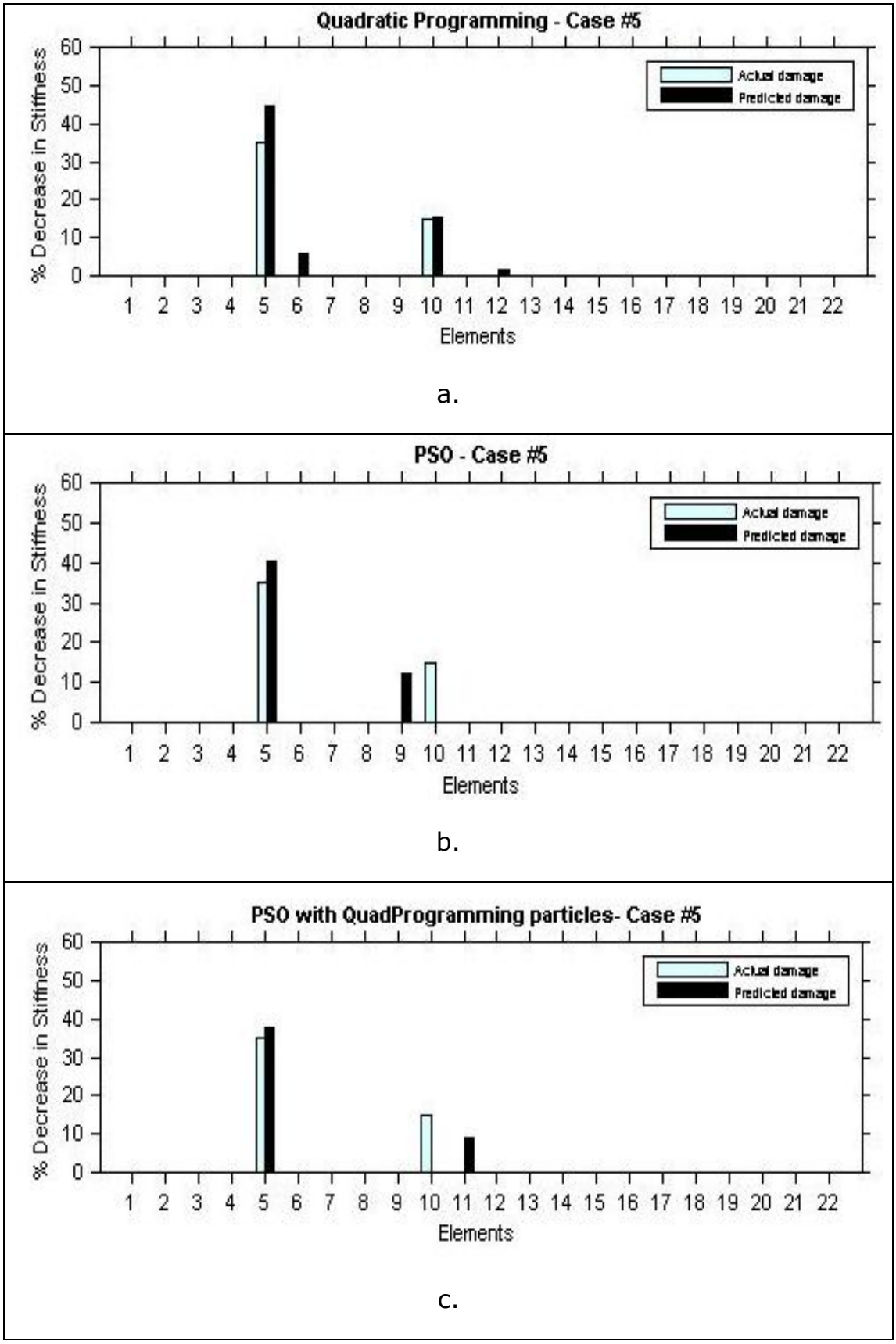
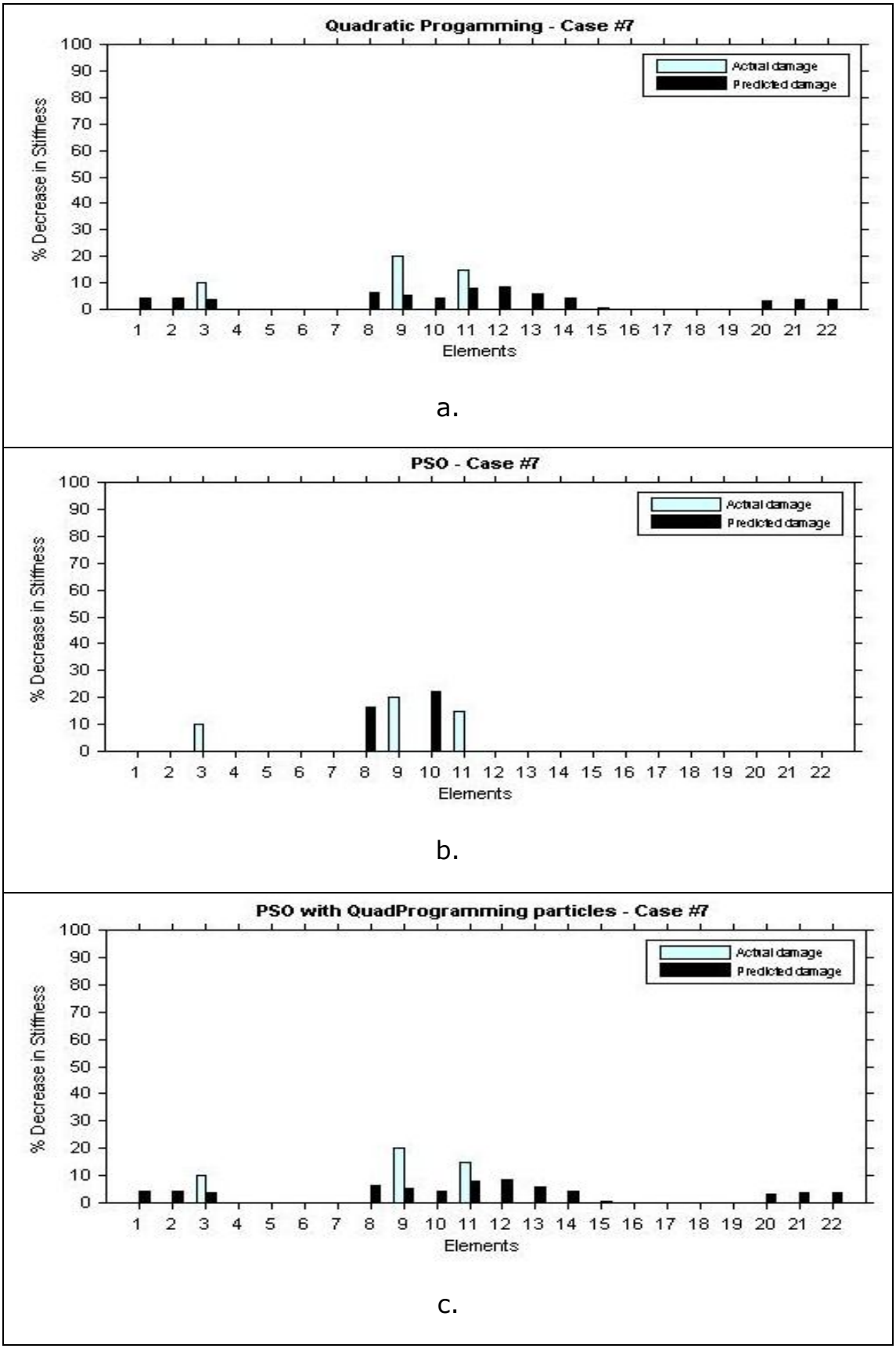
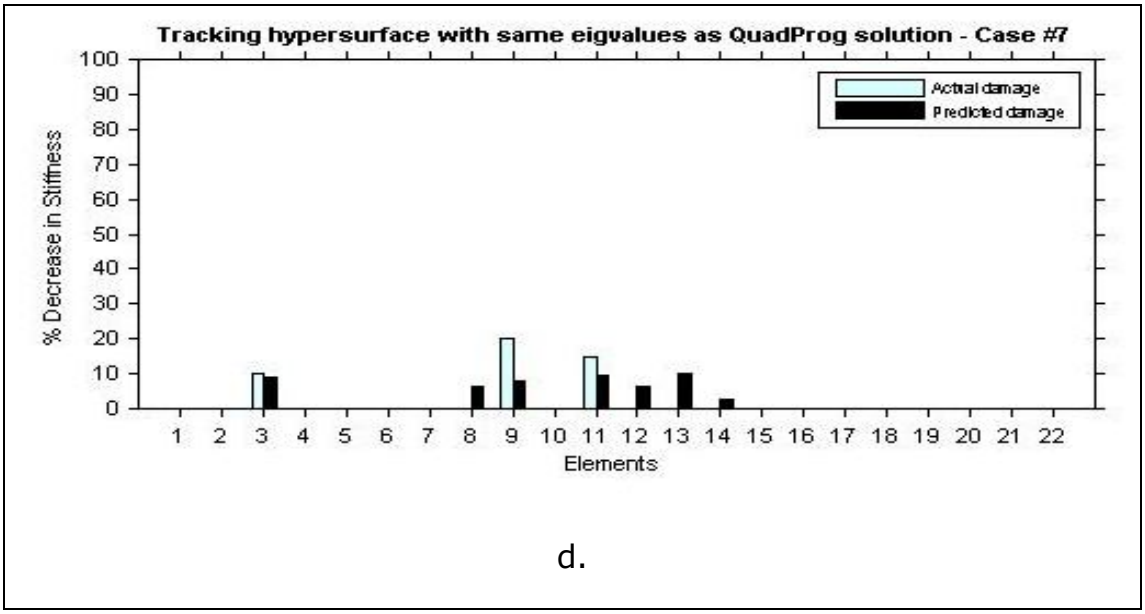


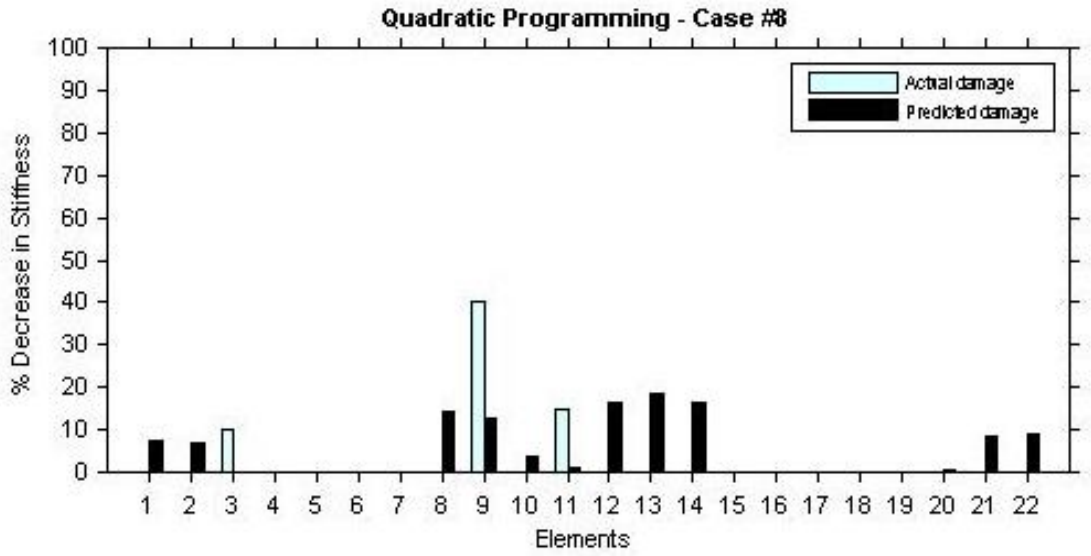
Figure 6.6 Damage identification of case#6 implementing a.Quadratic Programming b. PSO c. PSO with QuadProgramming particles


Quadratic Progamming - Case #7
Actual damage
Predicted damage
% Decrease in Stiffness
Elements
a.
PSO - Case #7
Actual damage
Predicted damage
% Decrease in Stiffness
Elements
b.
PSO with QuadProgramming particles - Case #7
Actual damage
Predicted damage
% Decrease in Stiffness
Elements
c.

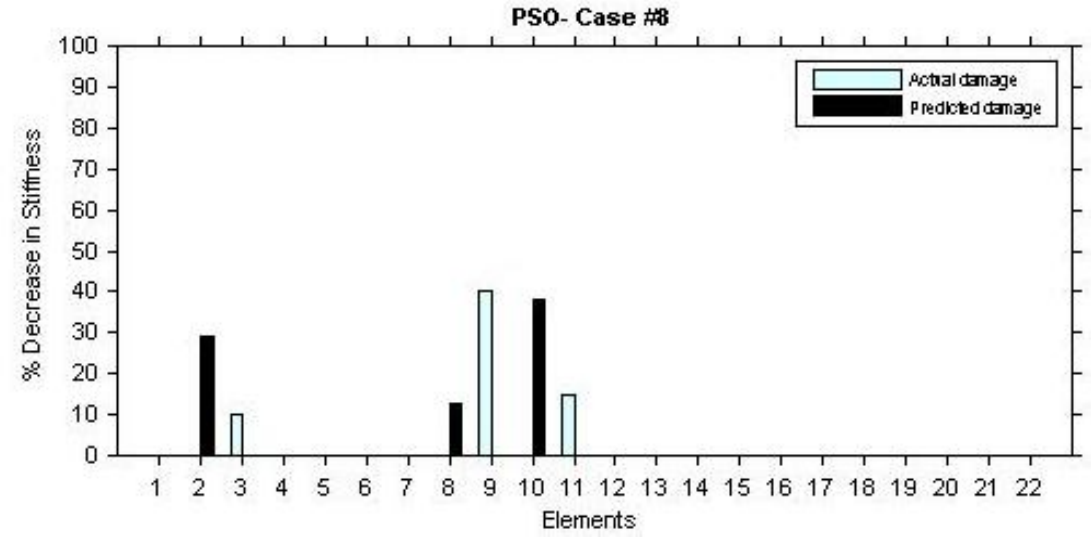


d.

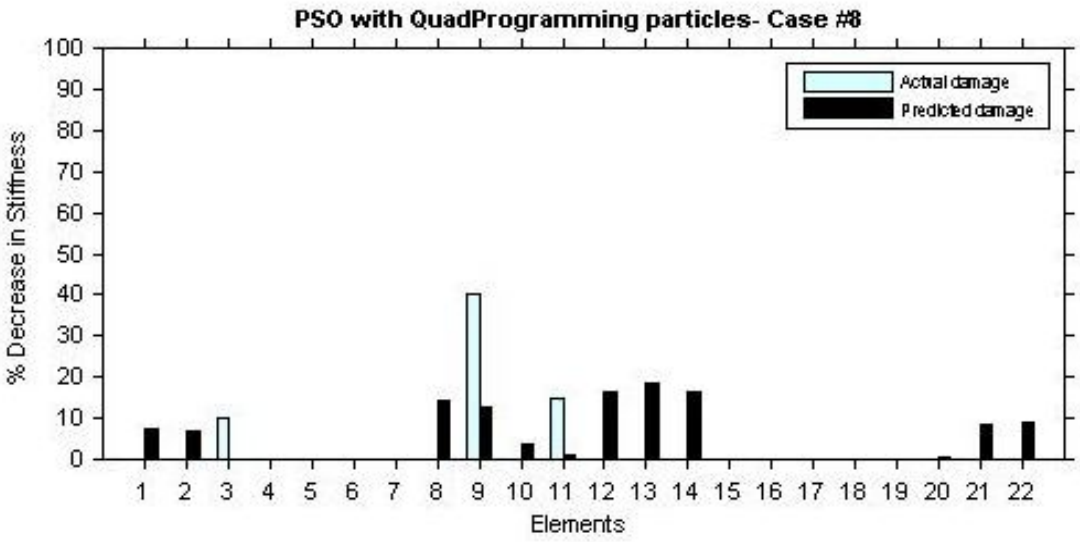
Figure 6.7 Damage identification of case#7 implementing a.Quadratic Programming b. PSO c. PSO with QuadProgramming particles d. Tracking intersection of hypersurfaces with same error as QuadProgramming solution


Quadratic Programming - Case #8
Actual damage
Predicted damage
% Decrease in Stiffness
Elements

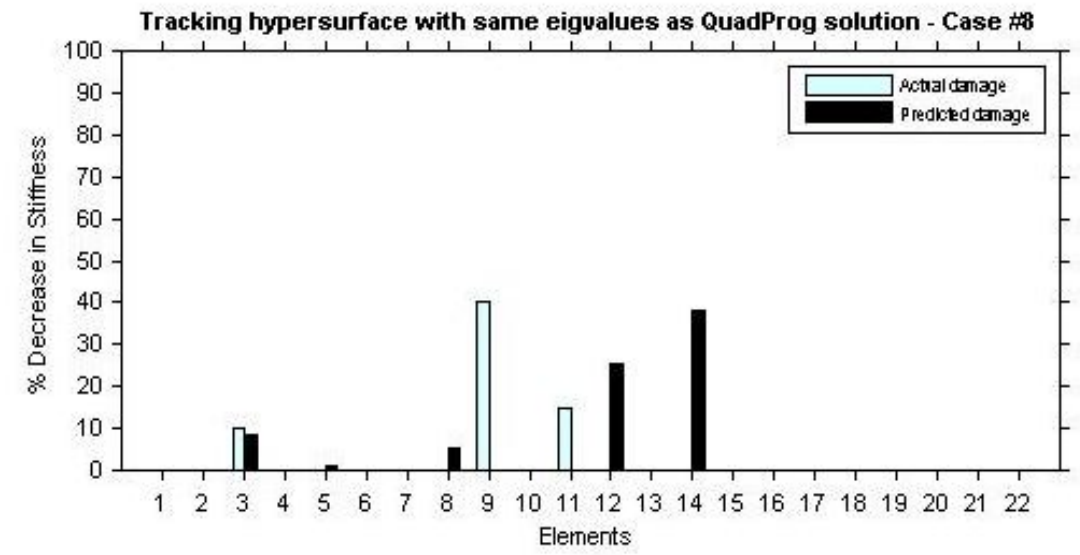
a.


PSO- Case #8
Actual damage
Predicted damage
% Decrease in Stiffness
Elements

b.



c.



d.

Figure 6.8 Damage identification of case#8 implementing a.Quadratic Programming b. PSO c. PSO with QuadProgramming particles d. Tracking intersection of hypersurfaces with same error as QuadProgramming solution

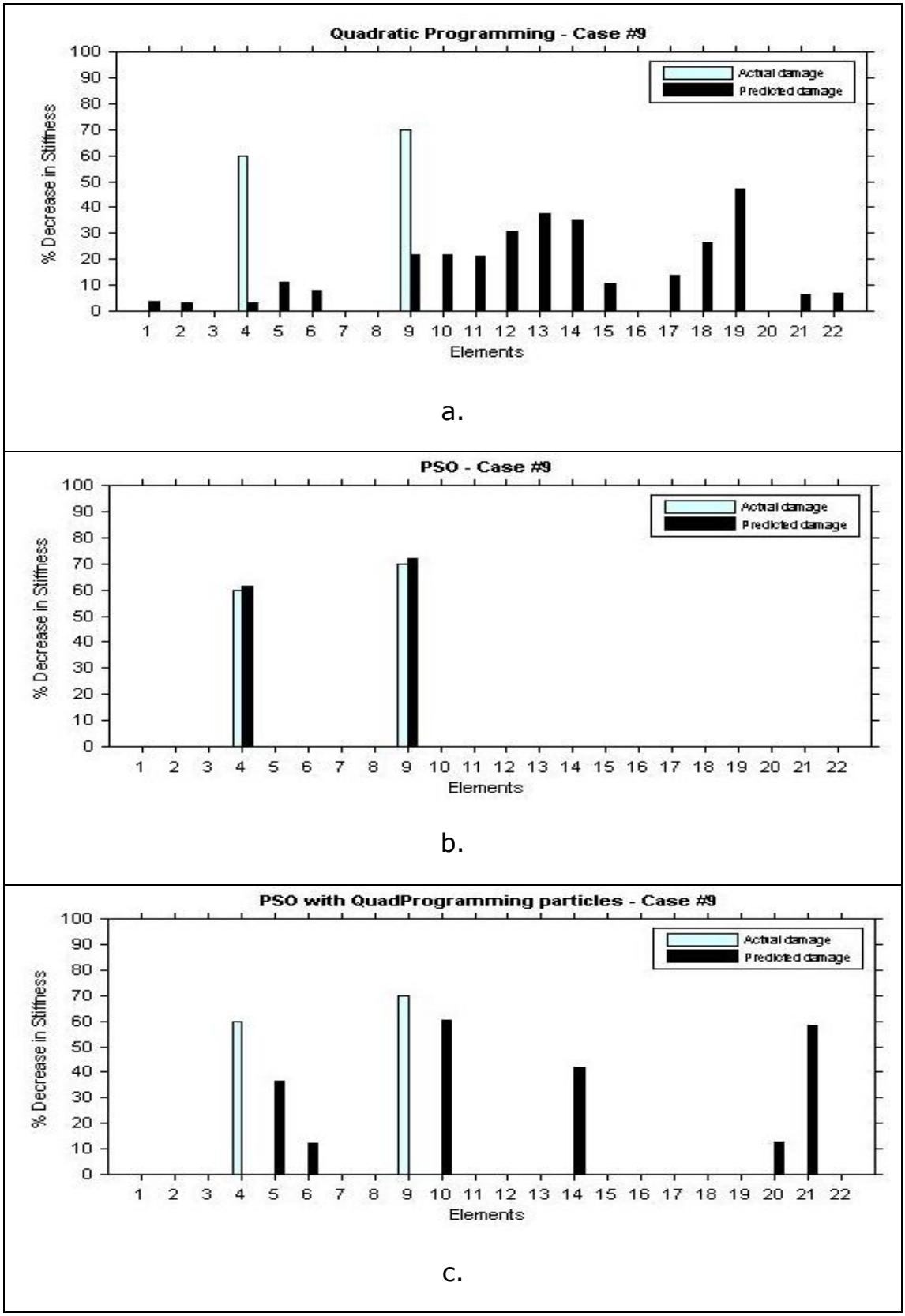


Figure 6.9 Damage identification of case#9 implementing a.Quadratic Programming b. PSO c. PSO with QuadProgramming particles

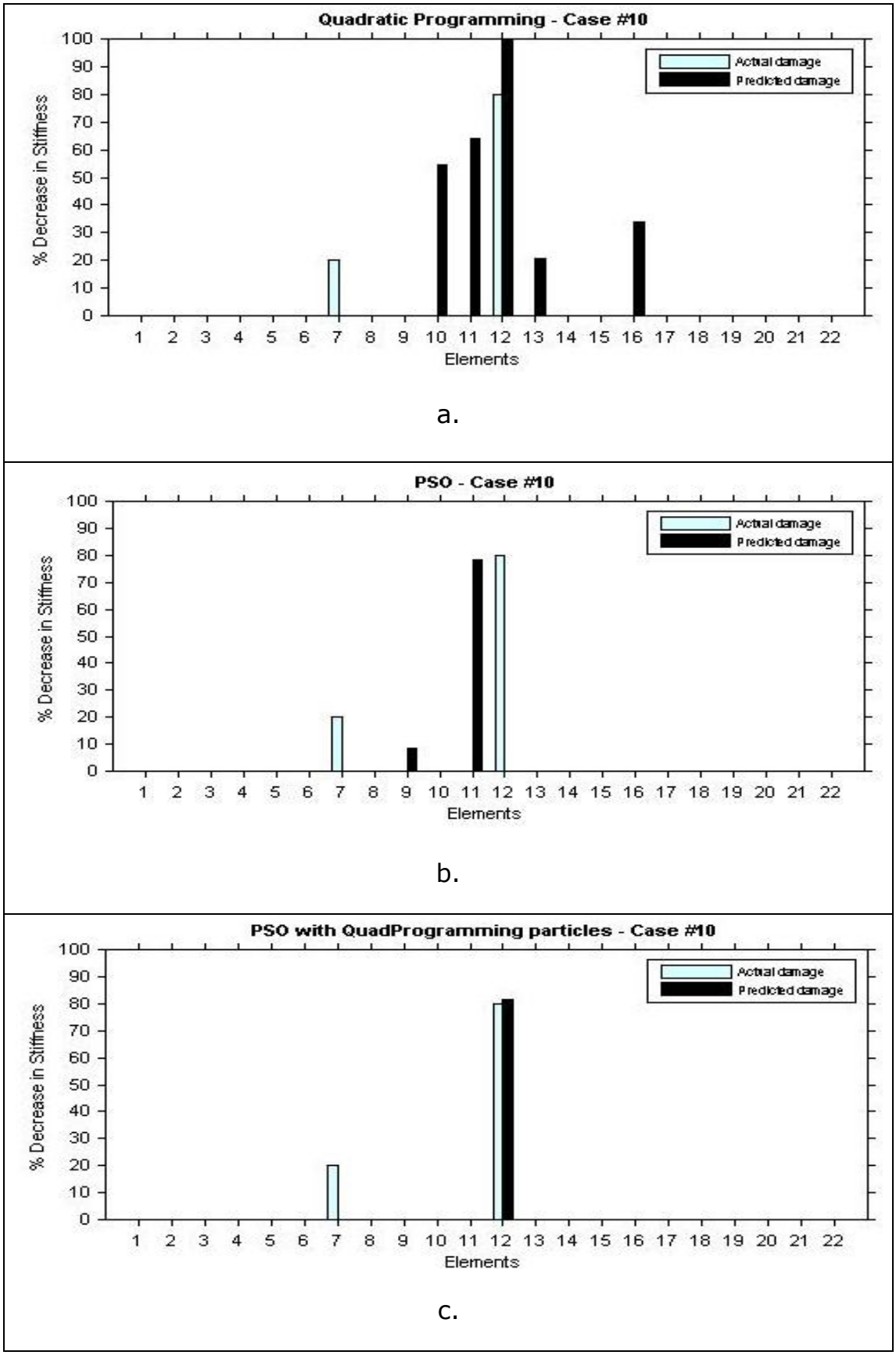
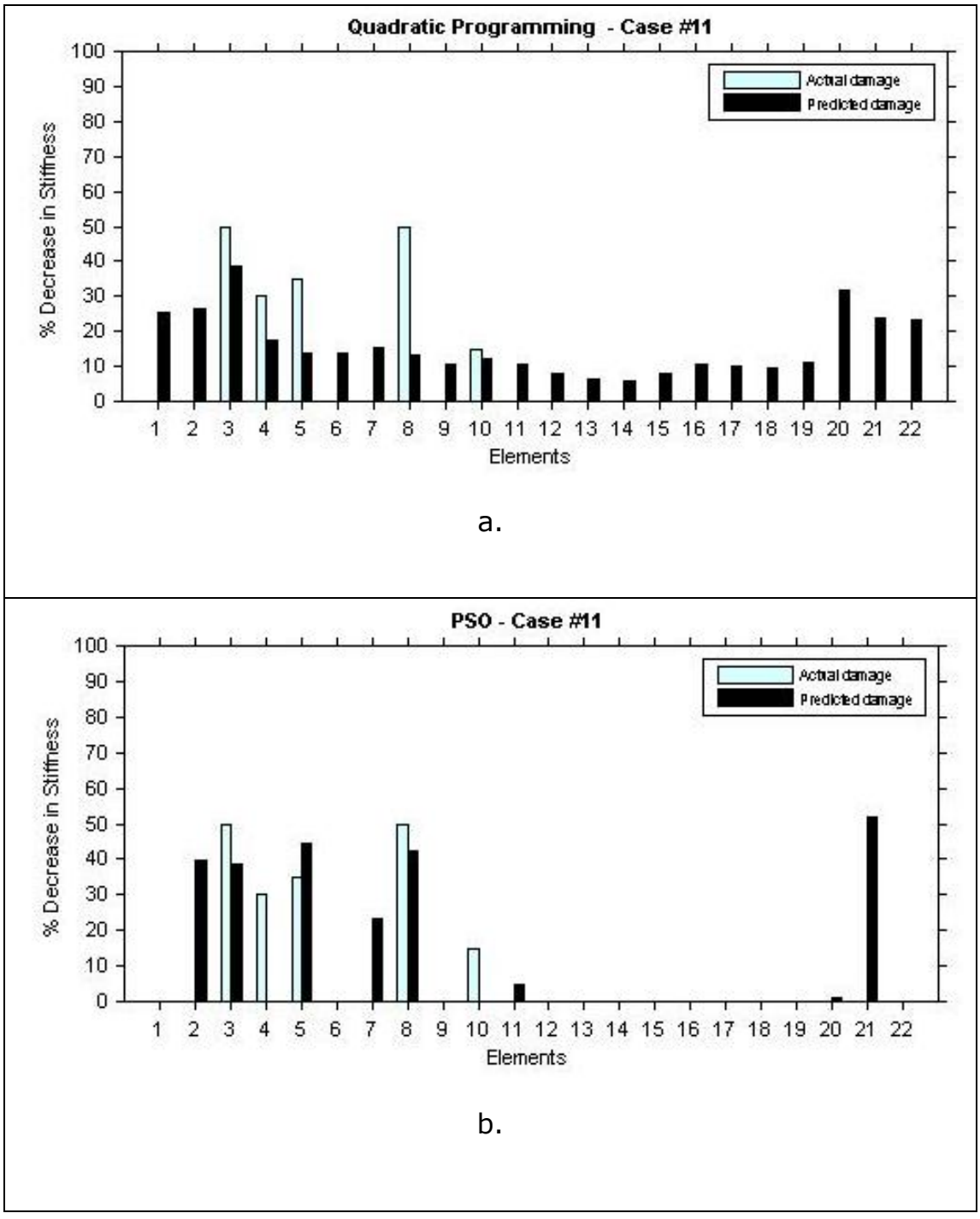
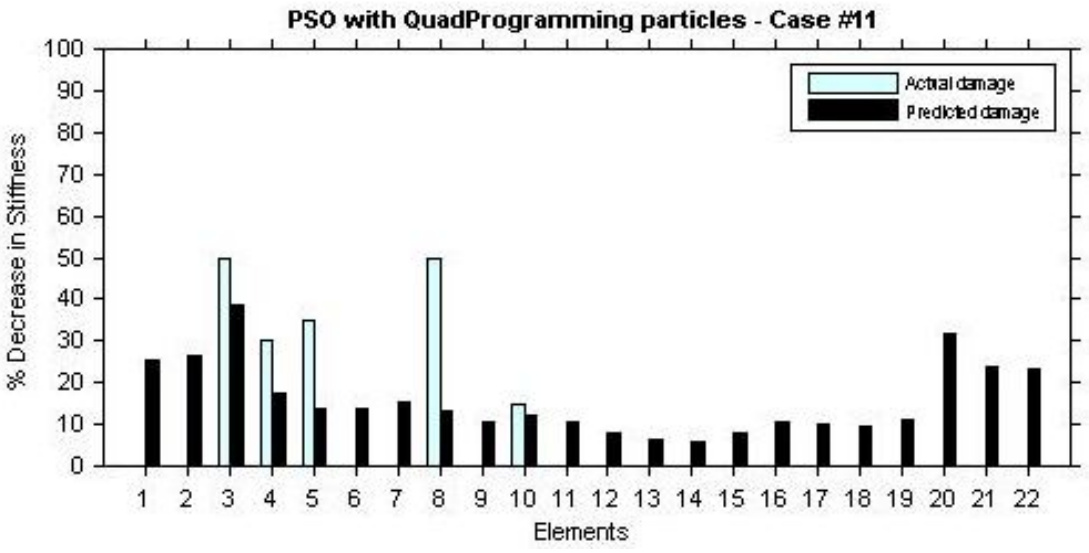
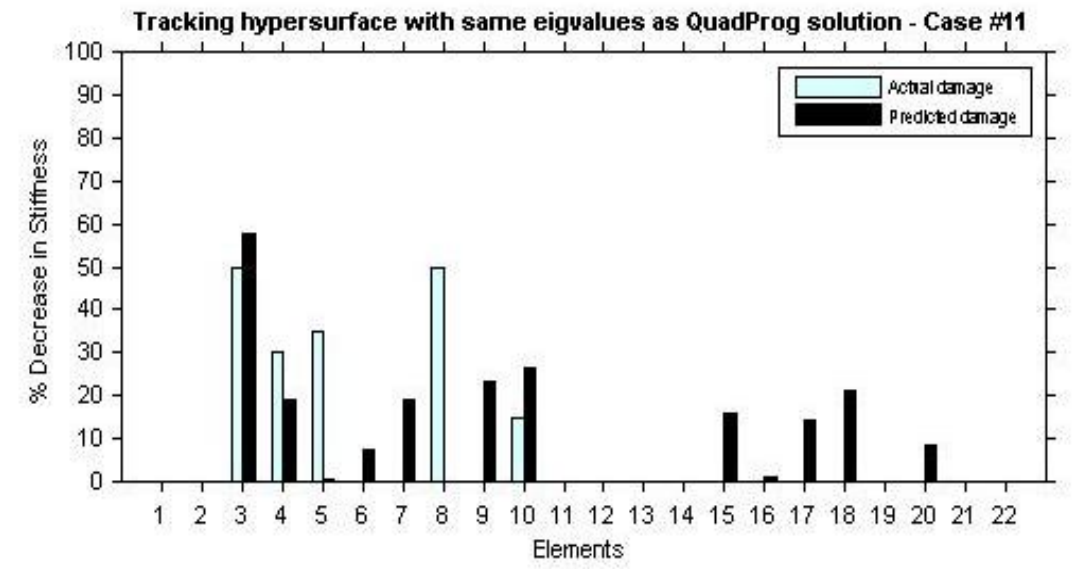


Figure 6.10 Damage identification of case#10 implementing a.Quadratic Programming b. PSO c. PSO with QuadProgramming particles


Quadratic Programming - Case #11
% Decrease in Stiffness
Elements
Actual damage
Predicted damage
a.
PSO - Case #11
% Decrease in Stiffness
Elements
Actual damage
Predicted damage
b.



c.



d.

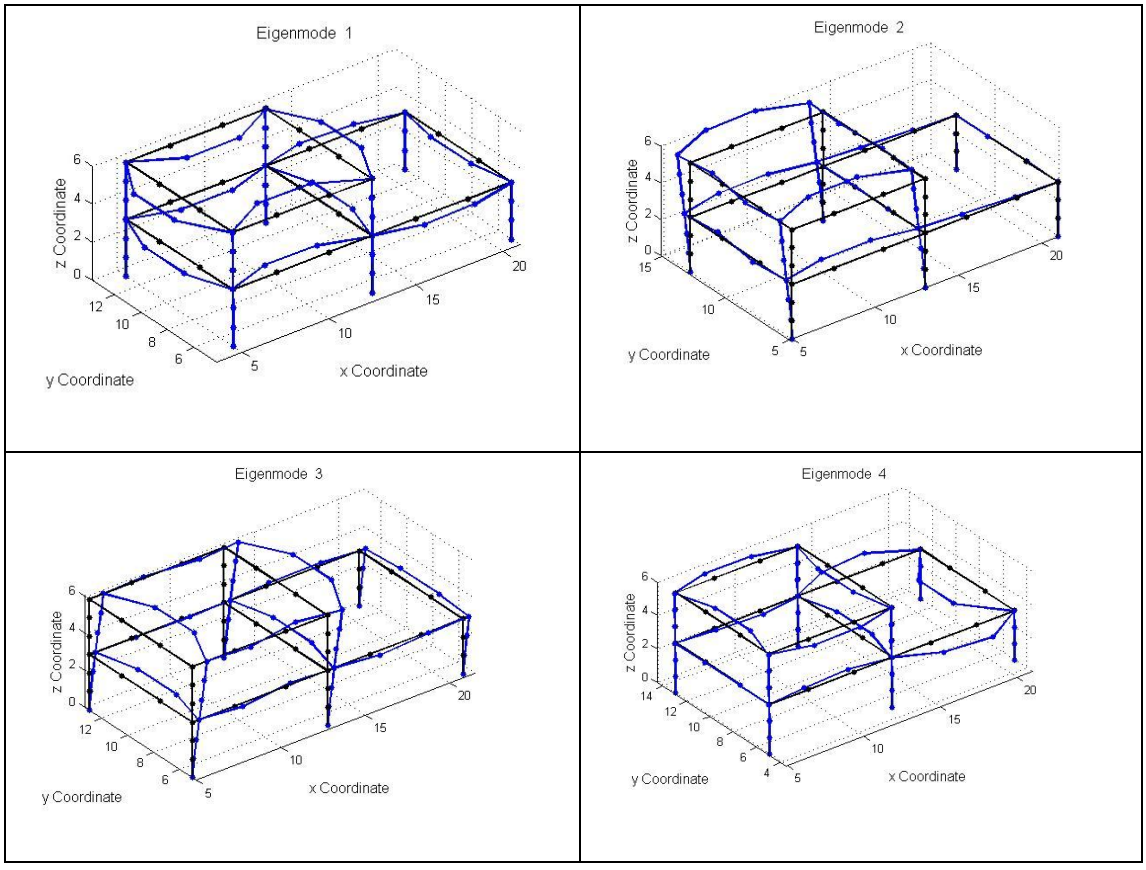
Figure 6.11 Damage identification of case#11 implementing a.Quadratic Programming b. PSO c. PSO with QuadProgramming particles d. Tracking intersection of hypersurfaces with same error as QuadProgramming solution

6.2 Numerical Results-Steel frame building

We implemented metaheuristic algorithms for damage identification at the building example. We used as input data the first ten eigenfrequencies. The eigenfrequencies for the undamaged case are presented below along with the first ten eigenmodes

No	Hz
1	2,472
2	2,594
3	2,785
4	2,987
5	3,200
6	3,309
7	3,577
8	3,577
9	3,735
10	3,946

Table 6.12 First ten eigenfrequencies of the building example



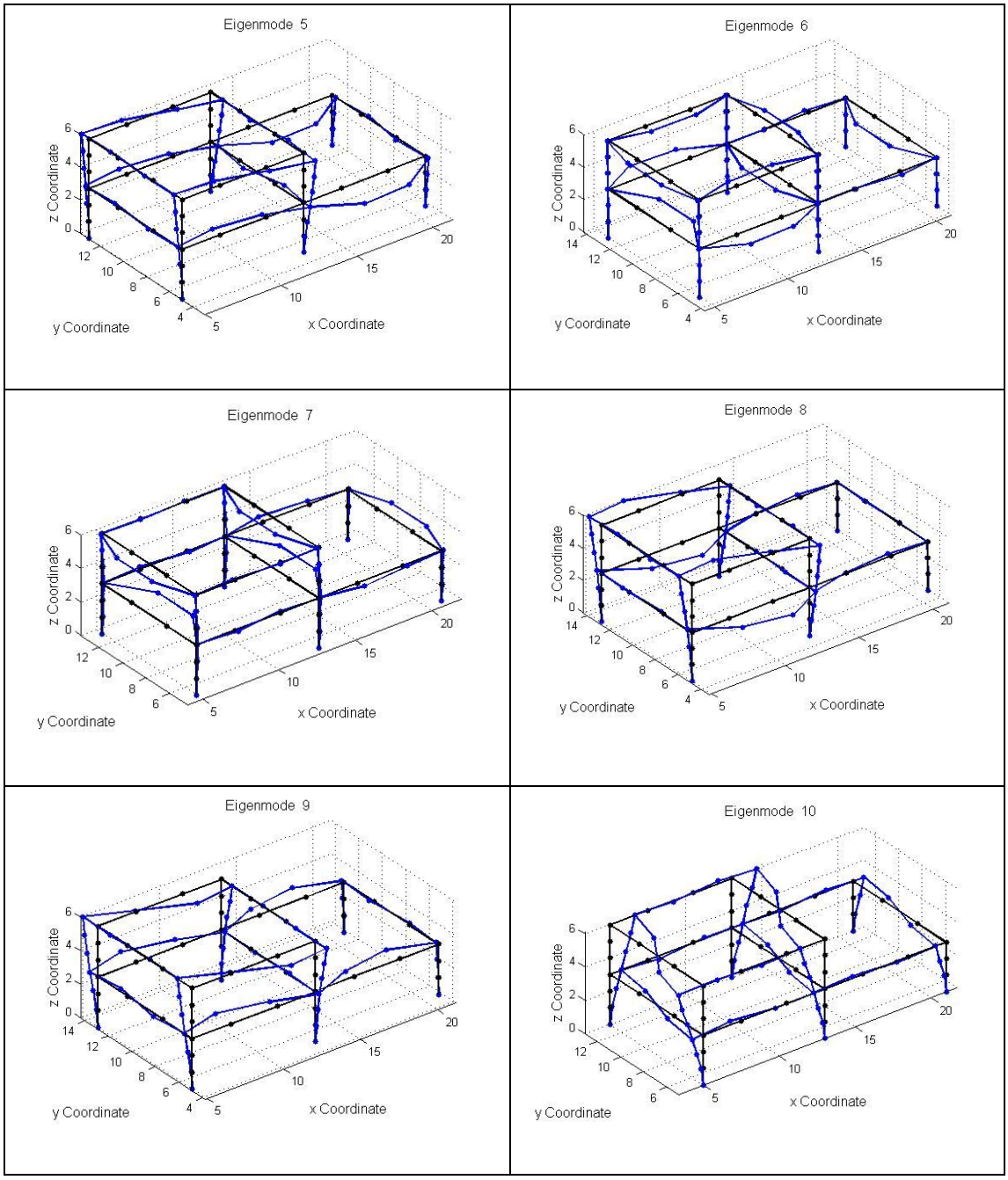


Figure 6.13 First ten eigenmodes of the building example

The spectrum is narrow and we even have eigenfrequencies with almost the same value (7 and 8), this is one of the reasons that this example was more sensitive than the beam and appeared to have more local minima for the algorithm's objective function. Quadratic programming was tried but did not produce satisfactory results, the same is true for PSO as implemented before, many times though it approached the damaged scenario but needed a lot of computational time. In order to reduce the search space we implemented a randomized discrete variable algorithm as described in chapter 4. At every iteration we calculate the error of a predetermined number of solutions (usually 50-100) and keep record of the best solutions found through the search (usually 10). At every iteration the candidate solutions are updated using a distribution, that is 50% of them are randomly updated while the rest of them came up from

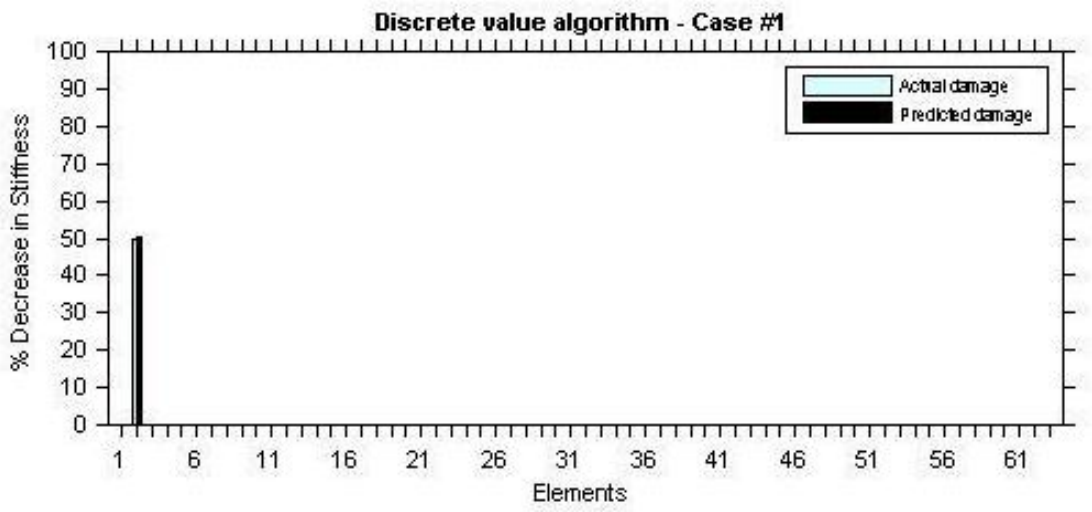
randomly updating some dimensions of the best solutions already found (the possibility of this depends on the rating of each best solution) . The i-th solution vector for k number of damages is

$$x_i = [x_{i1} \;\; x_{i2} \;\; \ldots \; x_{i(2(j-1)+1)} \;\; x_{i(2j)} \;\; \ldots \;\; x_{i(2k)}]^T \qquad (62)$$

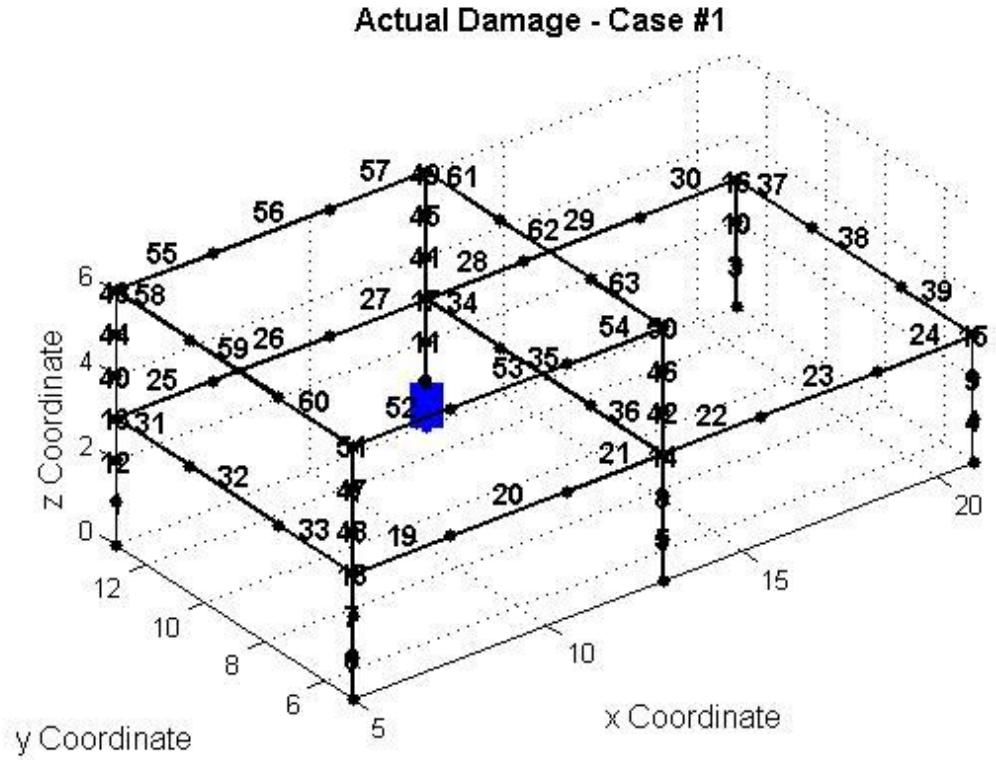
$$1 \le j \le k \;\; , \;\; 1 \le x_{i(2(j-1)+1)} \le 63 \;\; , \;\; x_{i(2(j-1)+1)} \in \mathbb{N}$$

$$0 \le x_{i(2j)} \le 1 \; , \qquad x_{i(2j)} \in \mathbb{R}$$

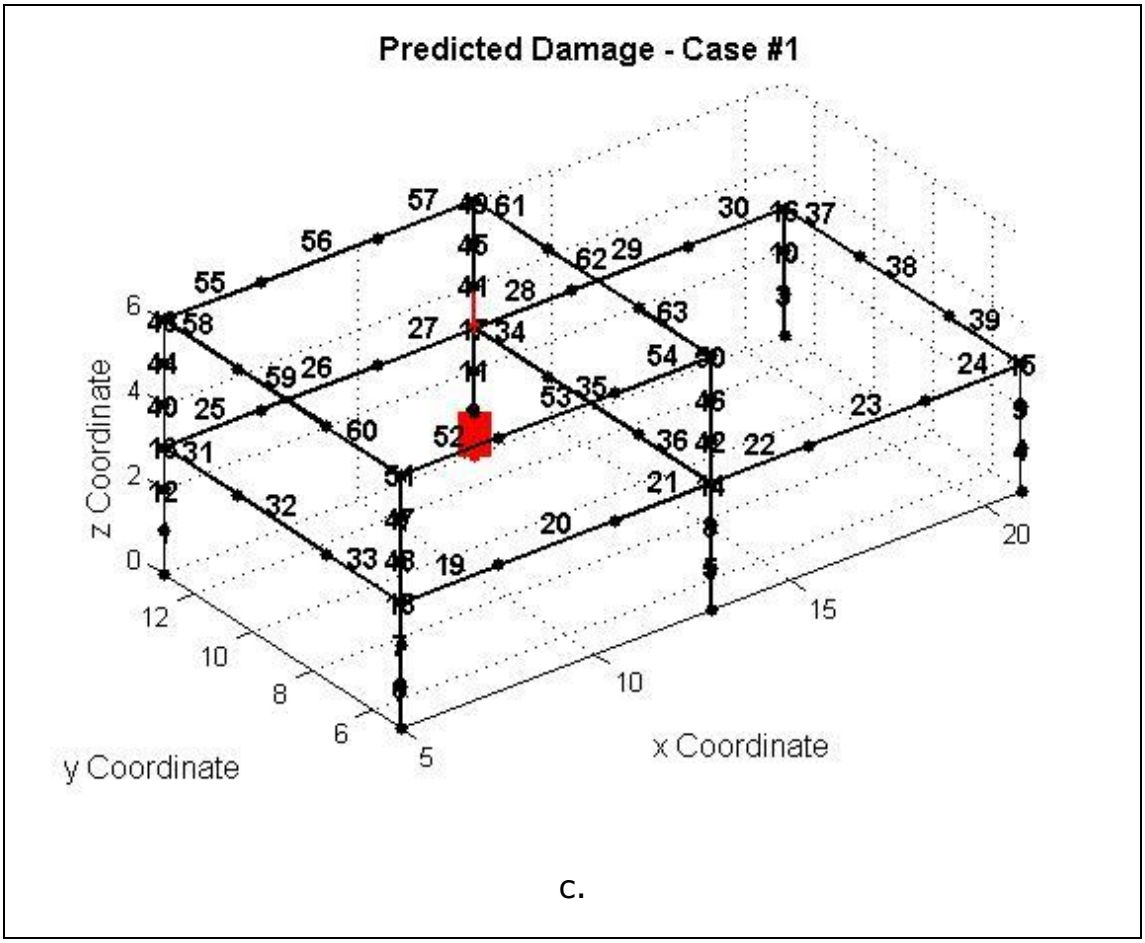
The odd dimension represents the damaged element while the even it's damage factor. We run the above algorithm scanning for two possible damages for a predetermined number of times and at the cases examined the success rate was very high (detecting the case or the symmetric scenario) below we present the results.


Discrete value algorithm - Case #1
Actual damage
Predicted damage
% Decrease in Stiffness
Elements
0
10
20
30
40
50
60
70
80
90
100
1
6
11
16
21
26
31
36
41
46
51
56
61

a.

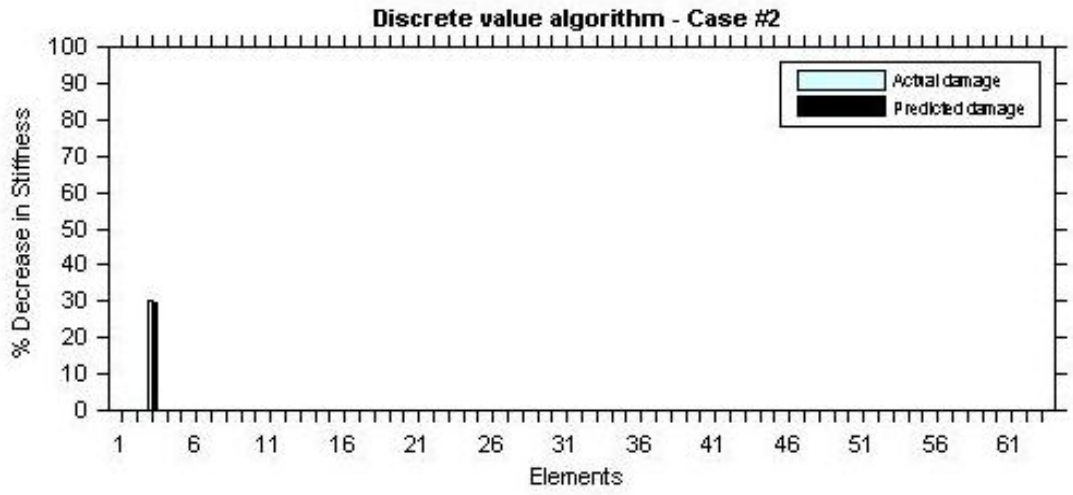

Actual Damage - Case #1
z Coordinate
y Coordinate
x Coordinate
0
2
4
6
12
10
8
6
5
10
15
20

b.

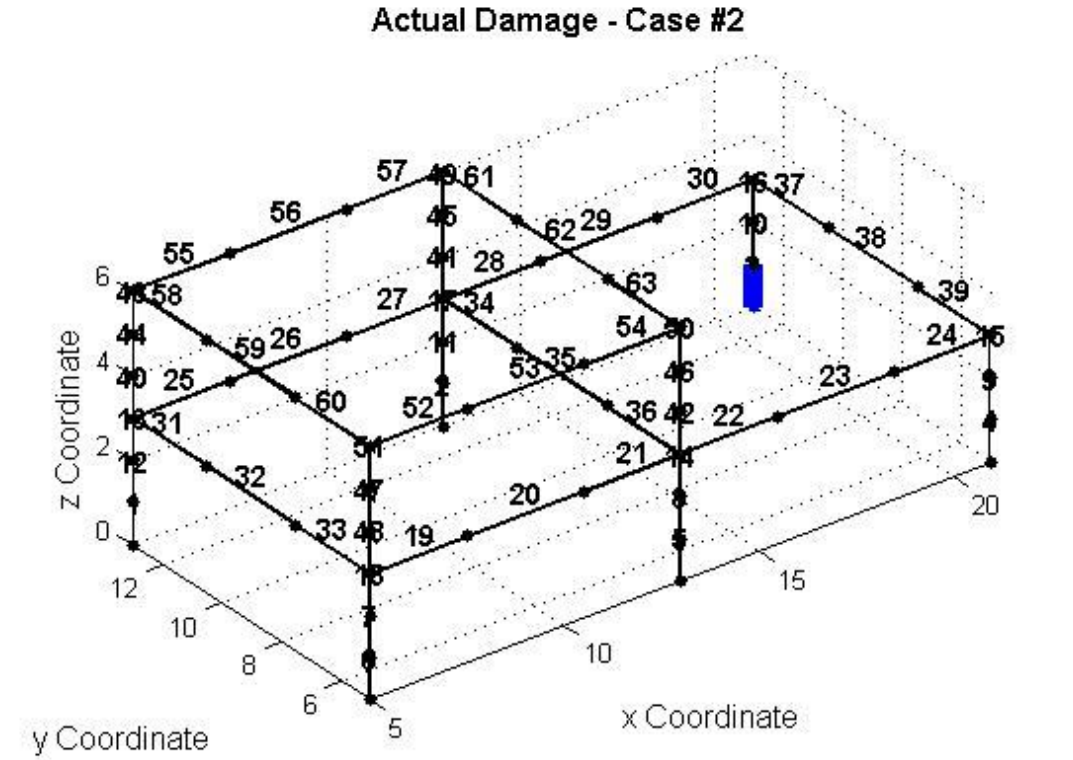


c.

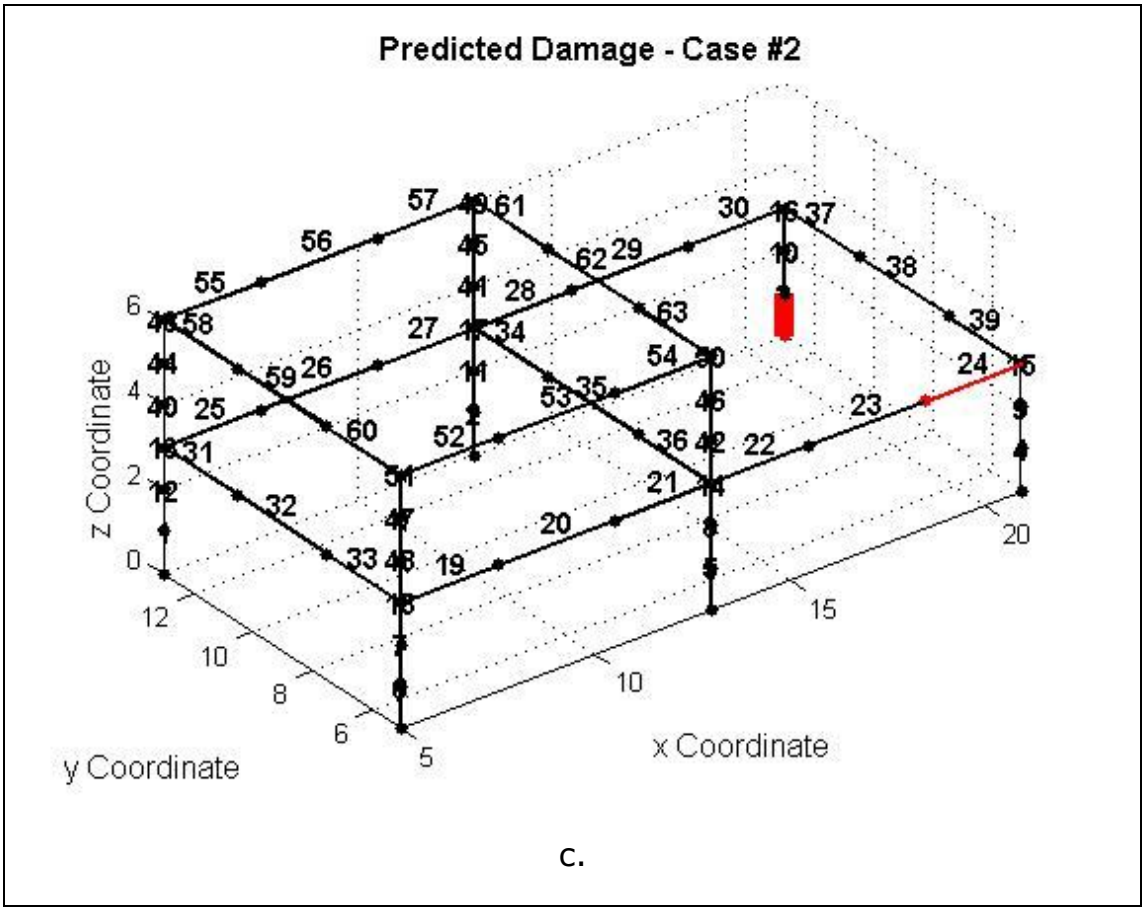
Figure 6.14 a. Damage Identification of case#1 b. Actual damage c. Predicted damage


Discrete value algorithm - Case #2
Actual damage
Predicted damage
% Decrease in Stiffness
Elements

a.

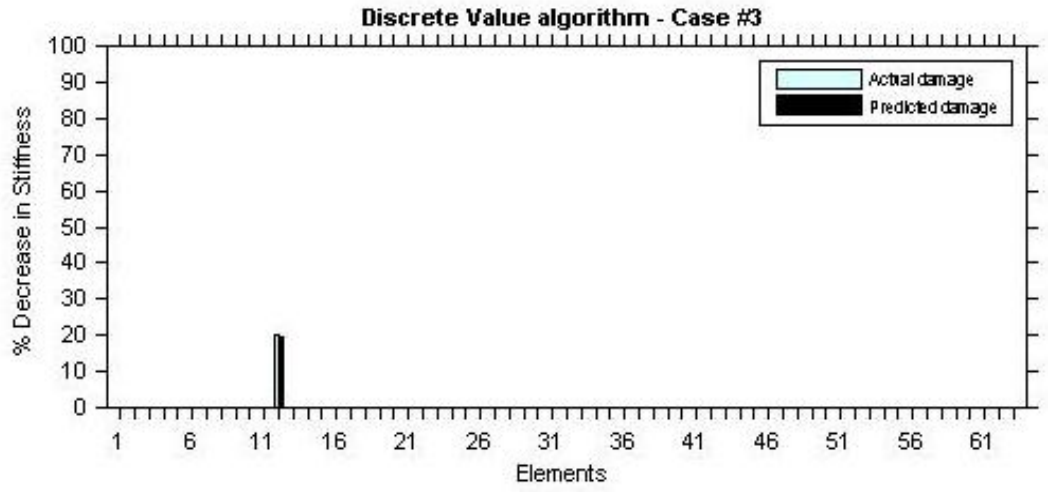

Actual Damage - Case #2
z Coordinate
y Coordinate
x Coordinate

b.

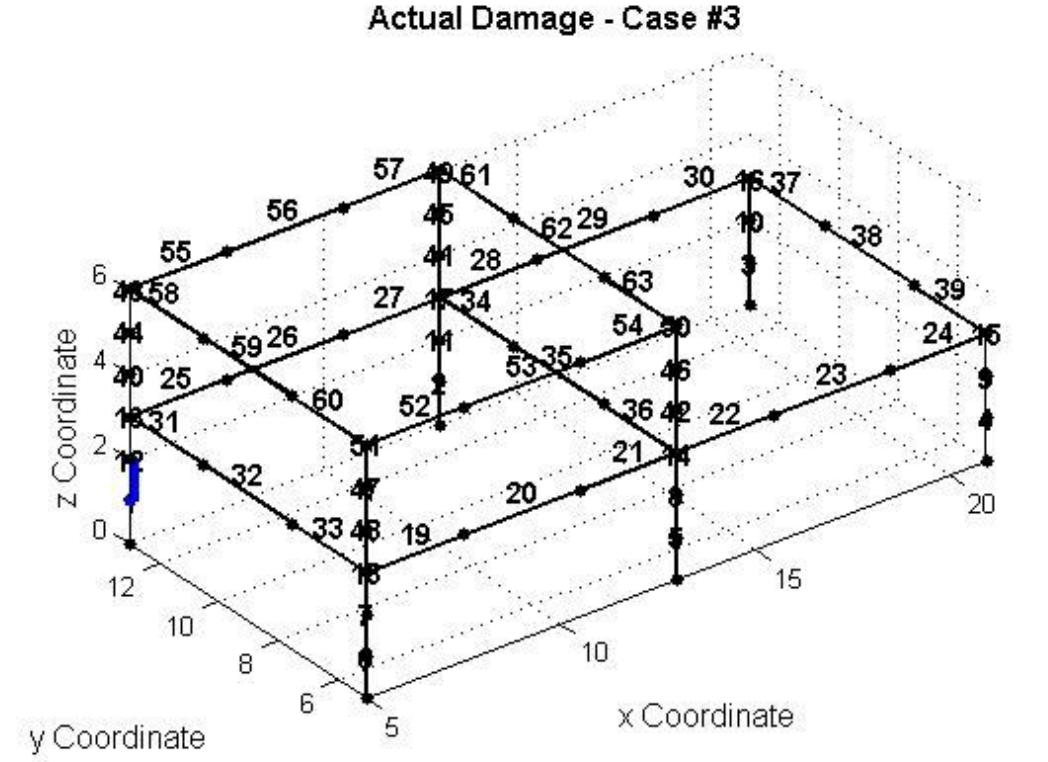


c.

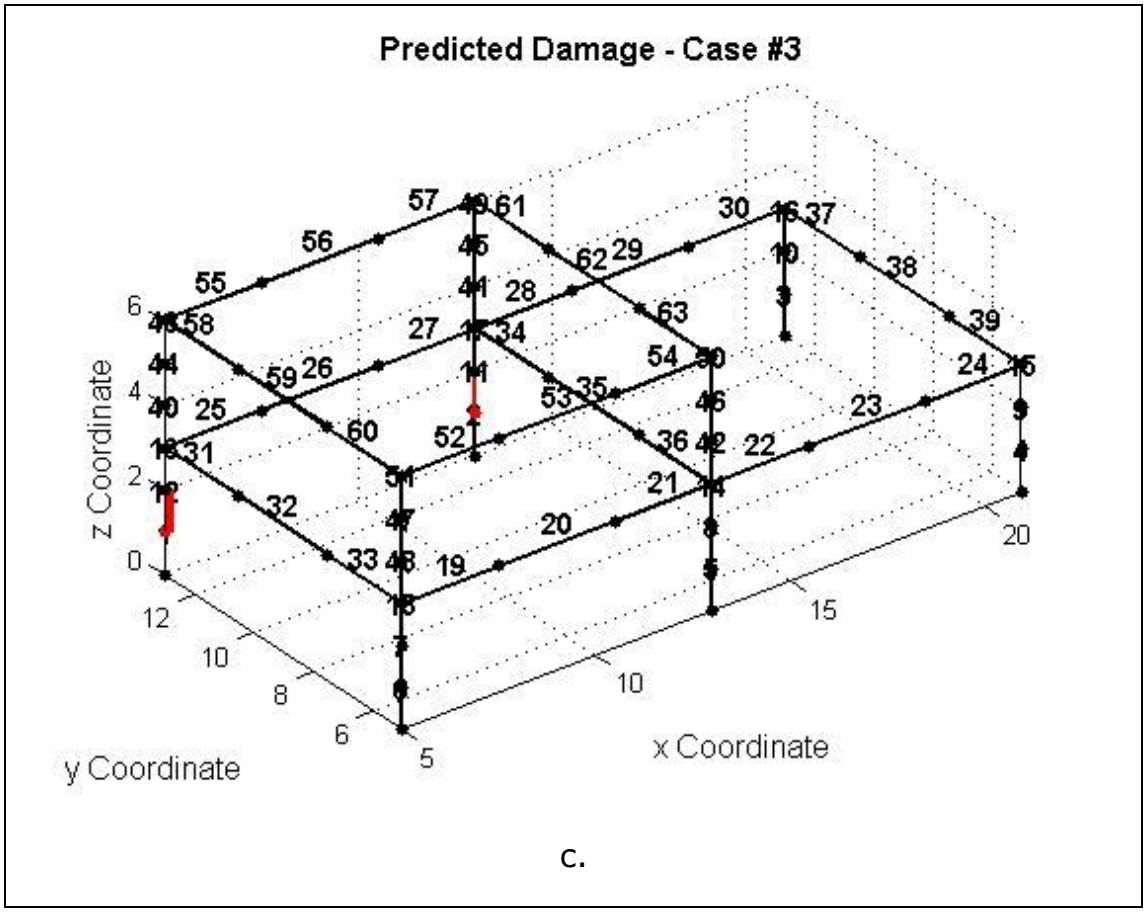
Figure 6.15 a. Damage Identification of case#2 b. Actual damage c. Predicted damage


Discrete Value algorithm - Case #3
Actual damage
Predicted damage
% Decrease in Stiffness
Elements

a.

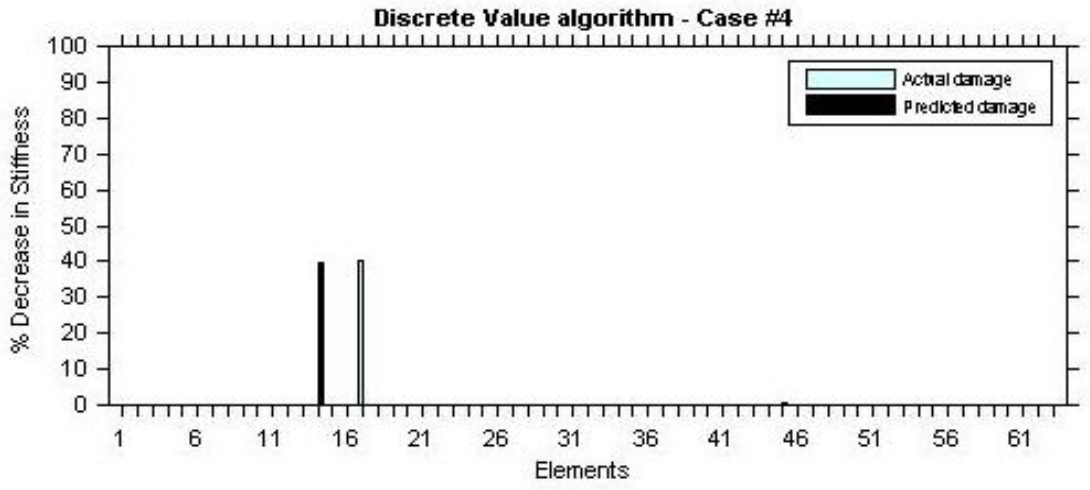

Actual Damage - Case #3
z Coordinate
y Coordinate
x Coordinate

b.

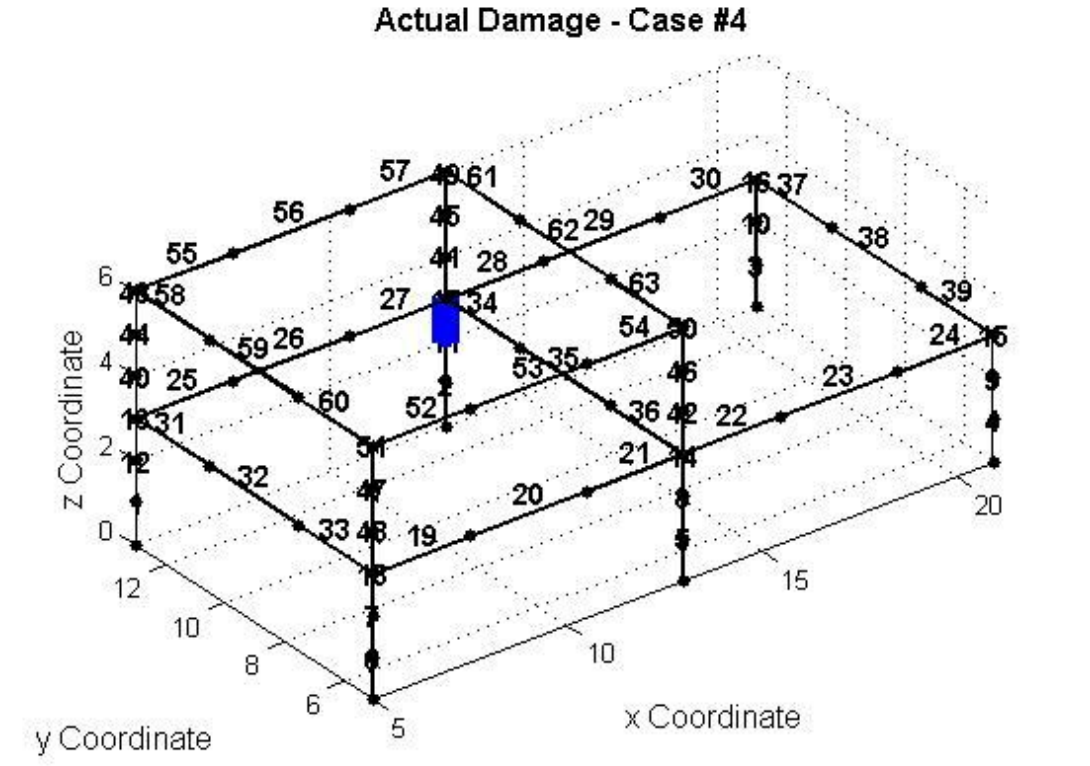


c.

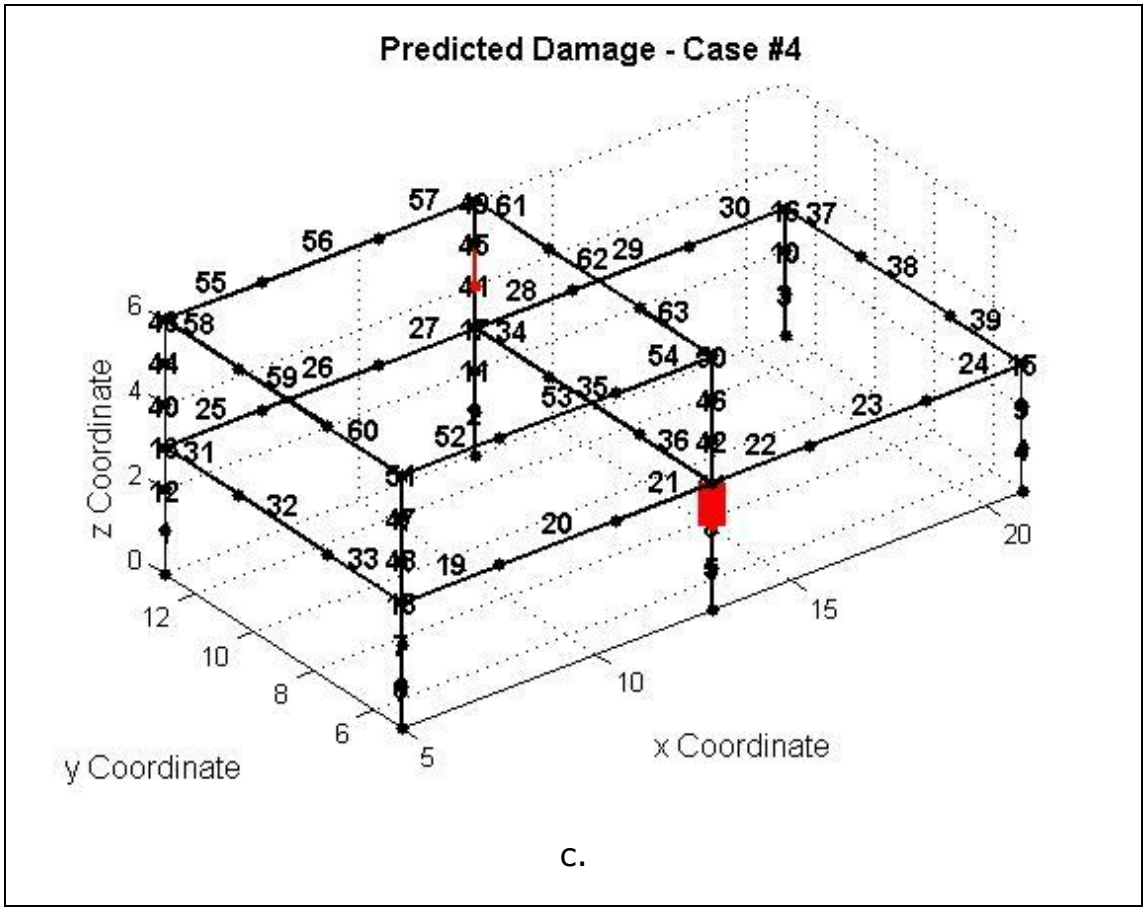
Figure 6.16 a. Damage Identification of case#3 b. Actual damage c. Predicted damage


Discrete Value algorithm - Case #4
% Decrease in Stiffness
Elements
Actual damage
Predicted damage

a.

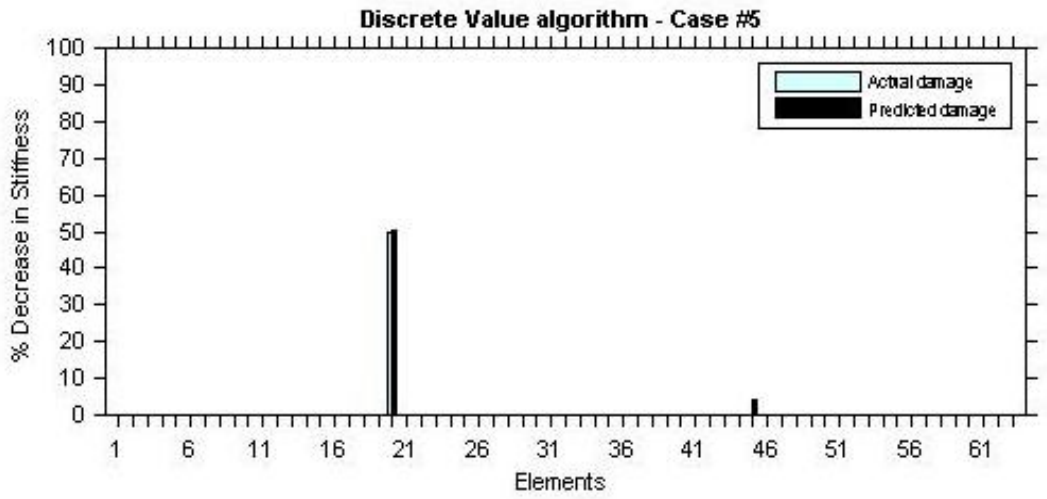

Actual Damage - Case #4
z Coordinate
y Coordinate
x Coordinate

b.

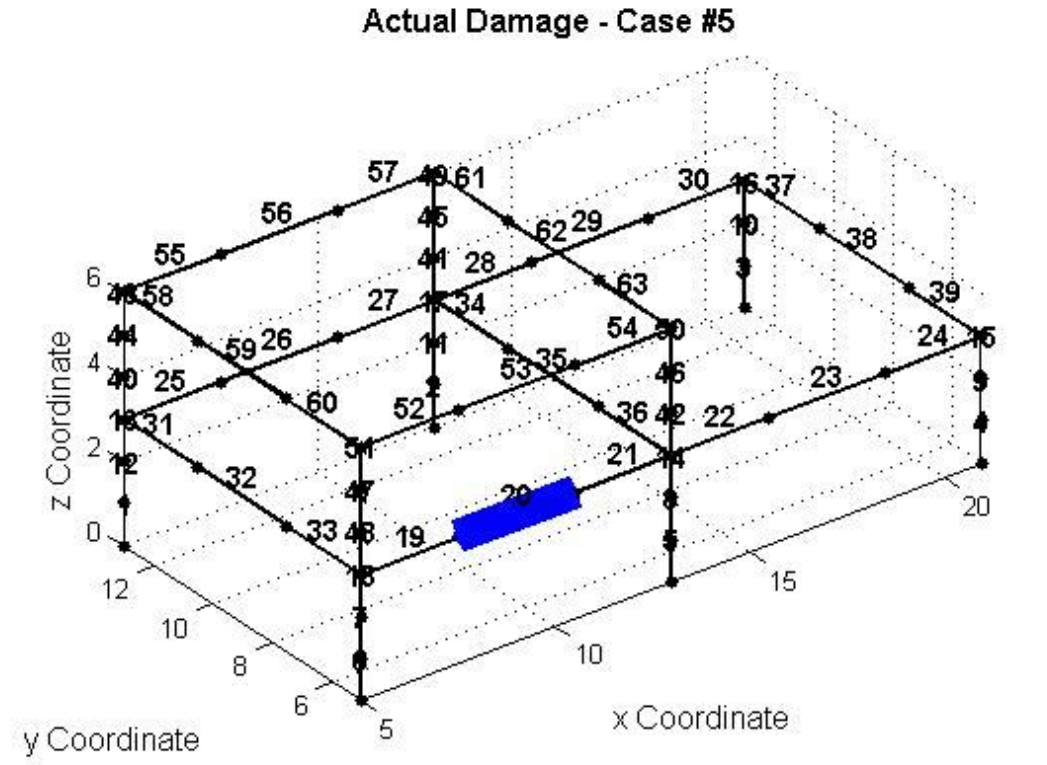


c.

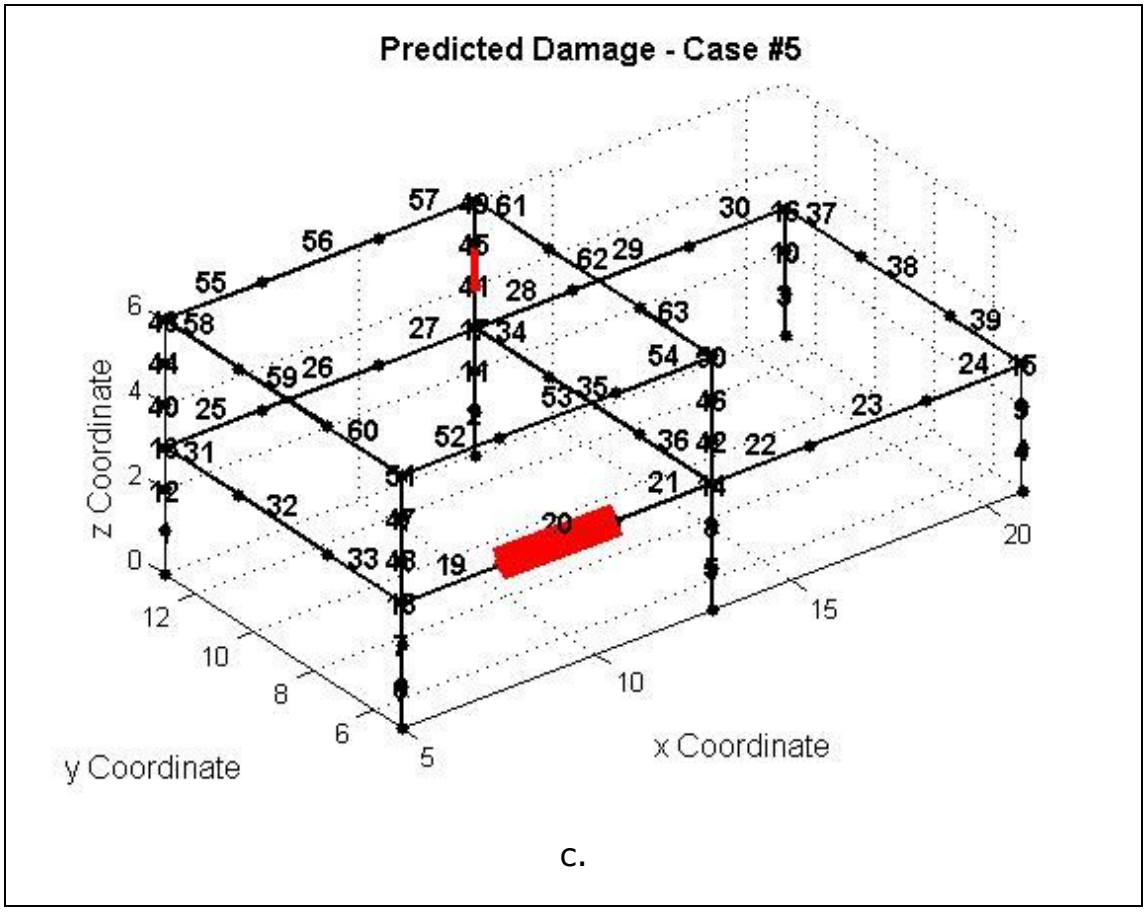
Figure 6.17 a. Damage Identification of case#4 b. Actual damage c. Predicted damage


Discrete Value algorithm - Case #5
% Decrease in Stiffness
Elements
Actual damage
Predicted damage
100
90
80
70
60
50
40
30
20
10
0
1
6
11
16
21
26
31
36
41
46
51
56
61

a.

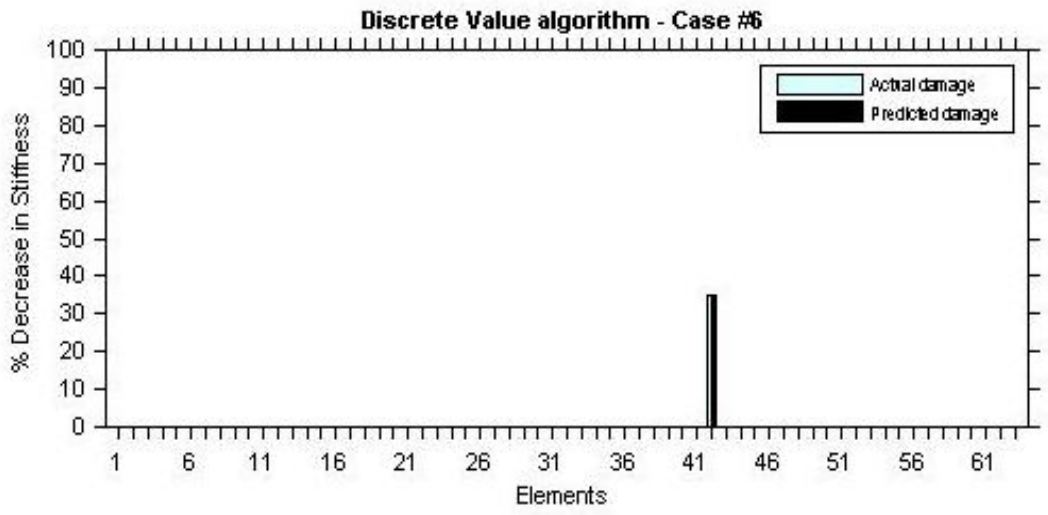

Actual Damage - Case #5
z Coordinate
y Coordinate
x Coordinate
0
2
4
6
12
10
8
6
5
10
15
20

b.

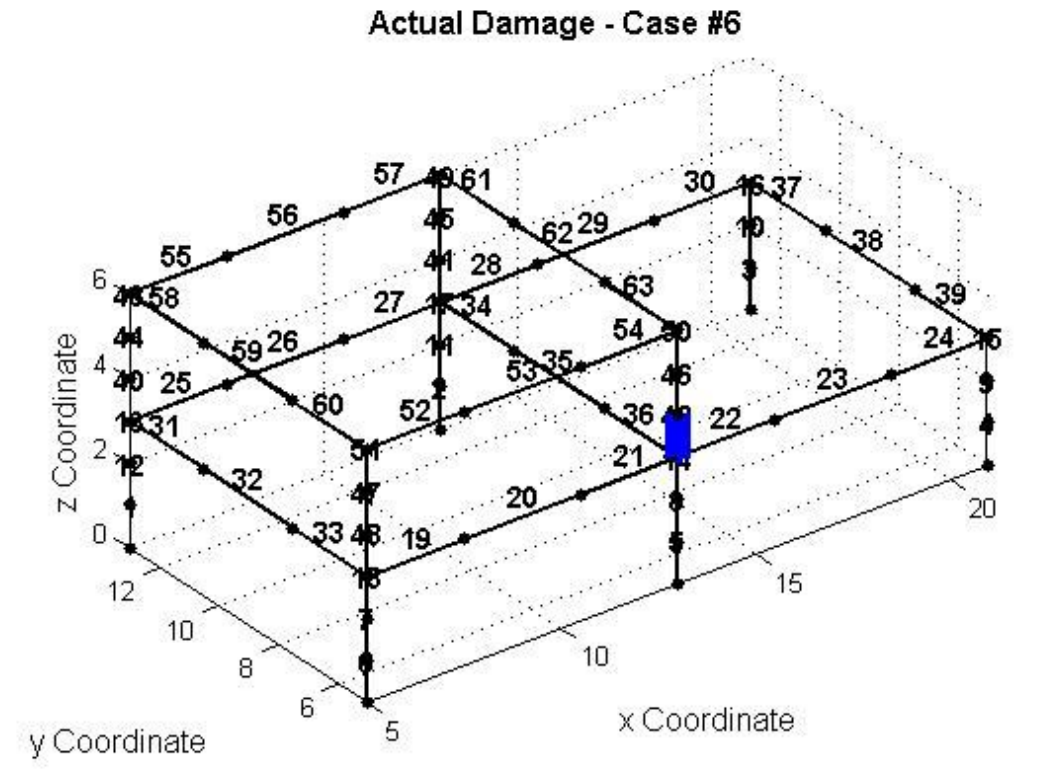


c.

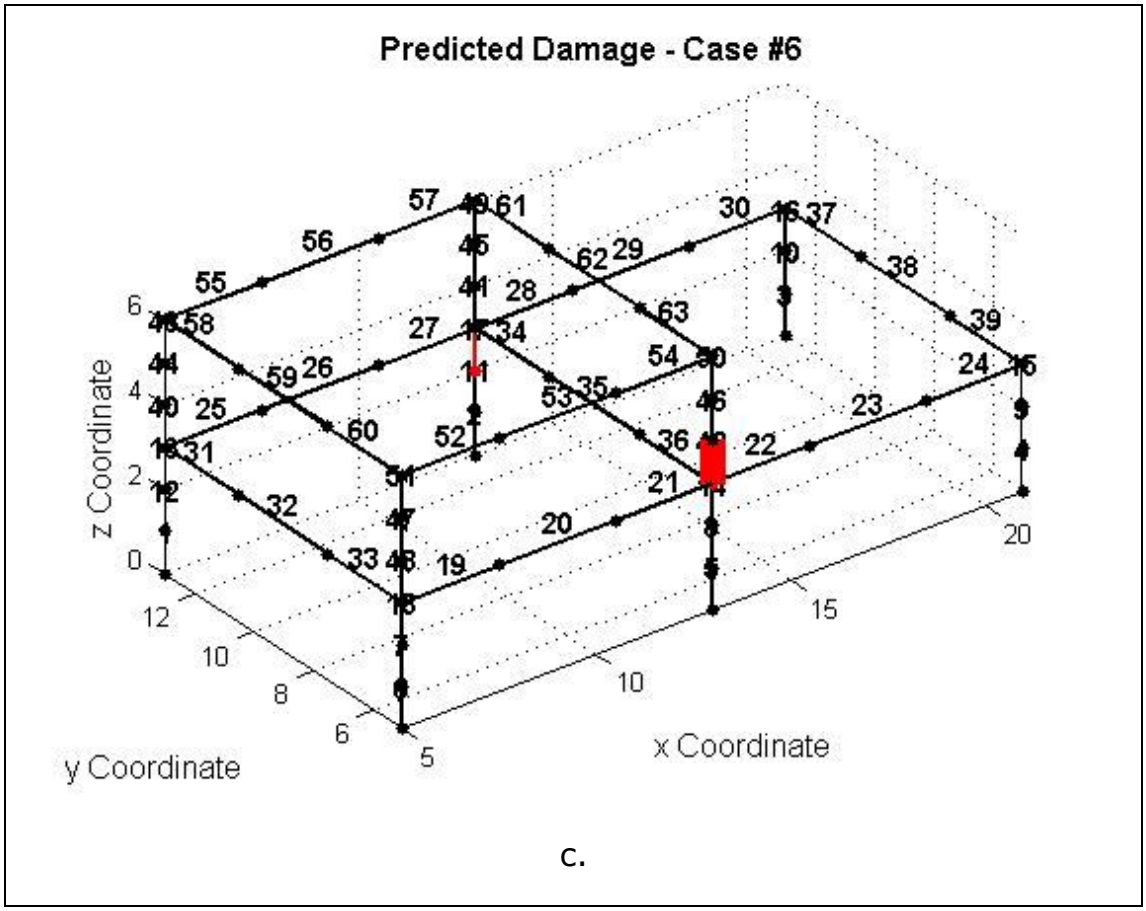
Figure 6.18 a. Damage Identification of case#5 b. Actual damage c. Predicted damage


Discrete Value algorithm - Case #6
Actual damage
Predicted damage
% Decrease in Stiffness
Elements

a.

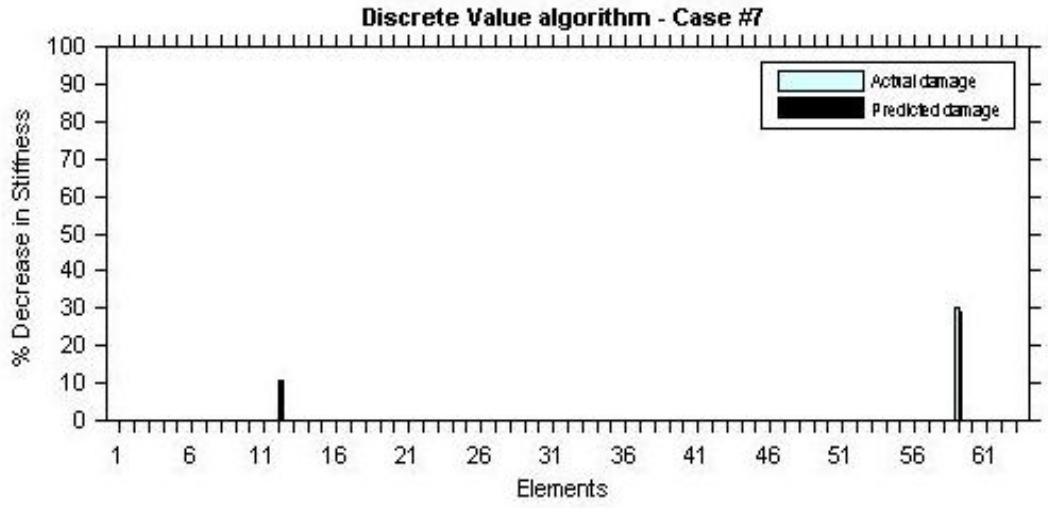

Actual Damage - Case #6
z Coordinate
y Coordinate
x Coordinate

b.

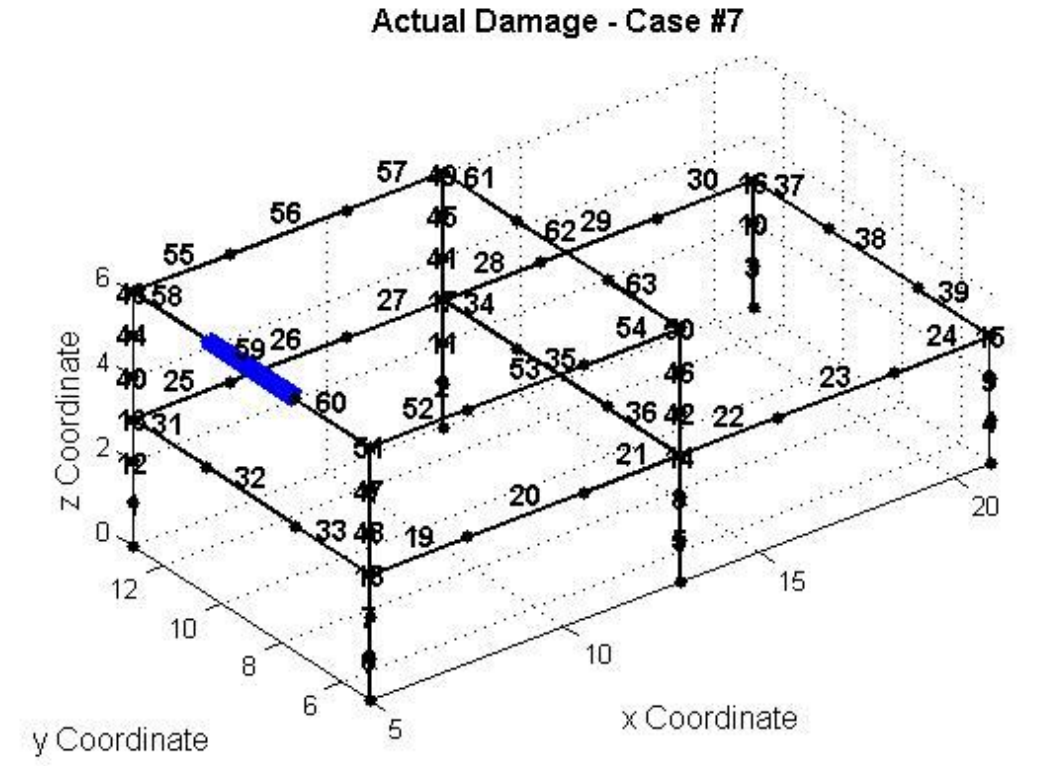


c.

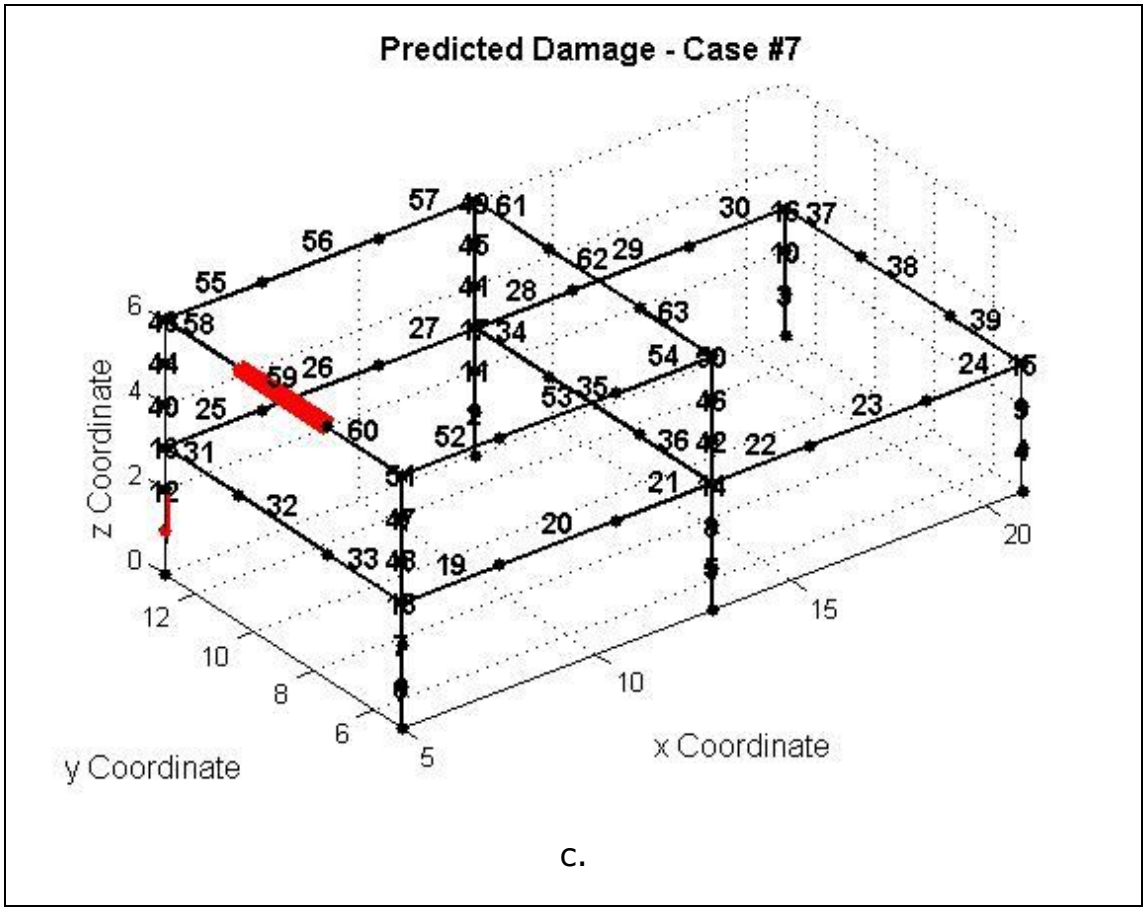
Figure 6.19 a. Damage Identification of case#6 b. Actual damage c. Predicted damage


Discrete Value algorithm - Case #7
% Decrease in Stiffness
Elements
Actual damage
Predicted damage

a.

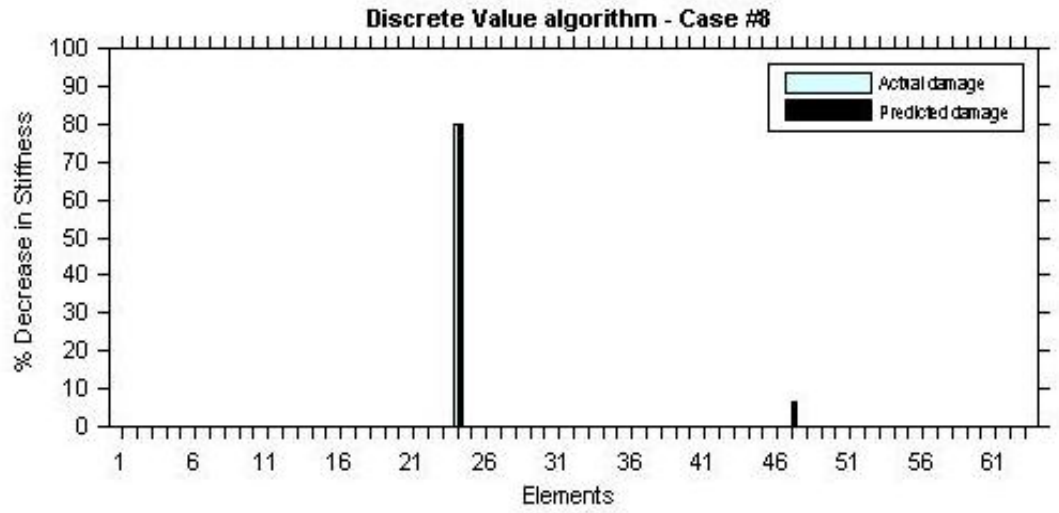

Actual Damage - Case #7
z Coordinate
y Coordinate
x Coordinate

b.

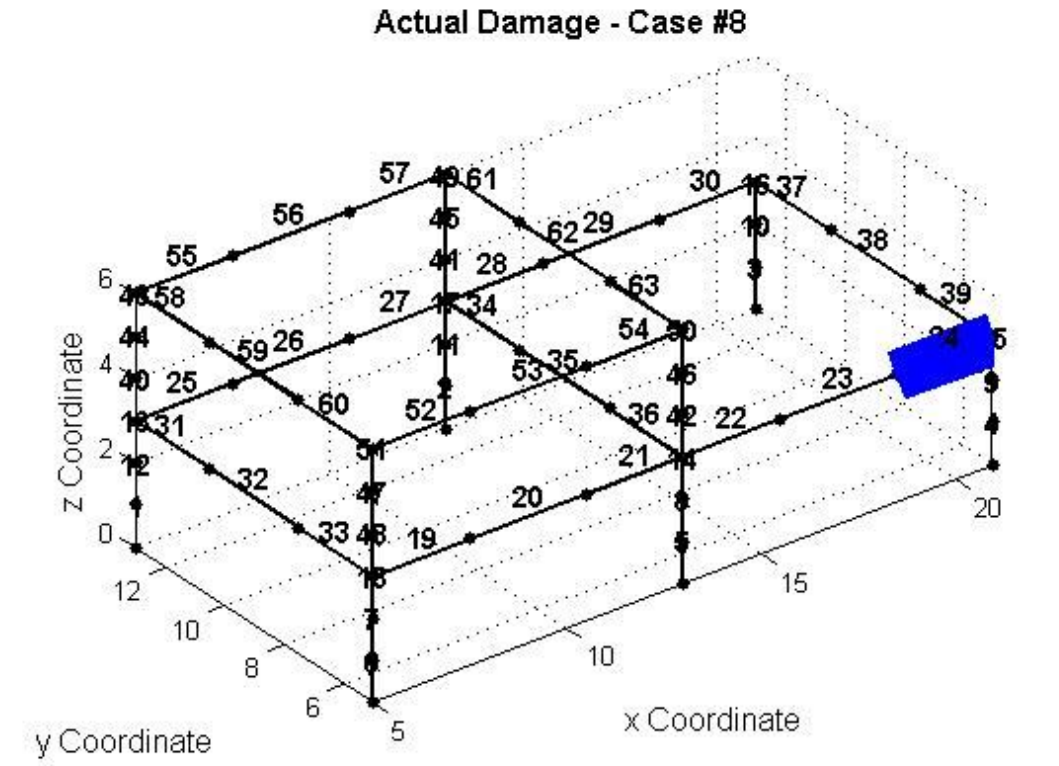


c.

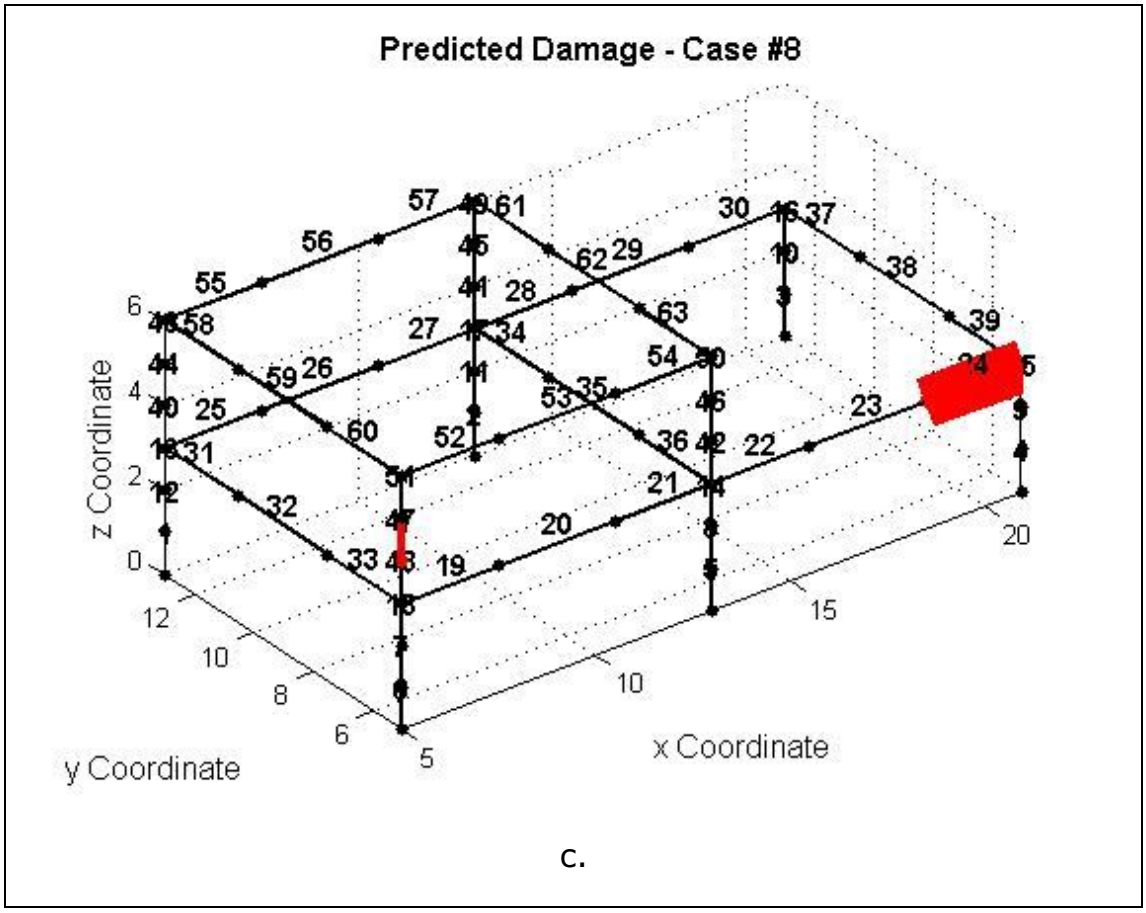
Figure 6.20 a. Damage Identification of case#7 b. Actual damage c. Predicted damage


Discrete Value algorithm - Case #8
% Decrease in Stiffness
Elements
Actual damage
Predicted damage

a.

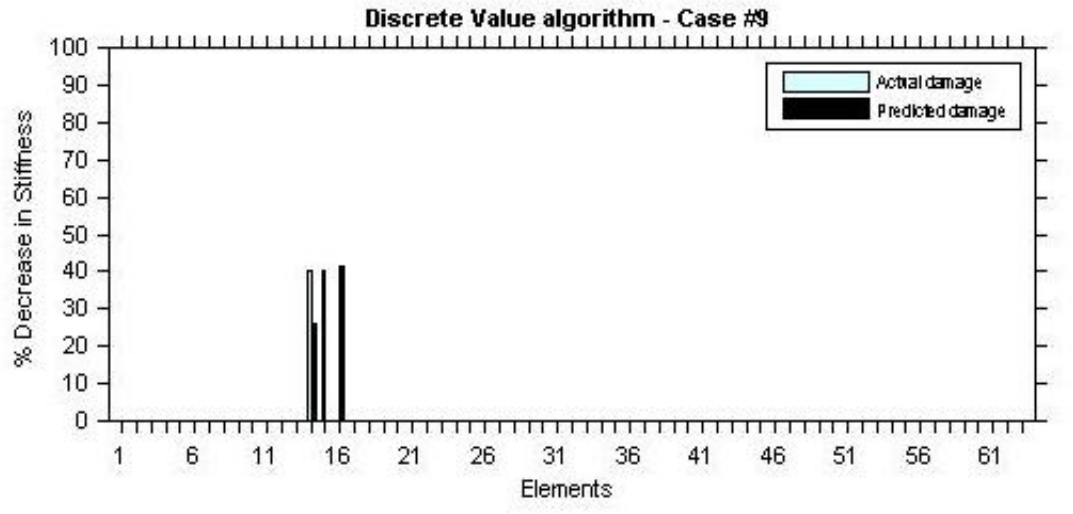

Actual Damage - Case #8
z Coordinate
y Coordinate
x Coordinate

b.



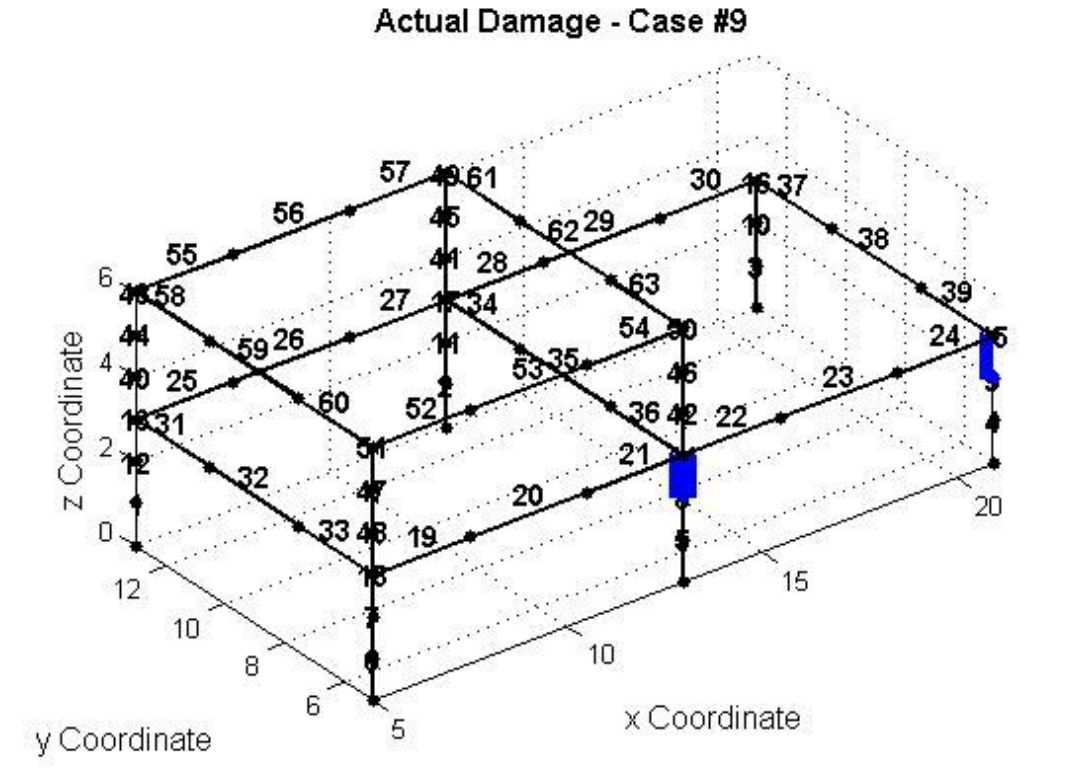
c.

Figure 6.21 a. Damage Identification of case#8 b. Actual damage c. Predicted damage

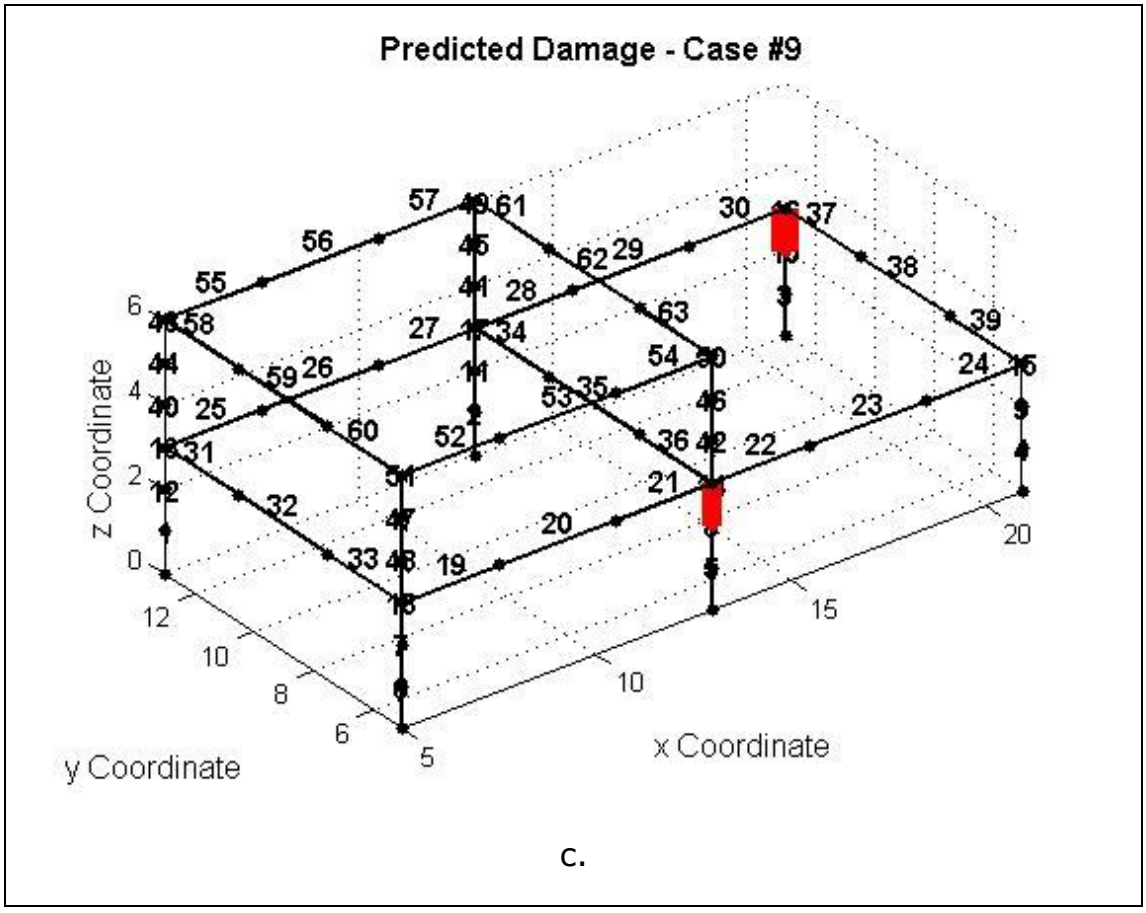

Discrete Value algorithm - Case #9
% Decrease in Stiffness
Elements
Actual damage
Predicted damage

a

.

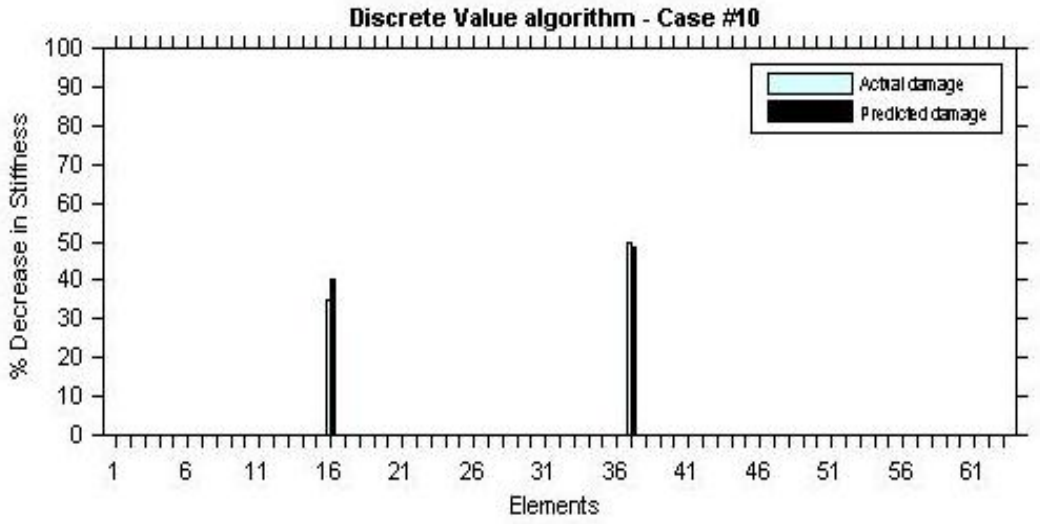

Actual Damage - Case #9
z Coordinate
y Coordinate
x Coordinate

b.

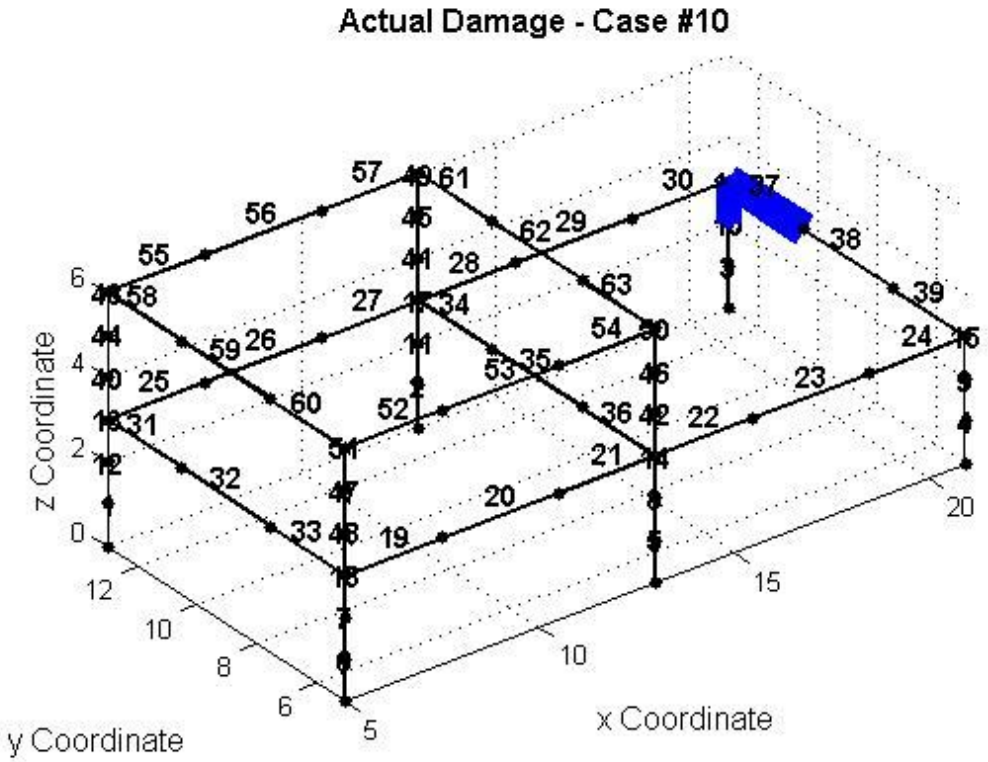


c.

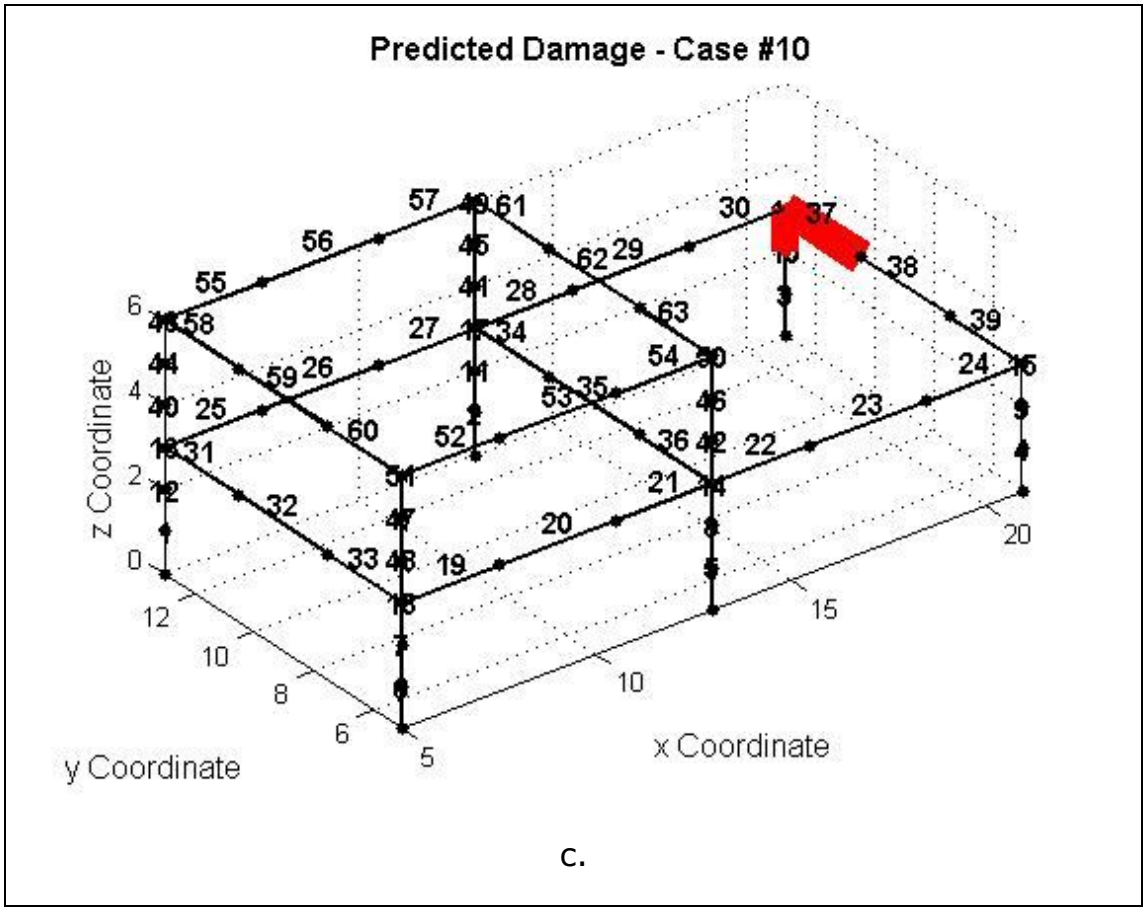
Figure 6.22 a. Damage Identification of case#9 b. Actual damage c. Predicted damage


Discrete Value algorithm - Case #10
Actual damage
Predicted damage
% Decrease in Stiffness
Elements

a.

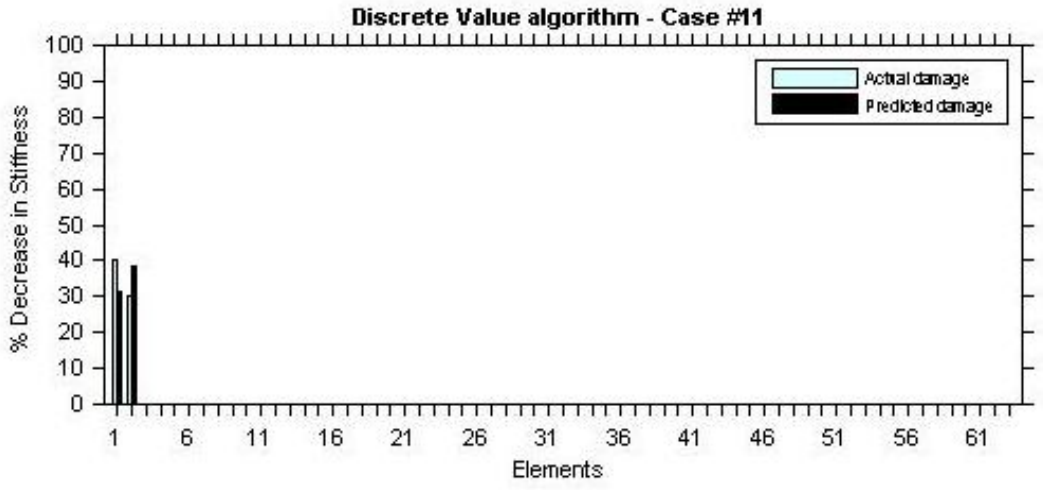

Actual Damage - Case #10
z Coordinate
y Coordinate
x Coordinate

b.

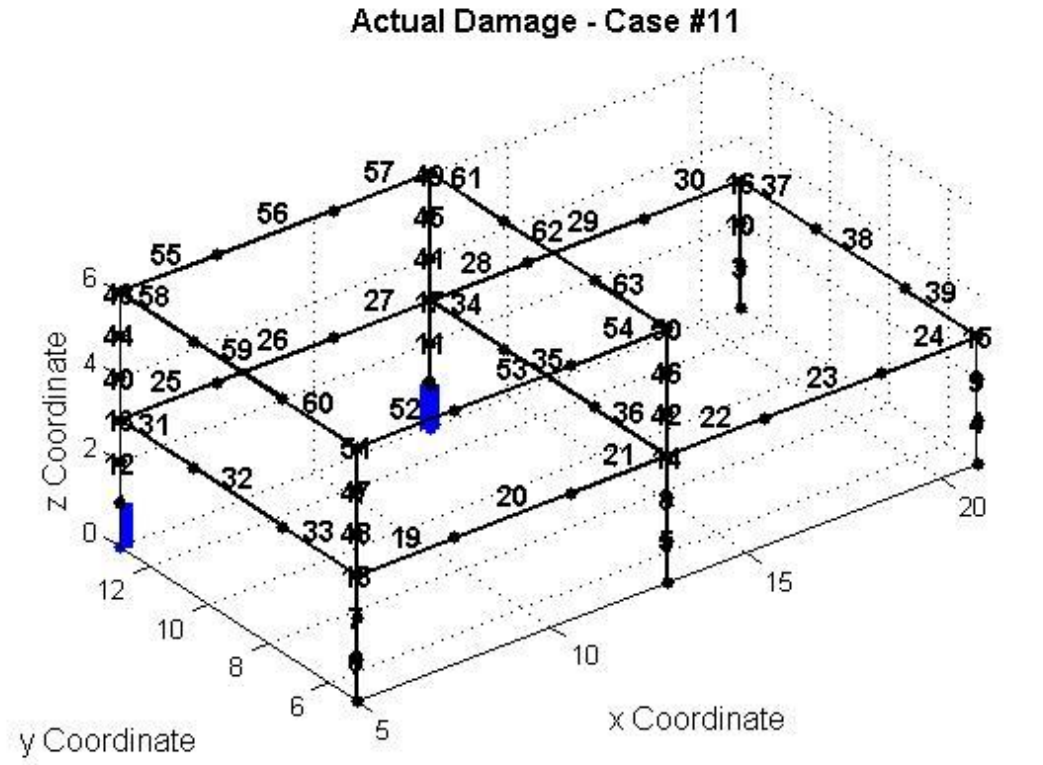


c.

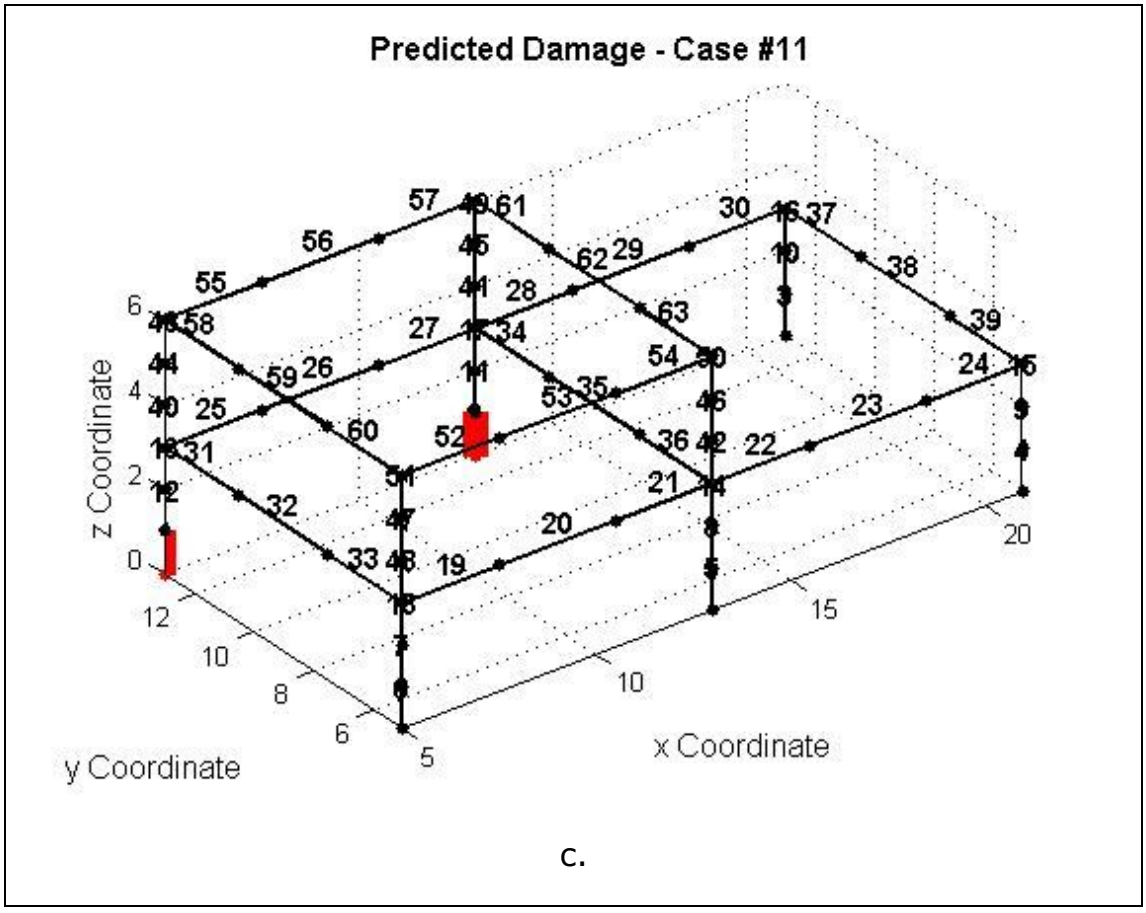
Figure 6.23 a. Damage Identification of case#10 b. Actual damage c. Predicted damage


Discrete Value algorithm - Case #11
% Decrease in Stiffness
Elements
Actual damage
Predicted damage

a.


Actual Damage - Case #11
z Coordinate
y Coordinate
x Coordinate

b.



c.

Figure 6.24 a. Damage Identification of case#11 b. Actual damage c. Predicted damage

6.3 Numerical Results Component mode synthesis

We implemented the discrete value algorithm using component mode synthesis as described in chapter 5. The approximation of the eigenfrequencies at the component mode synthesis procedure is very important in order to perform damage identification. One important aspect is the form of the mass matrix. In the building example as the mass matrix was fully diagonial 312x312 matrix but with fewer significant mass degrees of freedom after little manipulation we were able to produce satisfactory approximations of the eigenvalues (max error 1e-3 for the undamaged case). For the beam on the other hand, even though the dimension was 46x46, the mass matrix was consistent and we had a systematic error (max error 3e-2 for the undamaged case) we included this error at the objective function. This time we run the discrete value algorithm scanning for one damage in the single damage cases and for two damage in the double damage cases. Below we present the results.

6.3.a Numerical Results Component mode synthesis- Beam

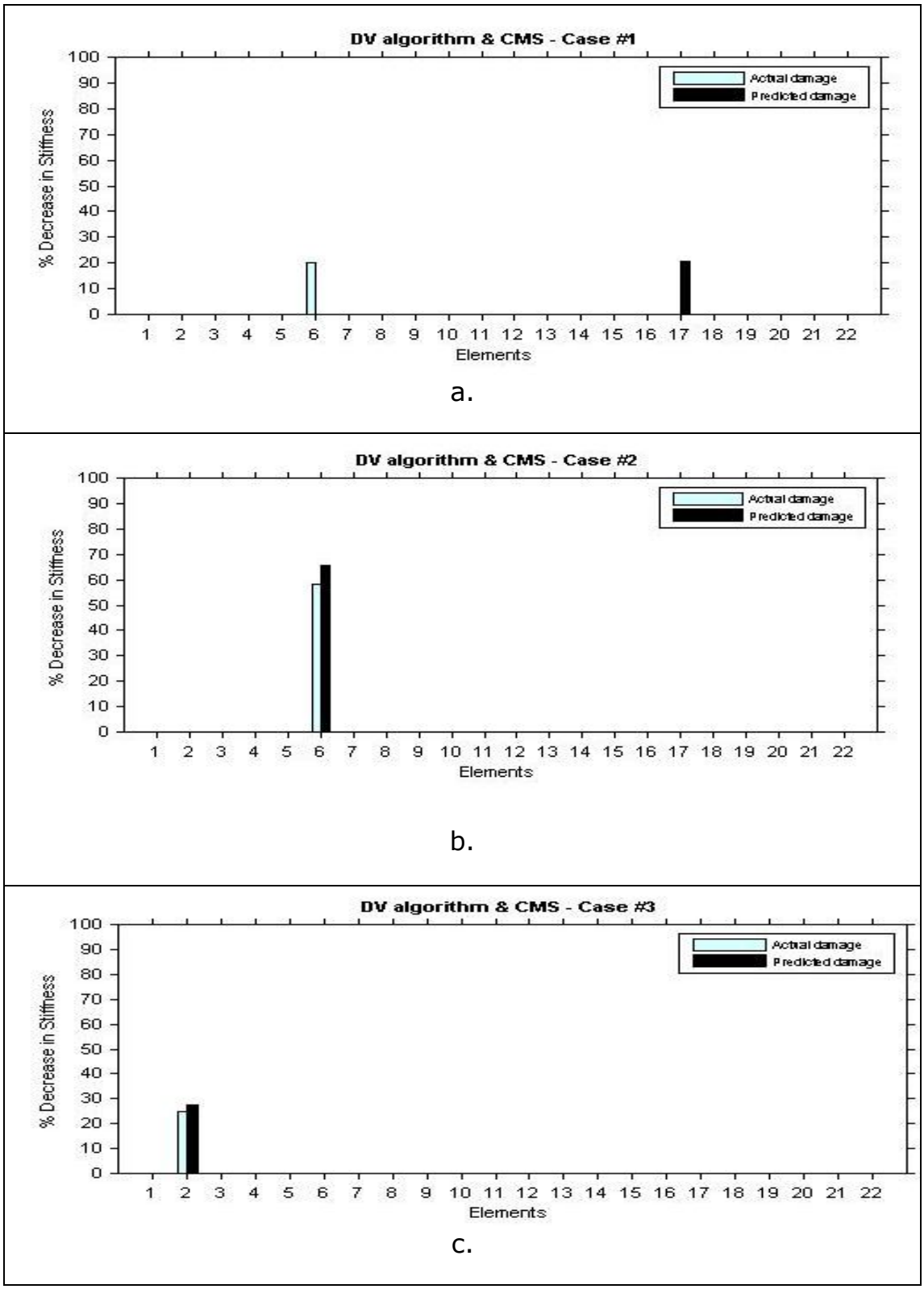


Figure 6.25 Damage identification implementing Discrete Value algorithm and Component Mode Synthesis a.case #1 b. case #2 c. case #3

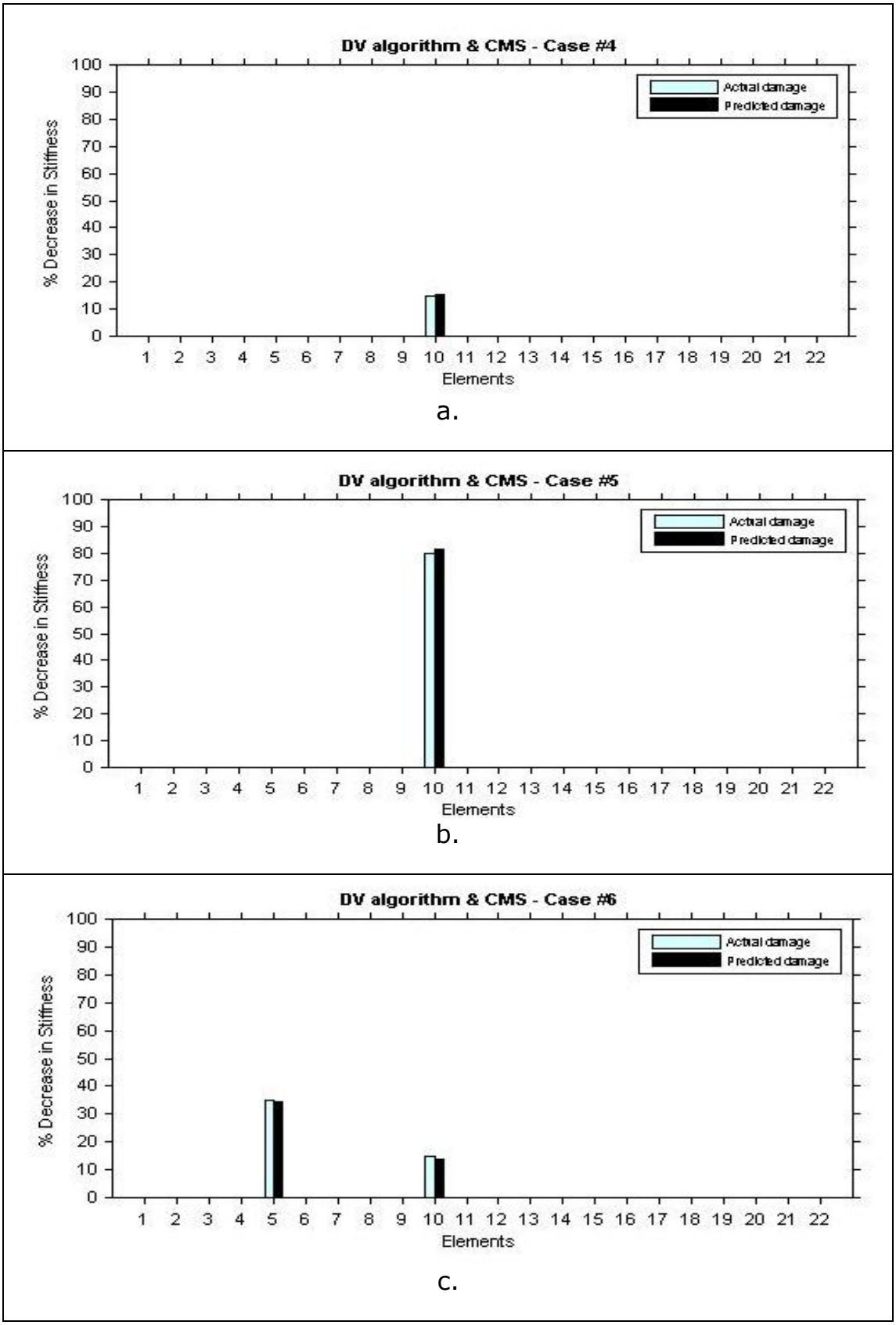


Figure 6.26 Damage identification implementing Discrete Value algorithm and Component Mode Synthesis a.case #4 b. case #5 c. case #6

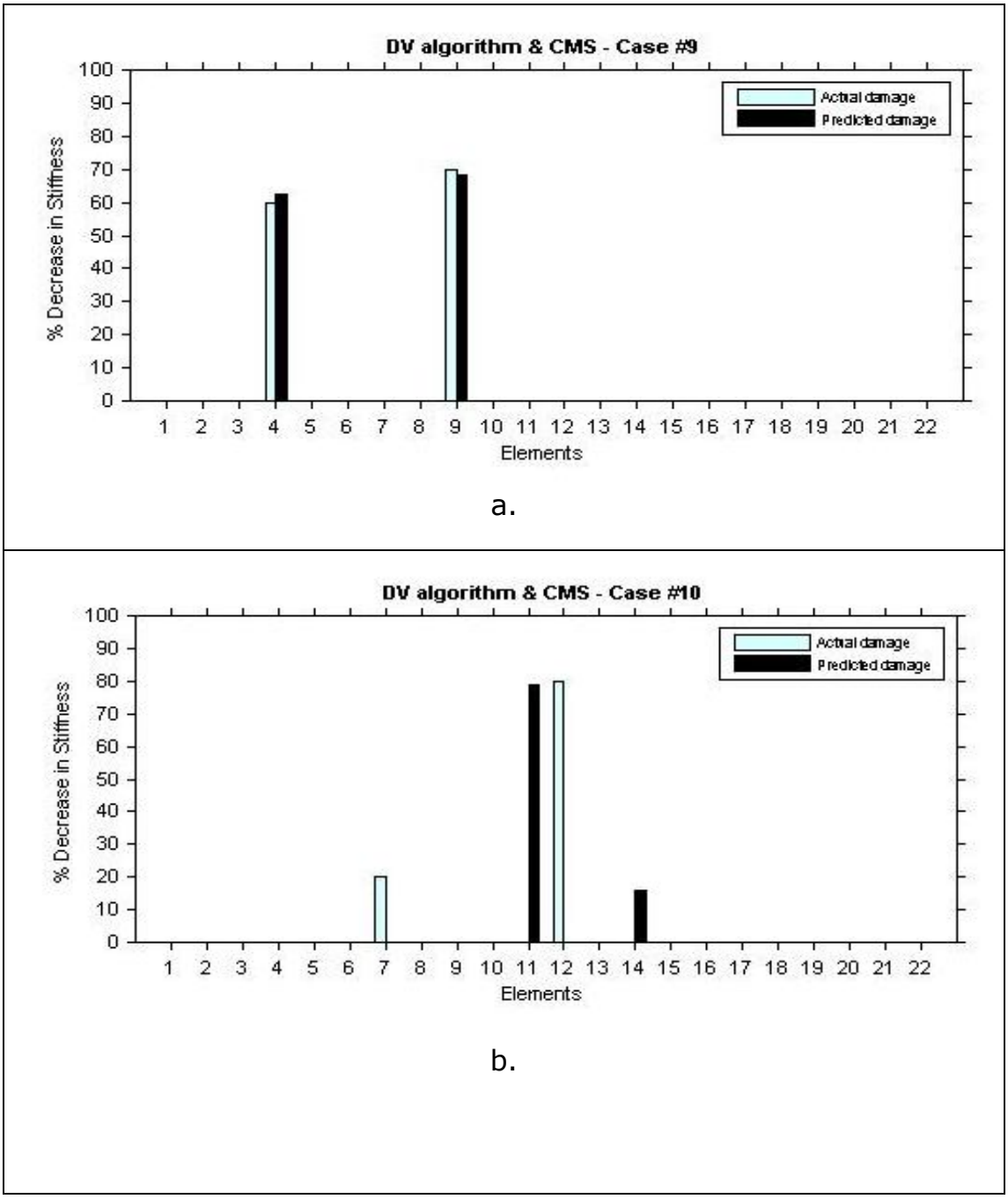
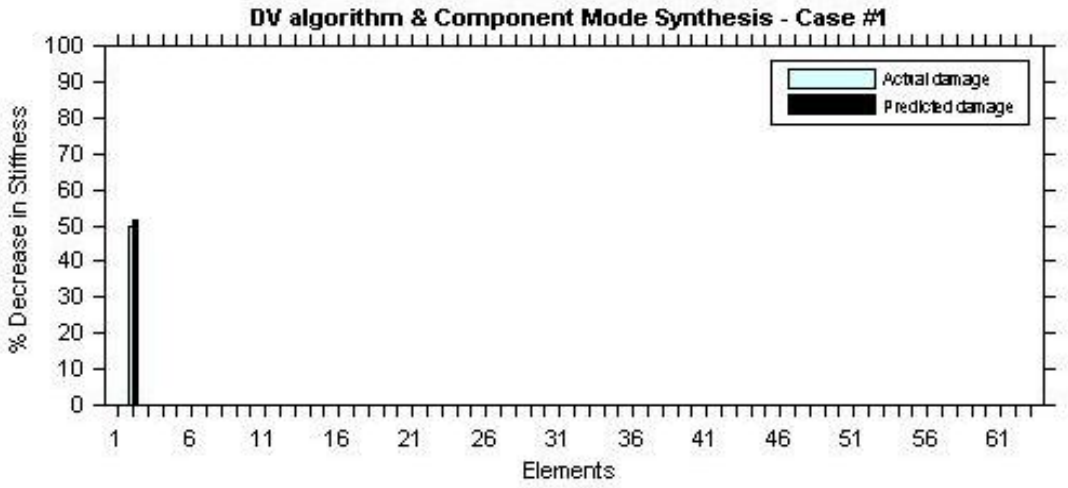
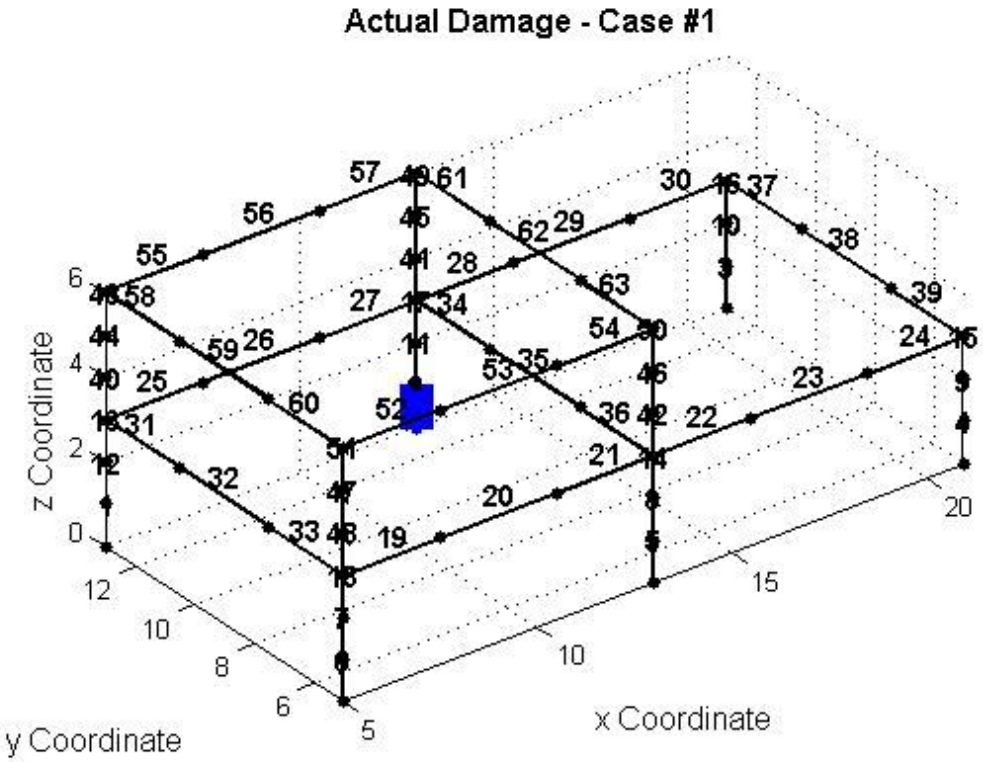


Figure 6.27 Damage identification implementing Discrete Value algorithm and Component Mode Synthesis a.case #9 b. case #10

6.3.b Numerical Results Component mode synthesis- Steel frame building



a.



b.

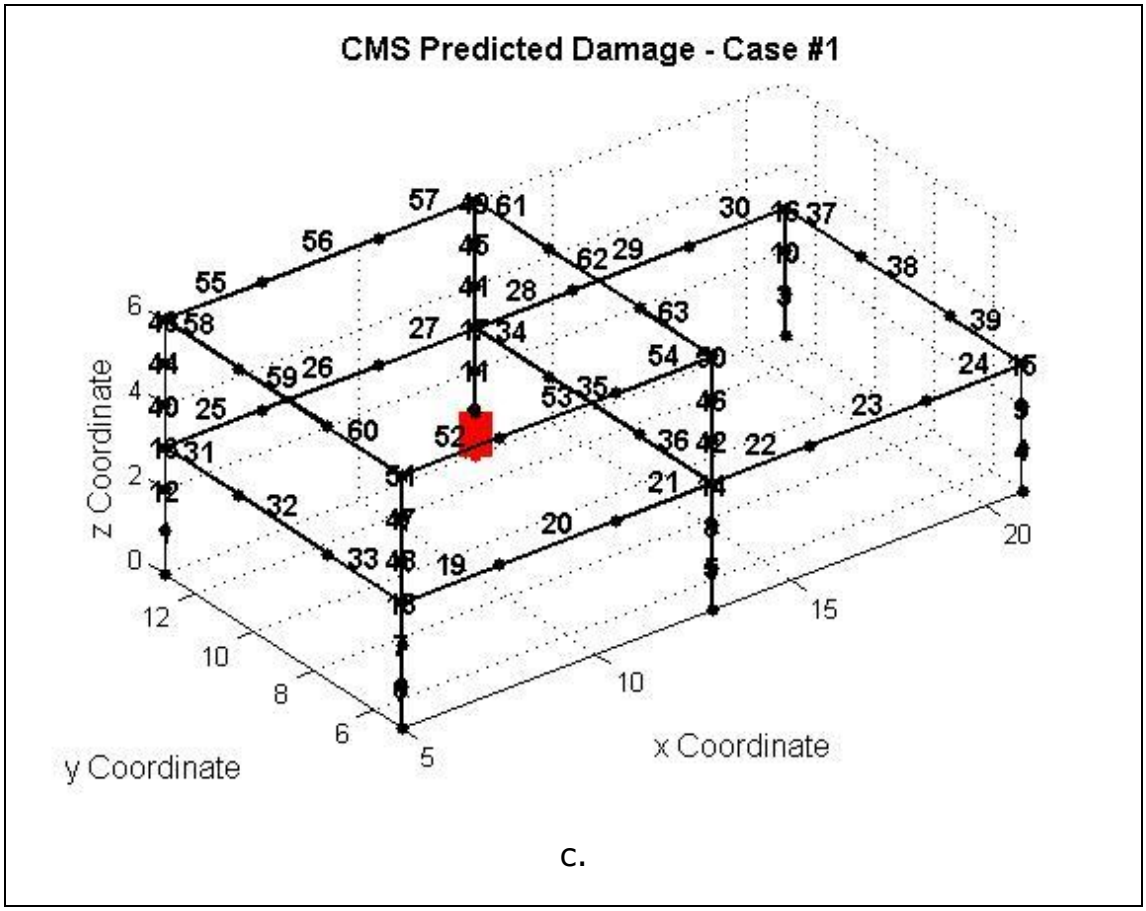
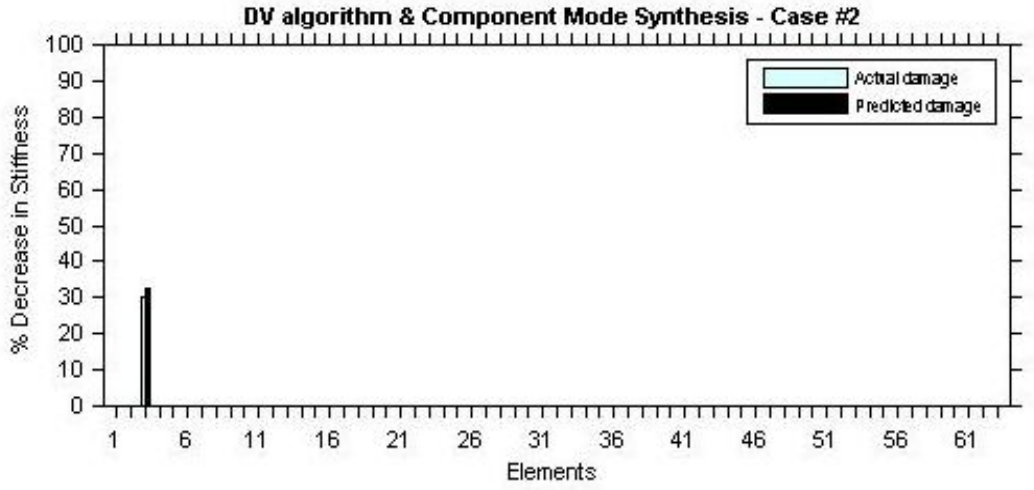
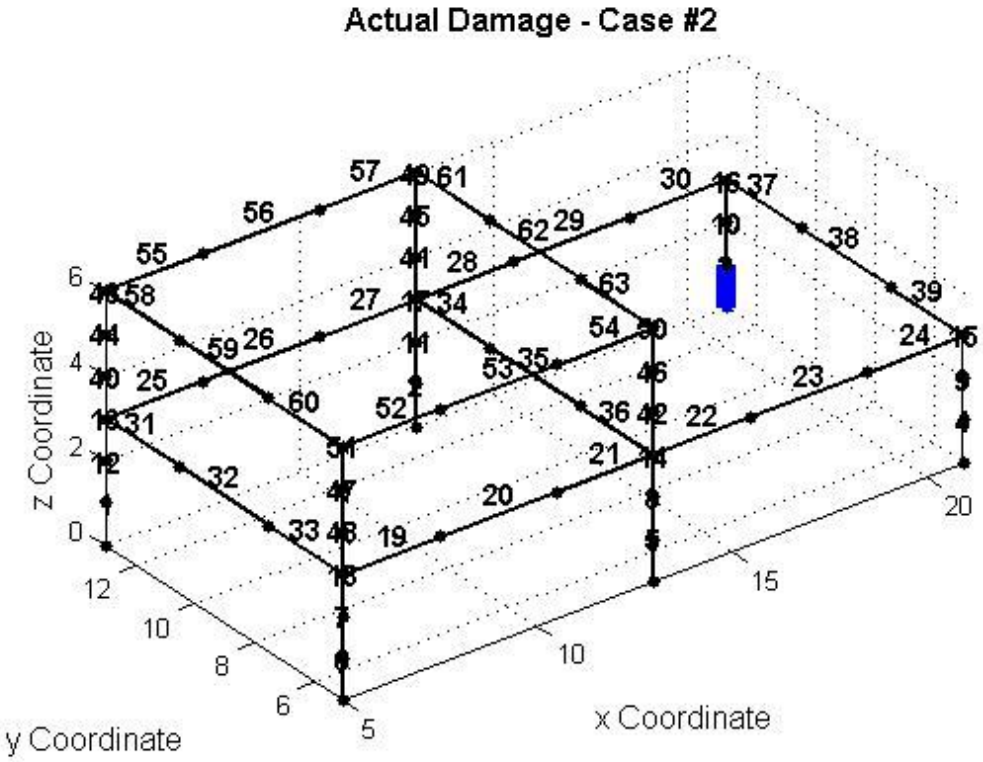


Figure 6.28 a. Damage identification implementing Discrete Value algorithm and Component Mode Synthesis case #1 b. Actual damage c. Predicted damage


DV algorithm & Component Mode Synthesis - Case #2
% Decrease in Stiffness
Elements
Actual damage
Predicted damage

a

.


Actual Damage - Case #2
z Coordinate
y Coordinate
x Coordinate

b.

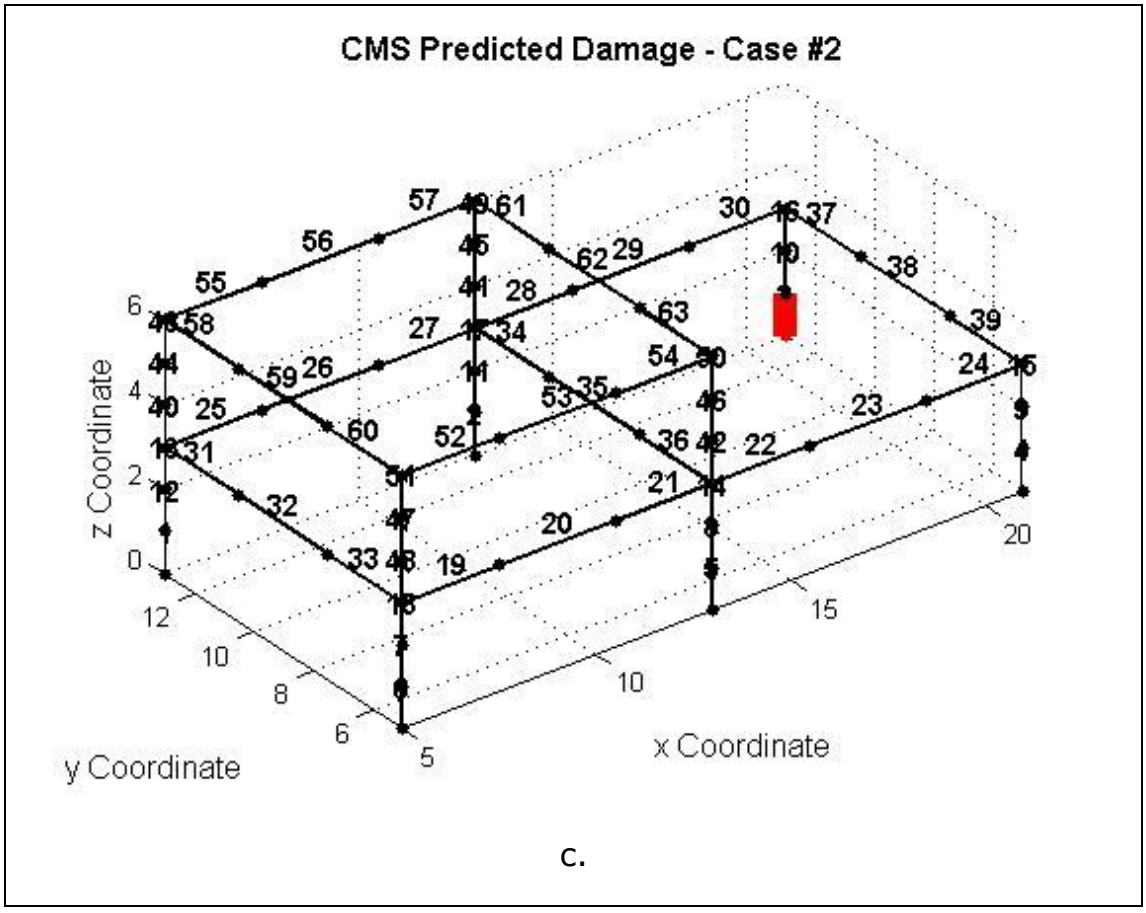
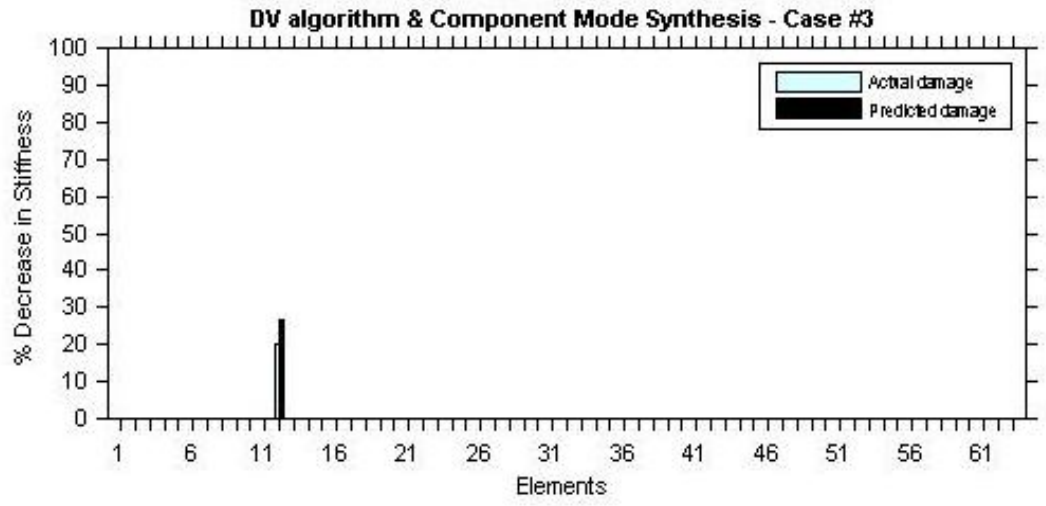
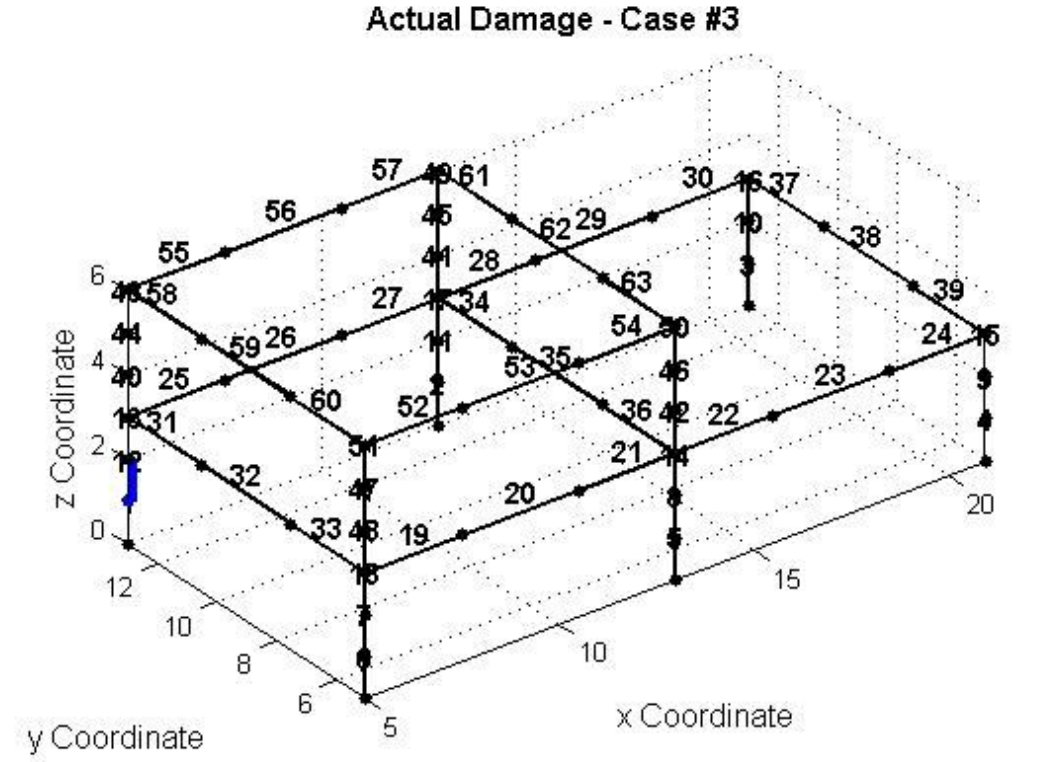


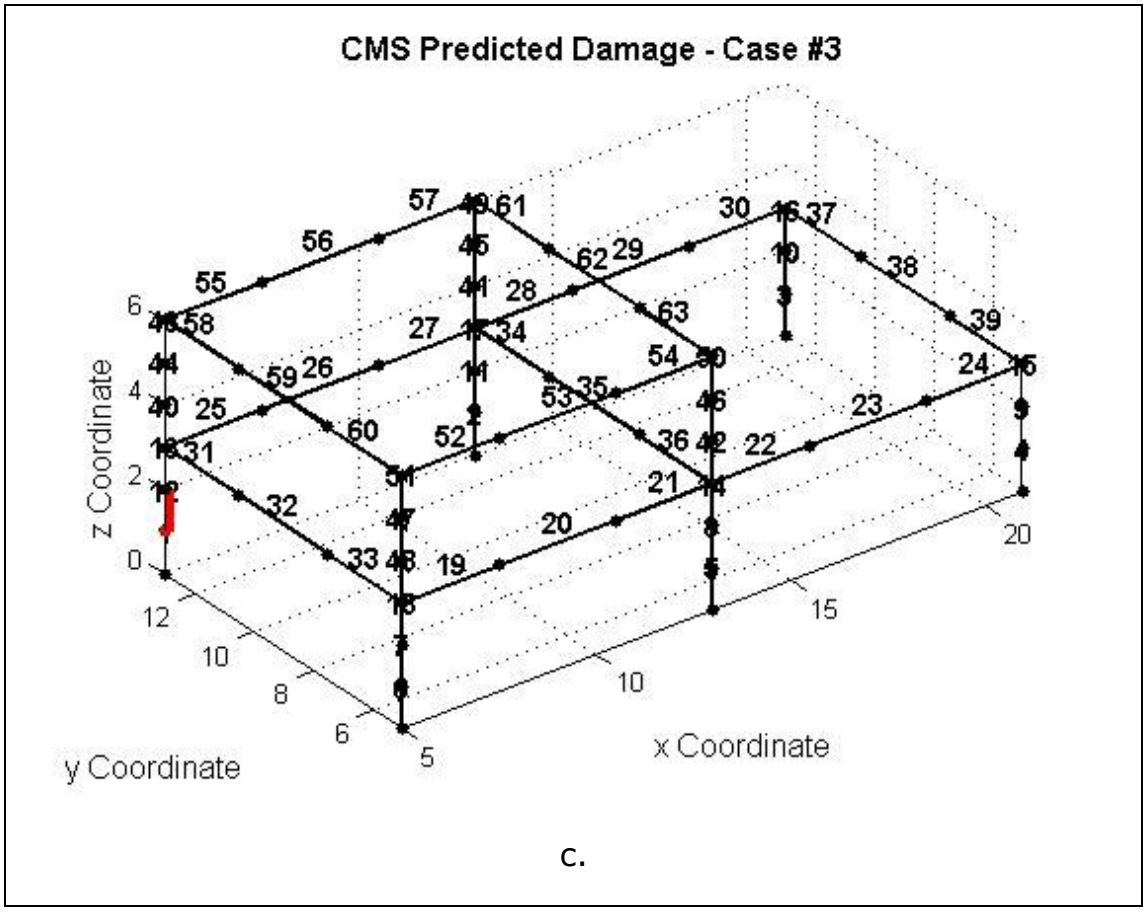
Figure 6.29 a. Damage identification implementing Discrete Value algorithm and Component Mode Synthesis case #2 b. Actual damage c. Predicted damage


DV algorithm & Component Mode Synthesis - Case #3
% Decrease in Stiffness
Elements
Actual damage
Predicted damage

a.

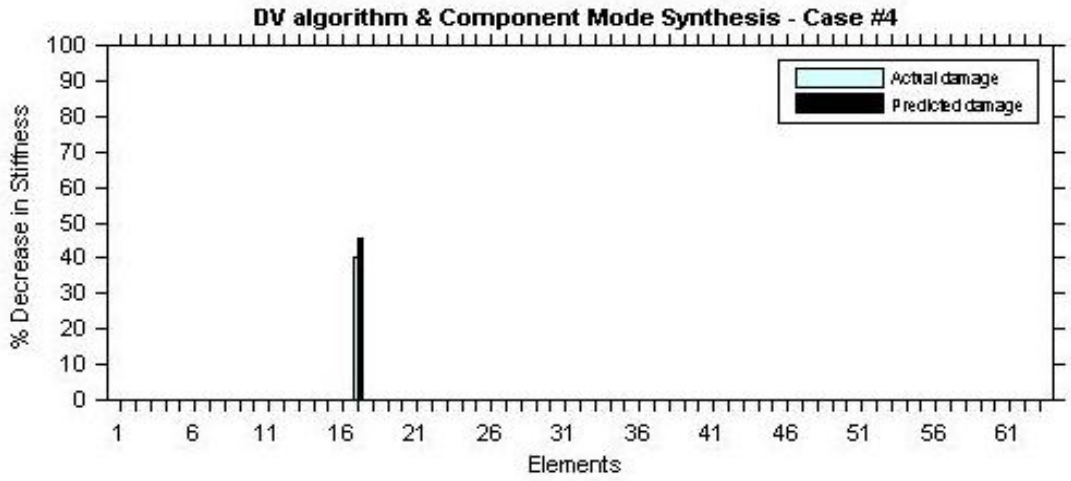

Actual Damage - Case #3
z Coordinate
y Coordinate
x Coordinate

b.

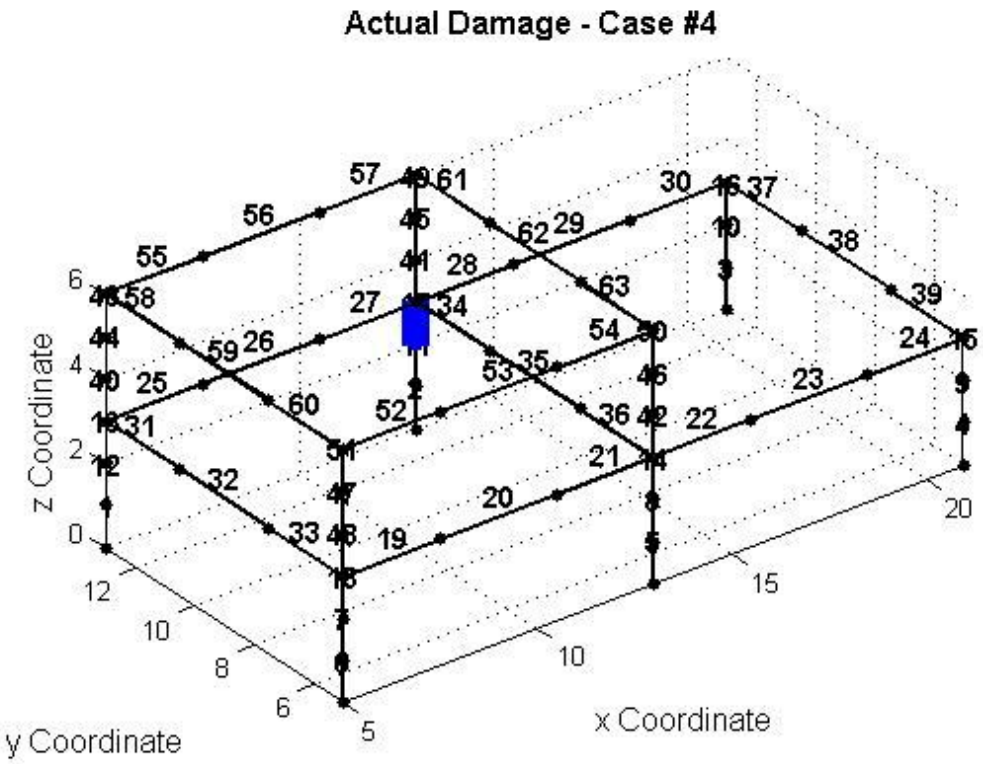


c.

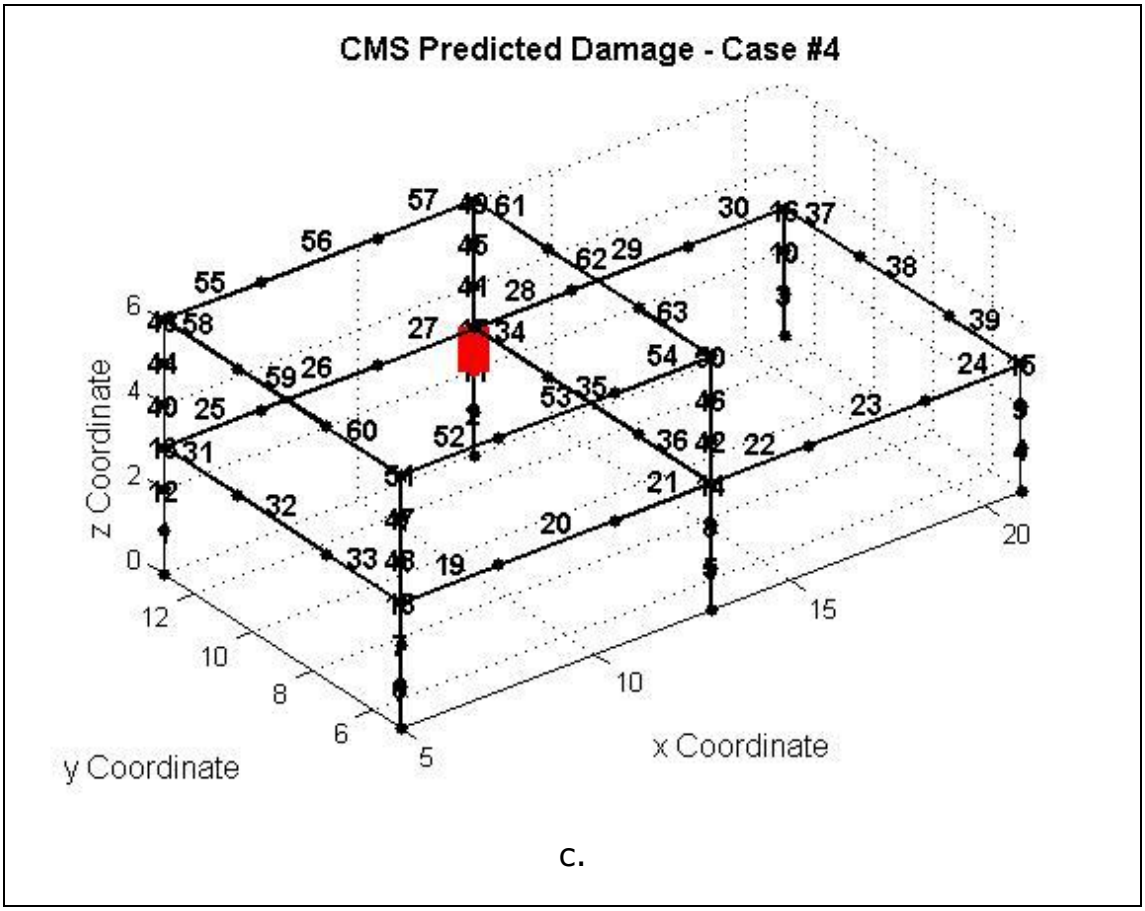
Figure 6.30 a. Damage identification implementing Discrete Value algorithm and Component Mode Synthesis case #3 b. Actual damage c. Predicted damage


DV algorithm & Component Mode Synthesis - Case #4
% Decrease in Stiffness
Elements
Actual damage
Predicted damage

a.

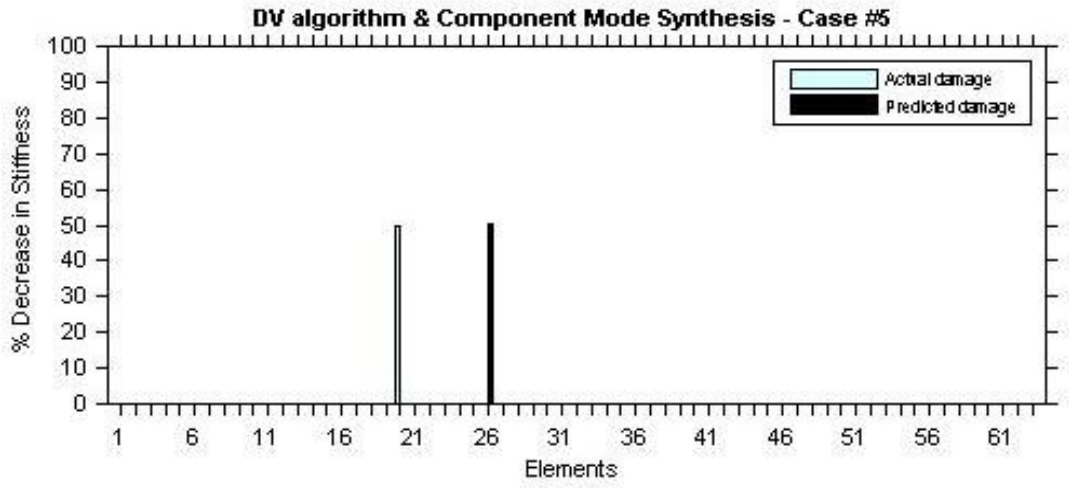

Actual Damage - Case #4
z Coordinate
y Coordinate
x Coordinate

b.

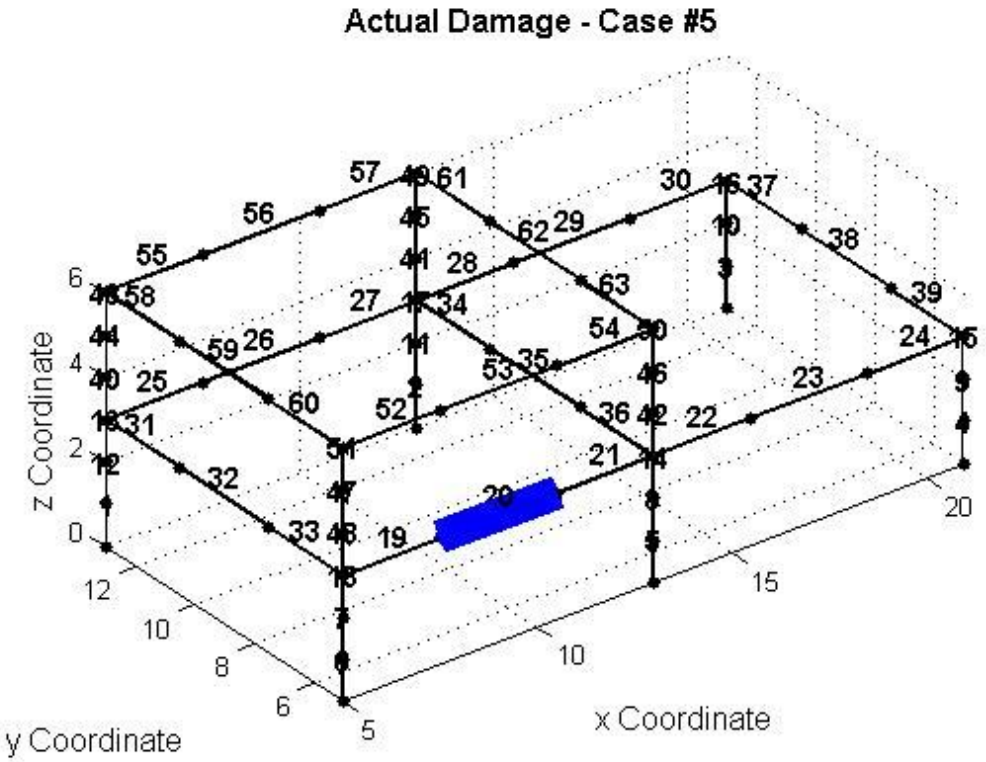


c.

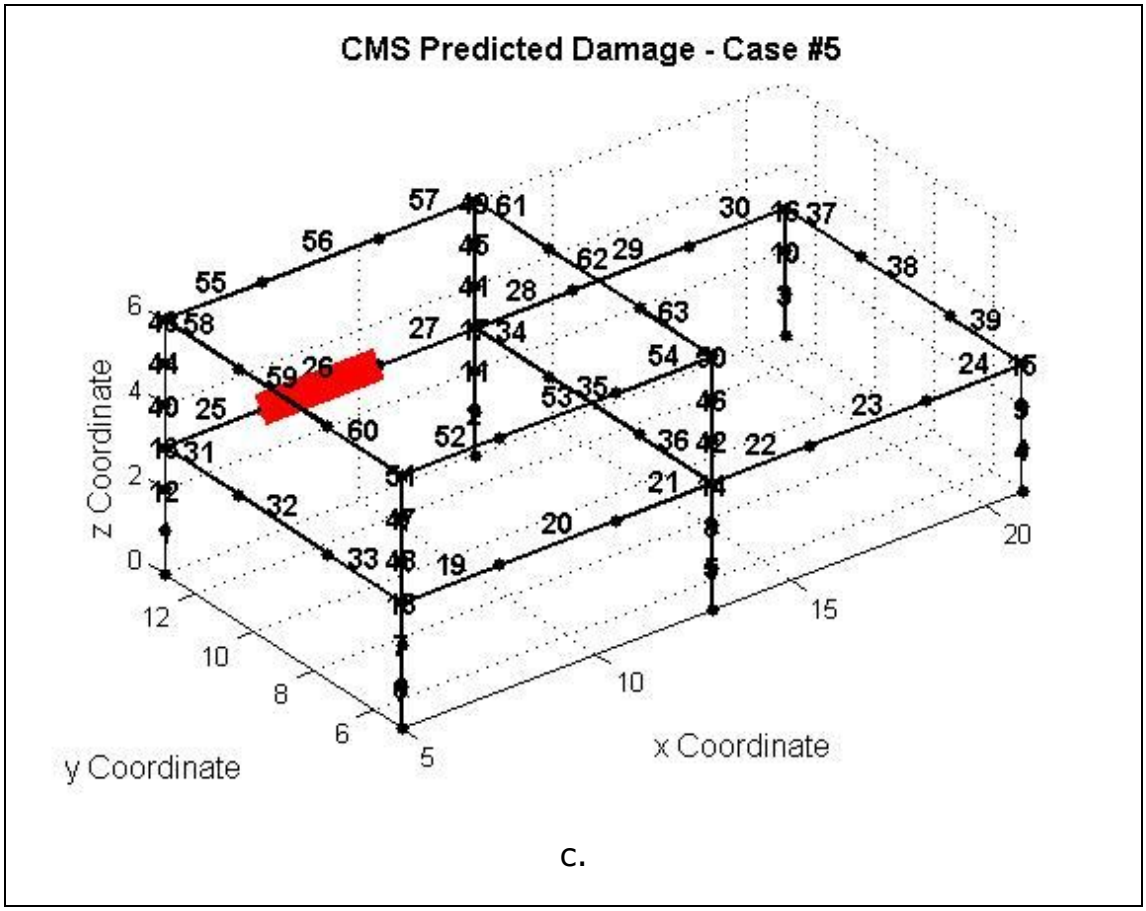
Figure 6.31 a. Damage identification implementing Discrete Value algorithm and Component Mode Synthesis case #4 b. Actual damage c. Predicted damage


DV algorithm & Component Mode Synthesis - Case #5
% Decrease in Stiffness
Elements
Actual damage
Predicted damage

a.

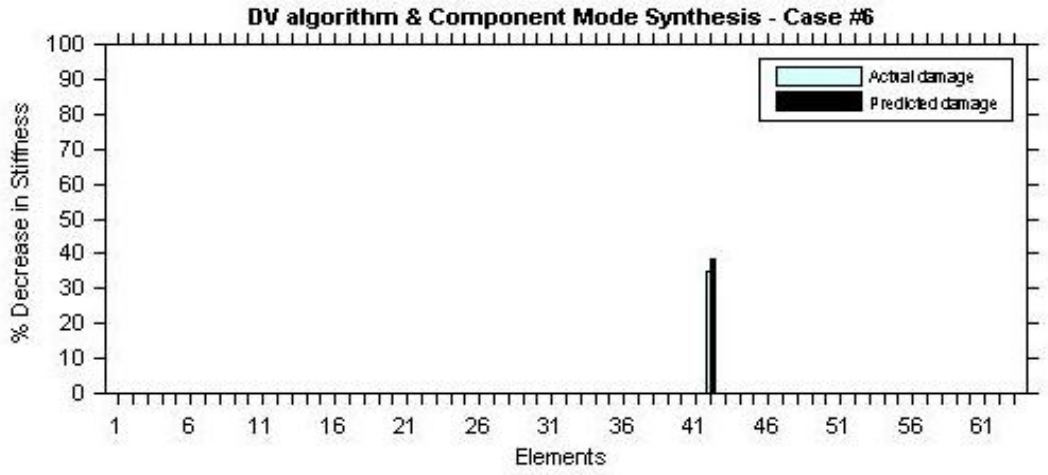

Actual Damage - Case #5
z Coordinate
y Coordinate
x Coordinate

b.

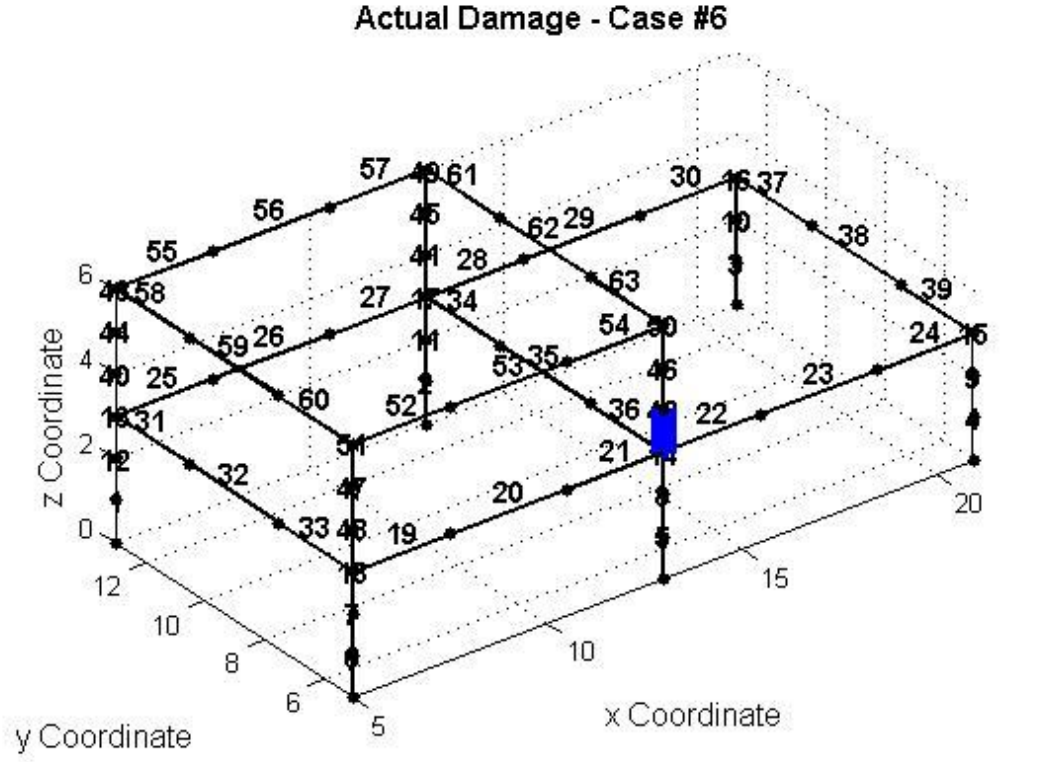


c.

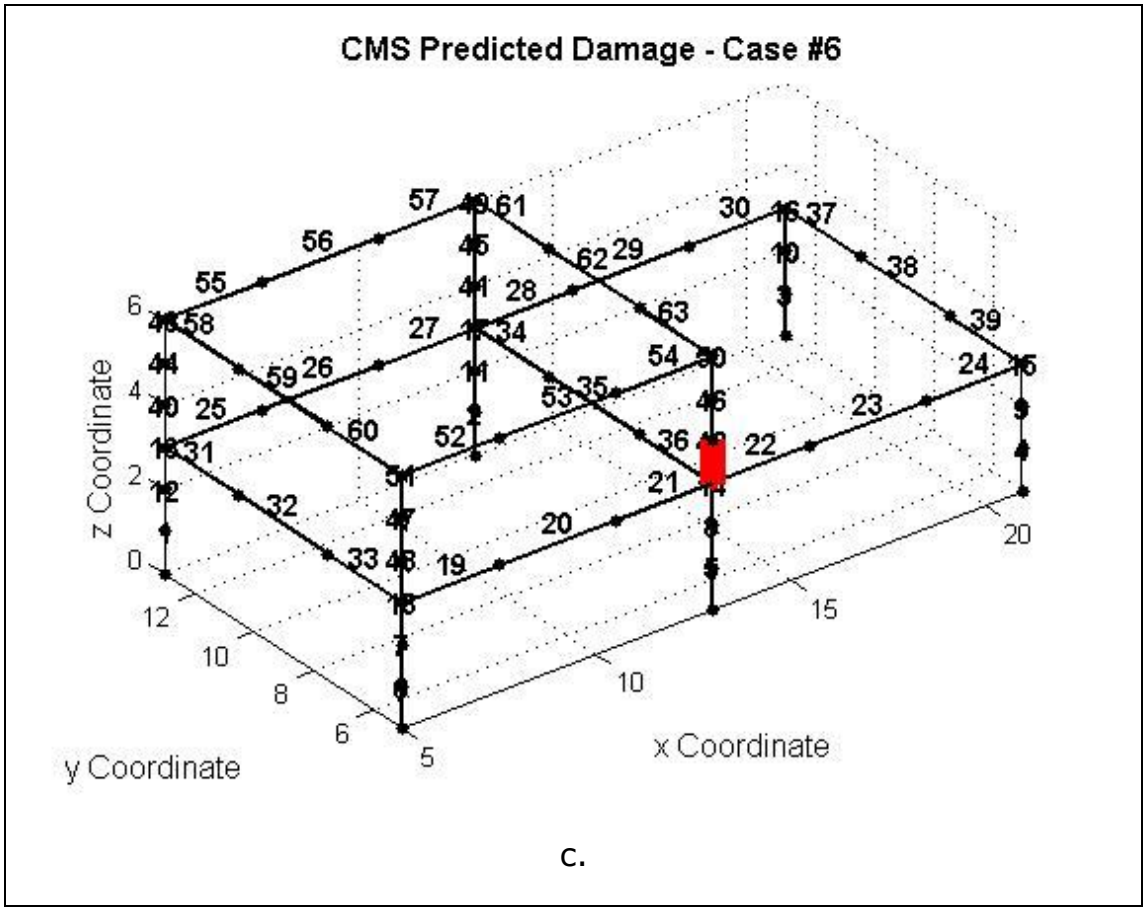
Figure 6.32 a. Damage identification implementing Discrete Value algorithm and Component Mode Synthesis case #5 b. Actual damage c. Predicted damage


DV algorithm & Component Mode Synthesis - Case #6
% Decrease in Stiffness
Elements
Actual damage
Predicted damage

a.

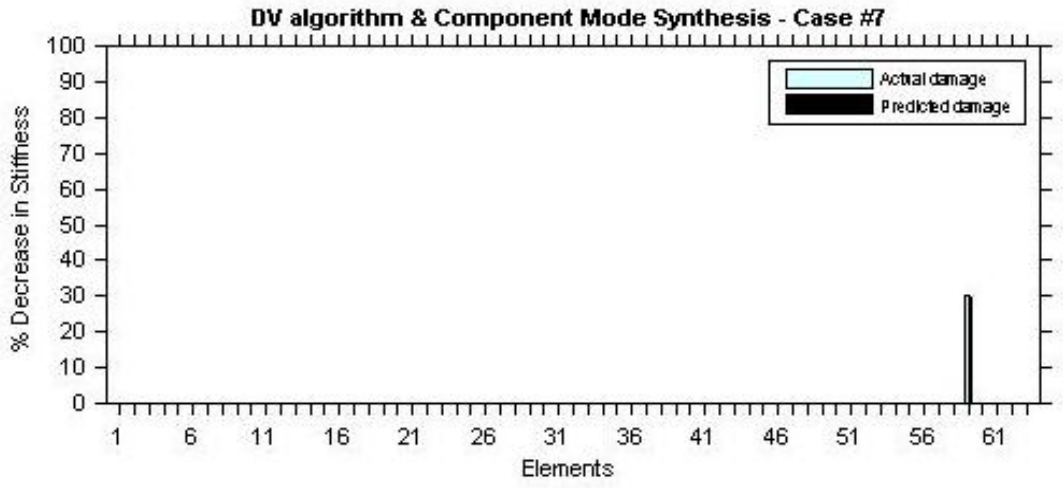

Actual Damage - Case #6
z Coordinate
y Coordinate
x Coordinate

b.

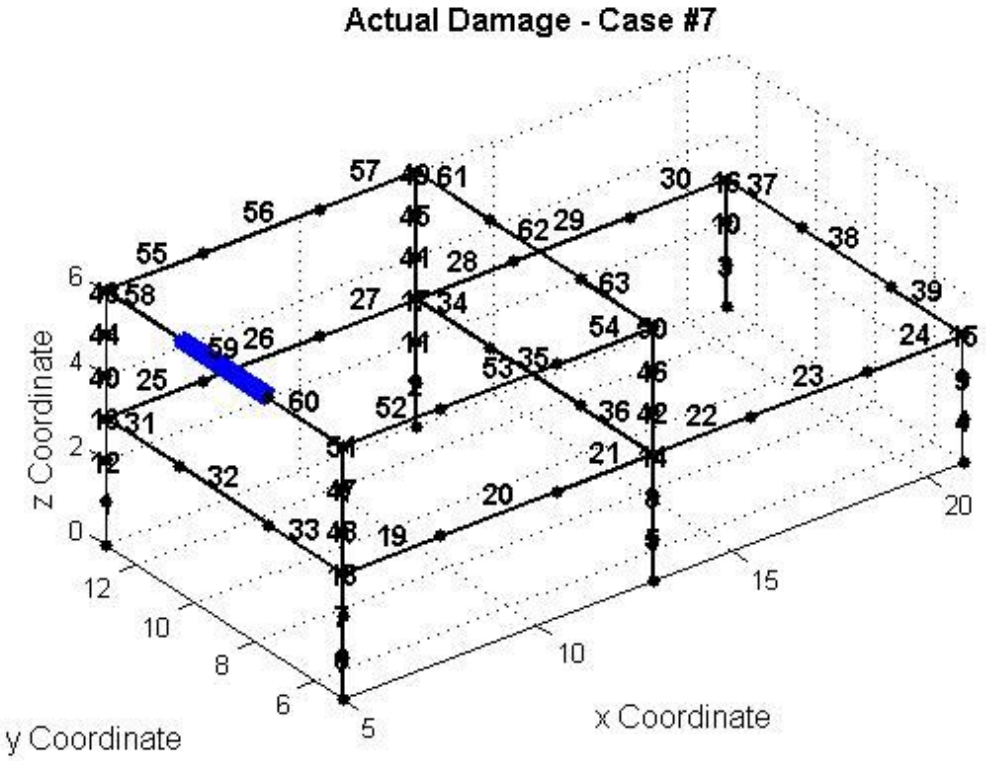


c.

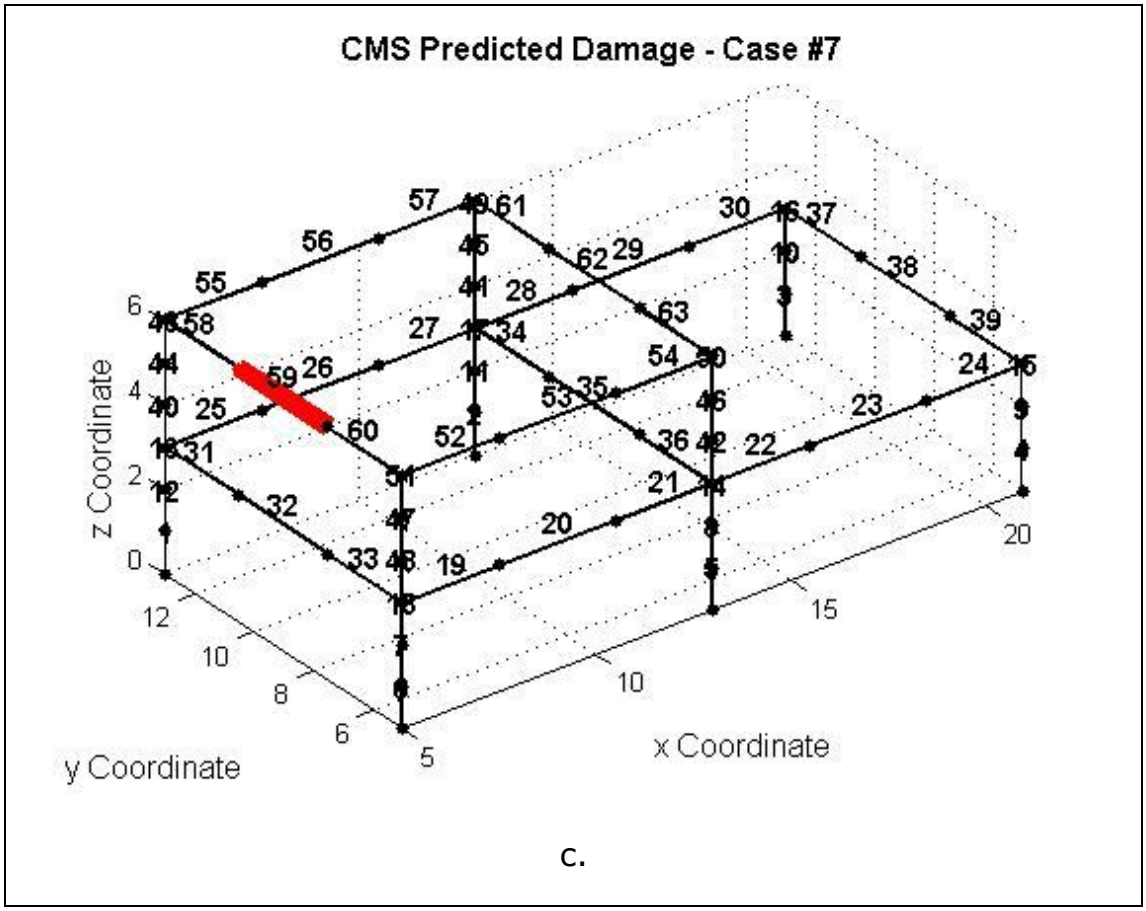
Figure 6.33 a. Damage identification implementing Discrete Value algorithm and Component Mode Synthesis case #6 b. Actual damage c. Predicted damage


DV algorithm & Component Mode Synthesis - Case #7
% Decrease in Stiffness
Elements
Actual damage
Predicted damage
0
10
20
30
40
50
60
70
80
90
100
1
6
11
16
21
26
31
36
41
46
51
56
61

a.

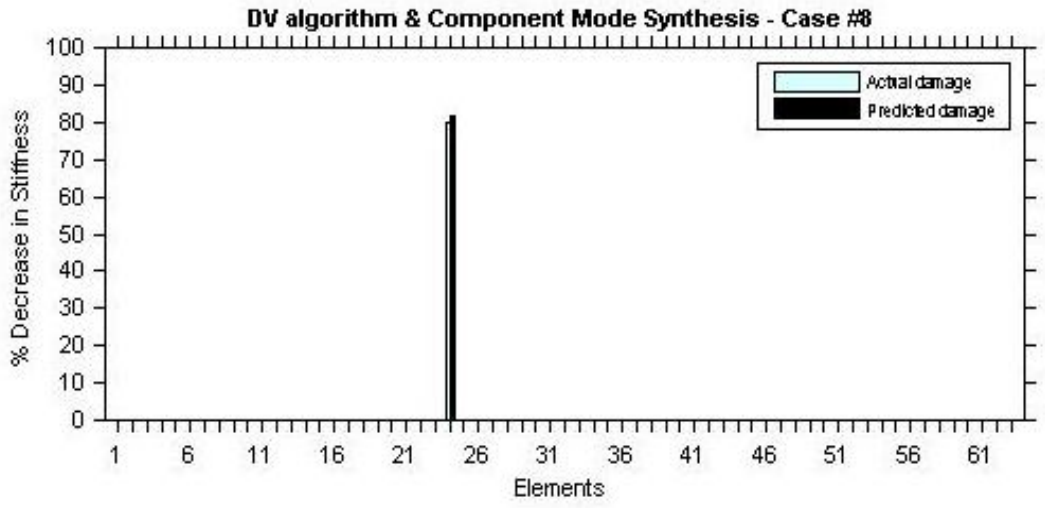

Actual Damage - Case #7
z Coordinate
y Coordinate
x Coordinate
0
2
4
6
12
10
8
6
5
10
15
20

b.

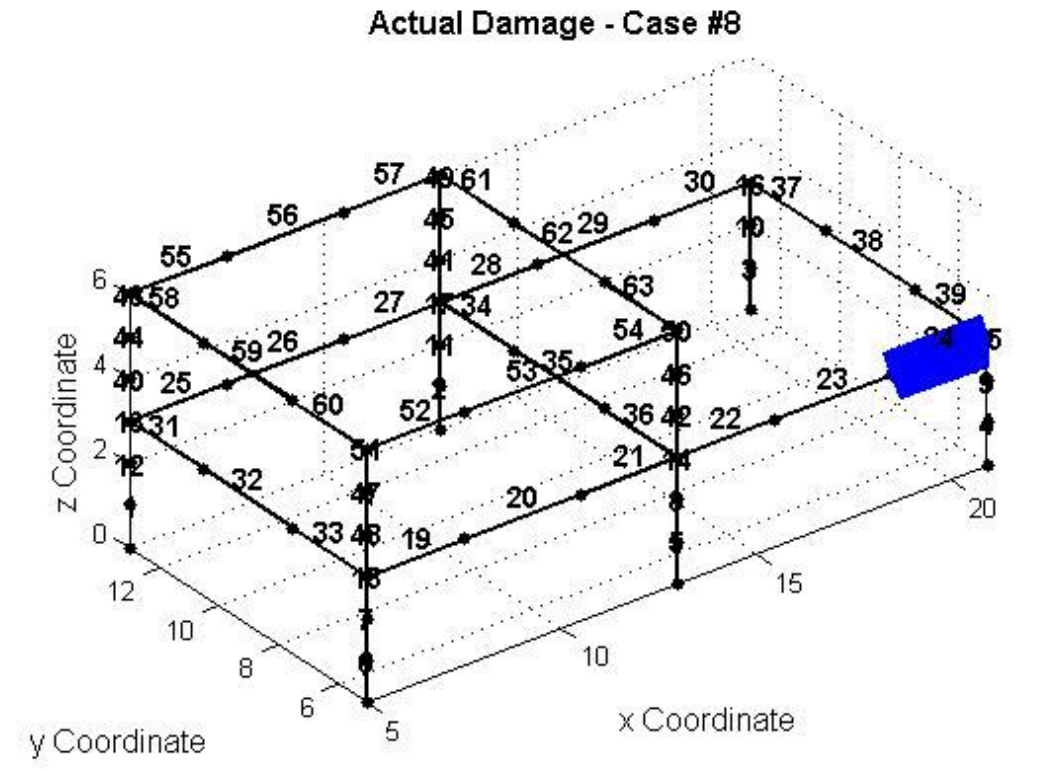


c.

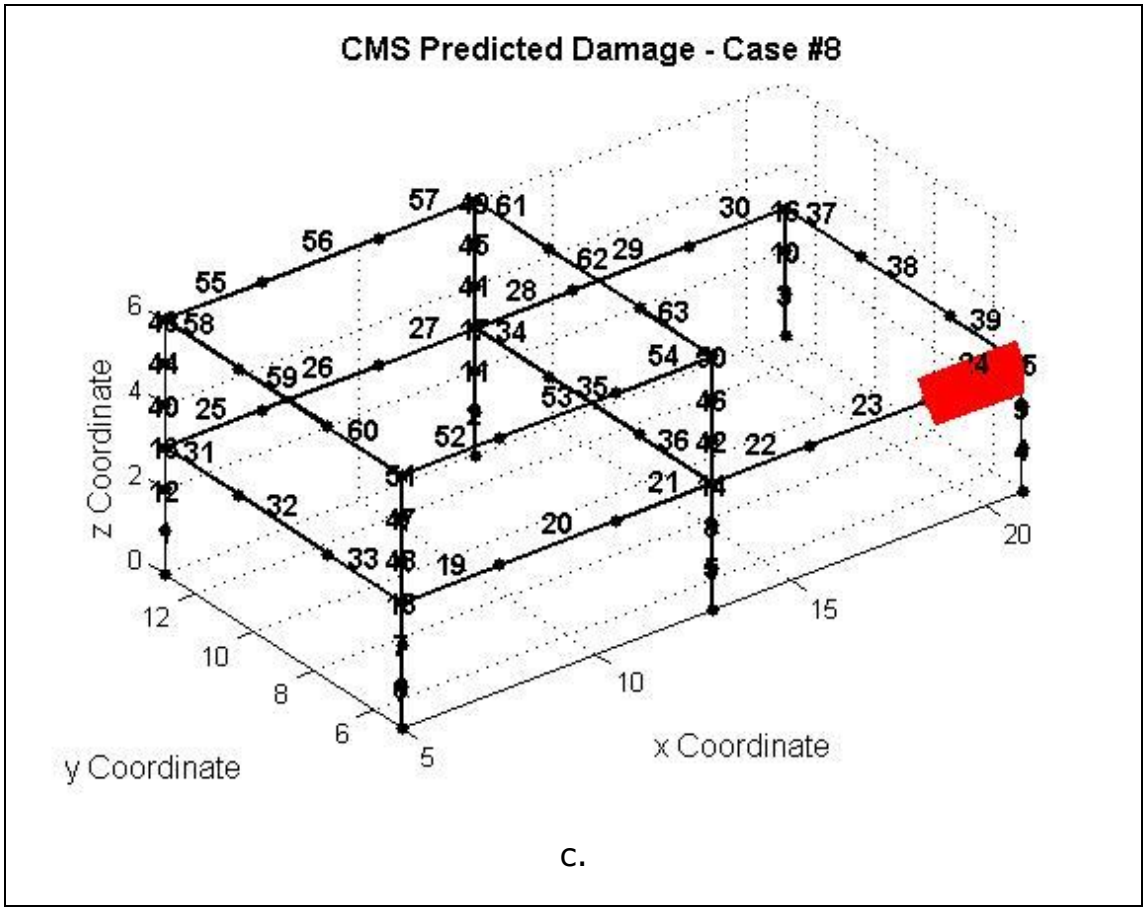
Figure 6.34 a. Damage identification implementing Discrete Value algorithm and Component Mode Synthesis case #7 b. Actual damage c. Predicted damage


DV algorithm & Component Mode Synthesis - Case #8
% Decrease in Stiffness
Elements
Actual damage
Predicted damage

a.

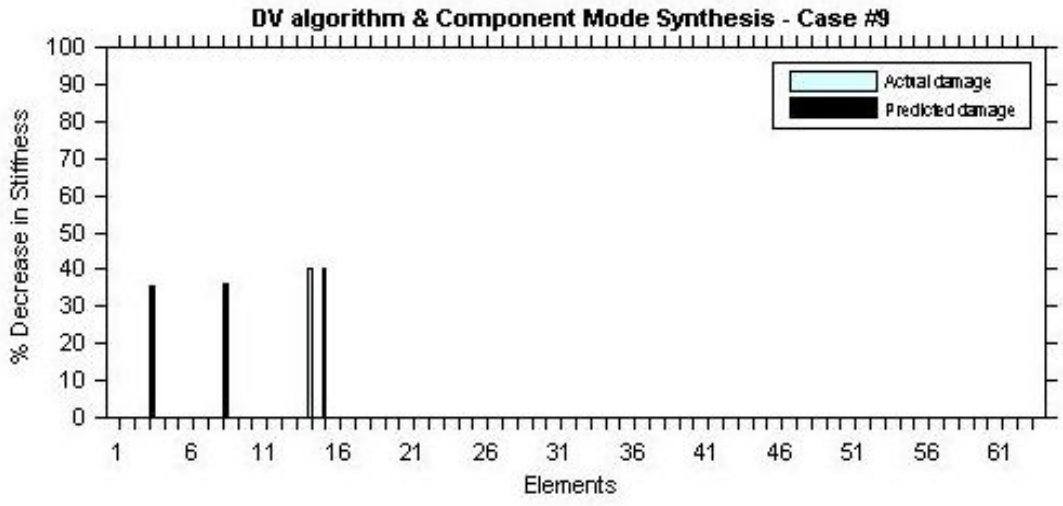

Actual Damage - Case #8
z Coordinate
y Coordinate
x Coordinate

b.

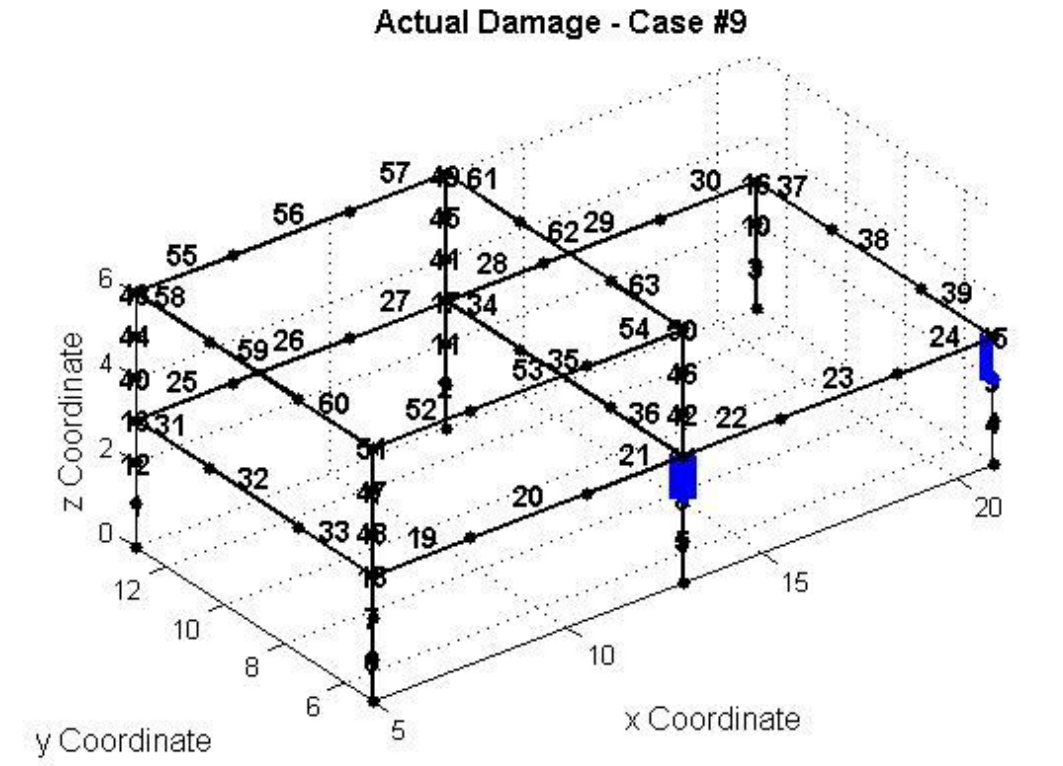


c.

Figure 6.35 a. Damage identification implementing Discrete Value algorithm and Component Mode Synthesis case #8 b. Actual damage c. Predicted damage


DV algorithm & Component Mode Synthesis - Case #9
Actual damage
Predicted damage
% Decrease in Stiffness
Elements

a.


Actual Damage - Case #9
z Coordinate
y Coordinate
x Coordinate

b.

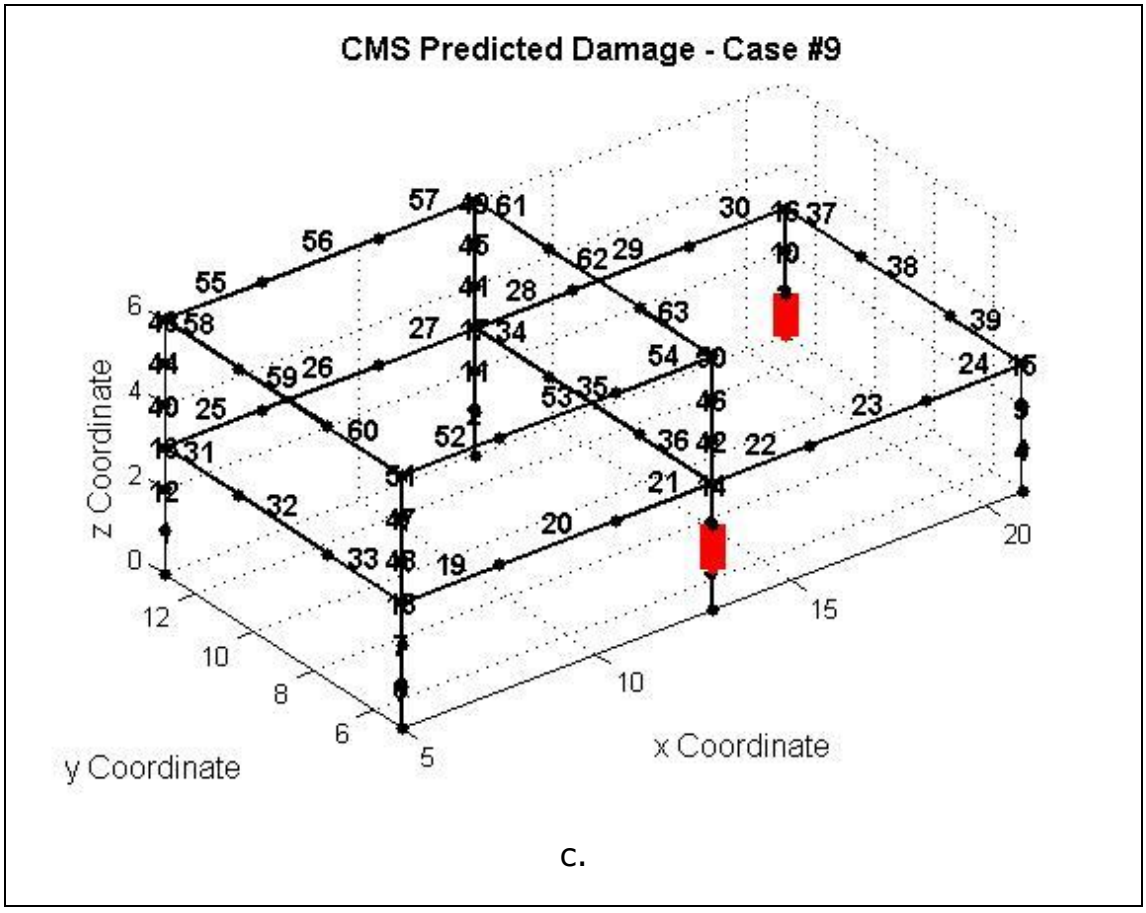
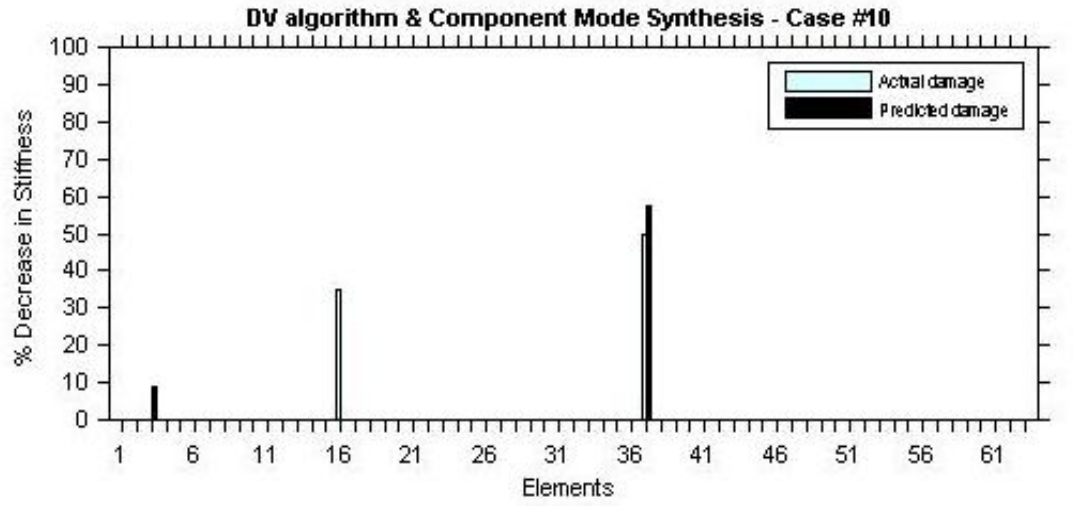
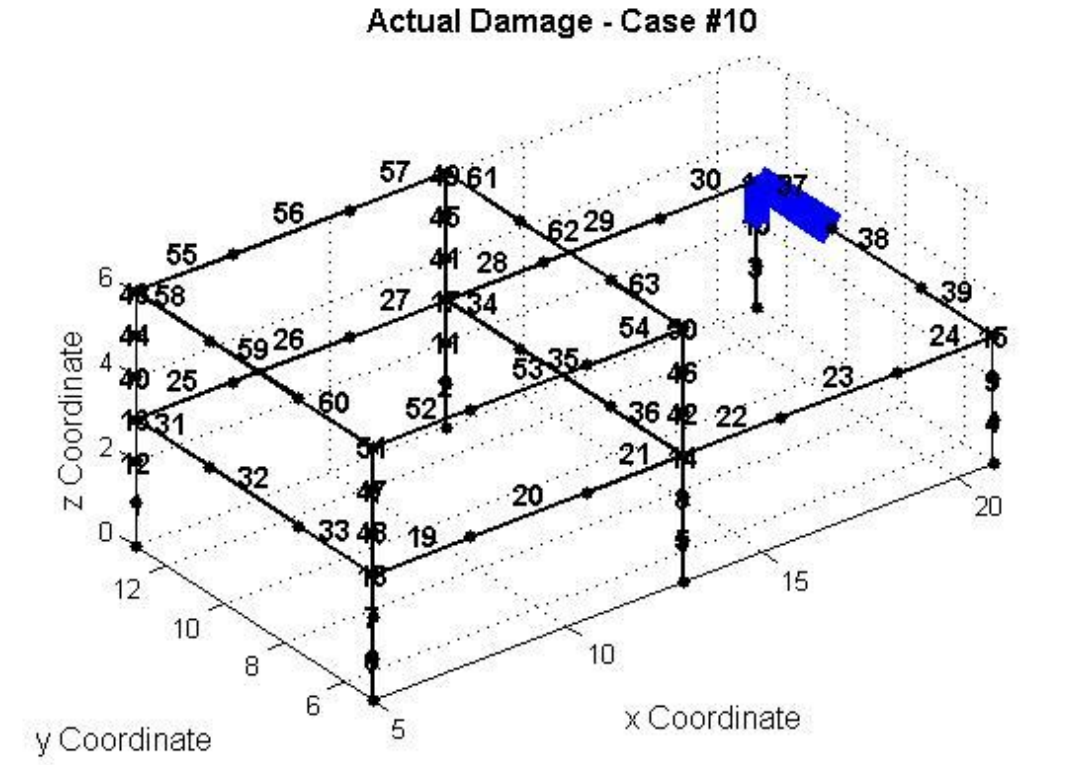


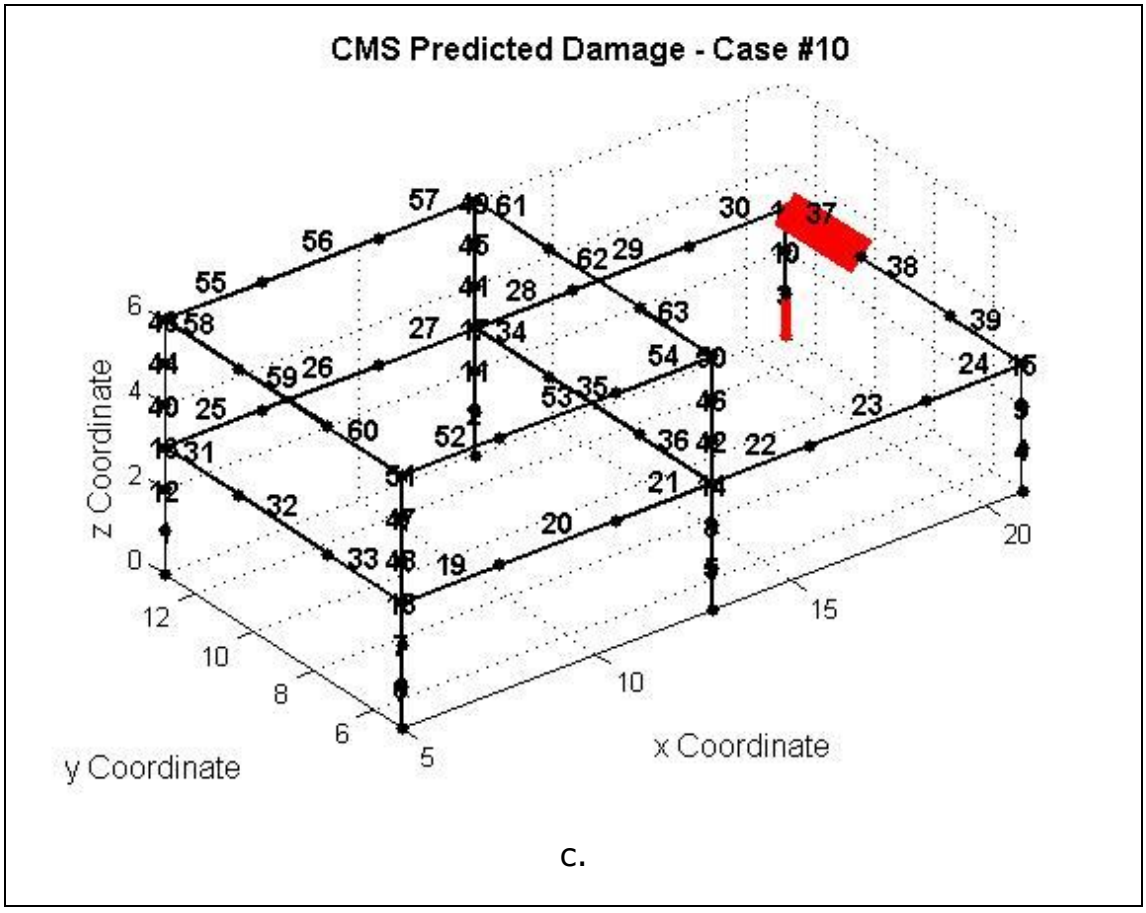
Figure 6.36 a. Damage identification implementing Discrete Value algorithm and Component Mode Synthesis case #9 b. Actual damage c. Predicted damage


DV algorithm & Component Mode Synthesis - Case #10
% Decrease in Stiffness
Elements
Actual damage
Predicted damage

a.

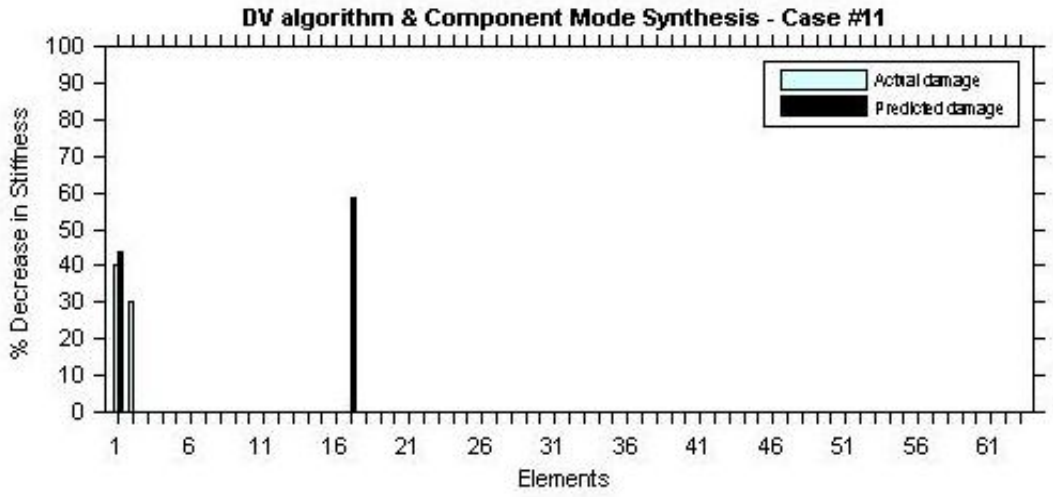

Actual Damage - Case #10
z Coordinate
y Coordinate
x Coordinate

b.

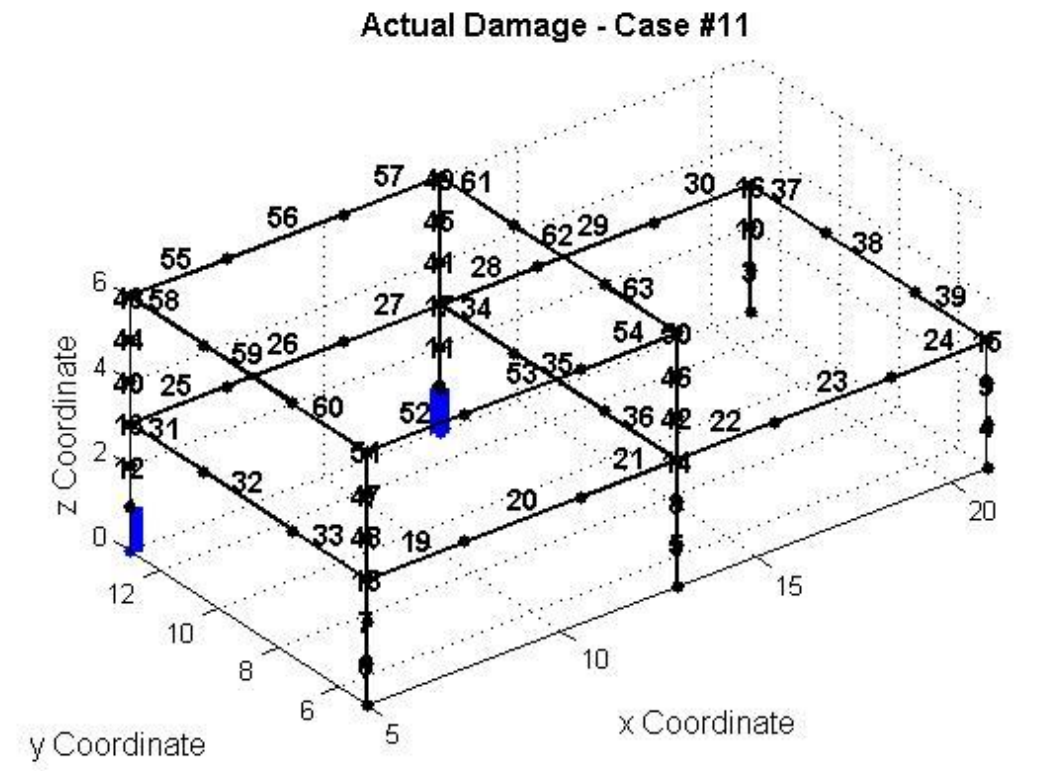


c.

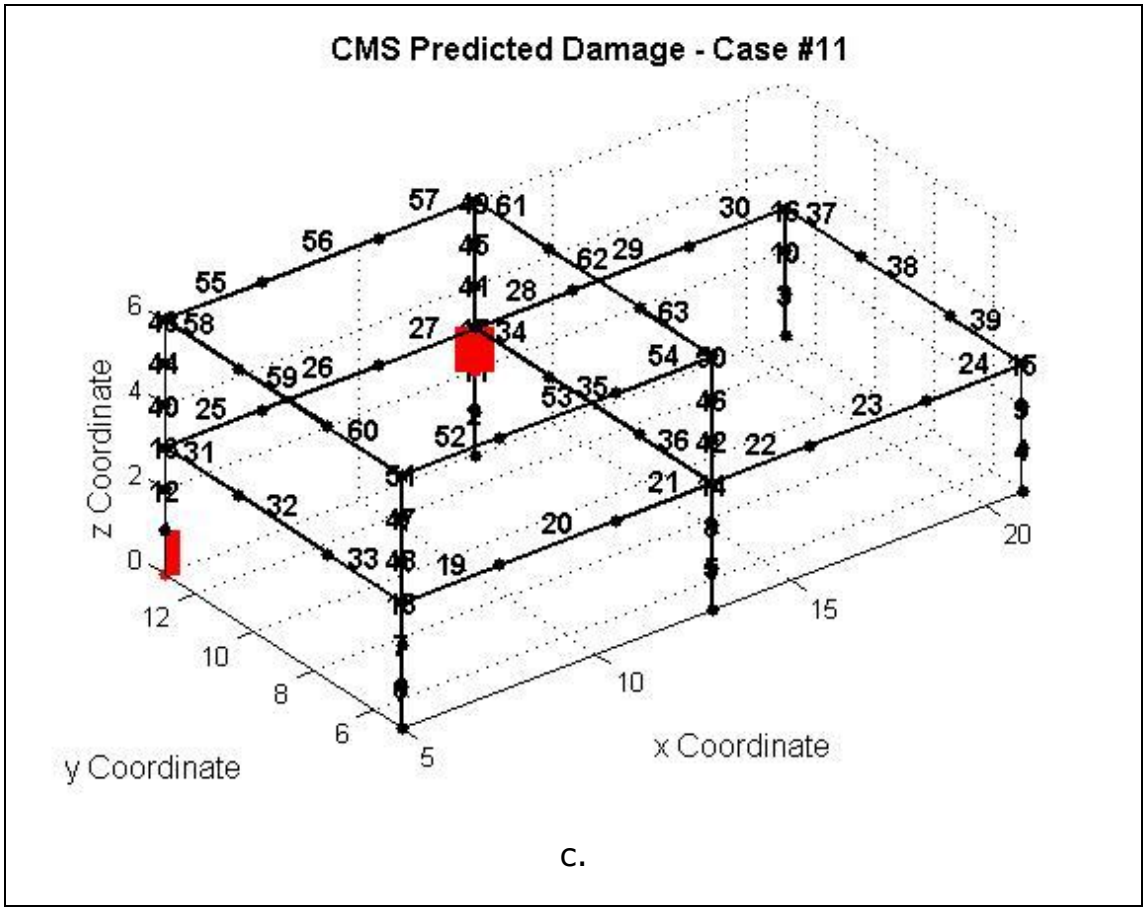
Figure 6.37 a. Damage identification implementing Discrete Value algorithm and Component Mode Synthesis case #10 b. Actual damage c. Predicted damage


DV algorithm & Component Mode Synthesis - Case #11
Actual damage
Predicted damage
% Decrease in Stiffness
Elements

a.


Actual Damage - Case #11
z Coordinate
y Coordinate
x Coordinate

b.



c.

Figure 6.38 a. Damage identification implementing Discrete Value algorithm and Component Mode Synthesis case #11 b. Actual damage c. Predicted damage

Conclusions

At the present study damage identification from measured eigenfrequencies using metaheuristic algorithms was implemented. As it was shown here and in numerous other studies, damage identification is possible. At the author's opinion as the problem is sensitive, certain requirements should be met. First, the stiffness matrix should represent the structure adequately that is the nonlinear effects should be insignificant (or properly introduced in the model) and we should have accurate information regarding the geometry and the materials of the structure. Of course the boundary conditions are very important and should be properly handled. Finally the distribution of the mass is crucial as it can totally alter the results.

Assuming the above requirements are met, metaheuristic algorithms can give possible damage scenario but as the size of the problem grows (larger models with many elements) the computer power requirement rises. The challenge is to find appropriate algorithms coupled with simplification and dynamic reduction of each problem.

Finally, a mathematical approach of the problem is needed in order to clarify uniqueness issues and the minimum number of damaged elements for a set of given eigenvalues.