Trans. Korean Soc. Noise Vib. Eng., 29(3) : 327~346, 2019
https://doi.org/10.5050/KSNVE.2019.29.3.327

한국소음진동공학회논문집 제 29 권 제 3 호, pp. 327~346, 2019
ISSN 1598-2785(Print), ISSN 2287-5476(Online)

Development of Structural Displacement Finite Element Codes Based on Physical Coordinates and High Performance Computing Methods Including PARDISO, PCG, GMRES and MATLAB or OCTAVE Intrinsic Operator '\'

물리적 좌표기반 구조 변위해석 유한요소 코드 및 PARDISO, PCG, GMRES와 MATLAB 또는 OCTAVE 연산자 '\' 를 포함하는 여러 고성능 계산기법 개발

Seok-Tae Park†
박 석 태†

(Received February 14, 2019 ; Revised April 23, 2019 ; Accepted April 23, 2019)

Key Words : Analytical Integration(해석적 적분), CG(Conjugate Gradient, 켤레 기울기법), Direct Solver(직접 해법), GMRES(Generalized Minimal RESidual, 일반화된 최소 잔차법), FGMRES(Flexible Generalized Minimal RESidual, 유연한 일반화된 최소 잔차법), Iterative Solver(반복 해법), Kirchhoff-Love Plate(키르히호프-러브 평판), Krylov Subspace Method(크릴로프 부 공간 법), MATLAB or OCTAVE Intrinsic Operator '\'(매트랩 또는 옥타브 내재 연산자 '\'), MGMRES(Multiple(Restarted) Generalized Minimal RESidual, 다중 일반화된 최소잔차법), Mindlin-Reissner Plate(민들린-라이즈너 평판), PARDISO(PARallel DIrect Sparse sOlver, 병렬 직접 희박 해법), PCG(Preconditioned Conjugate Gradient, 선조건 적용 켤레 기울기법), Physical Coordinates Based FEM(물리적 좌표기반 유한요소법), Sparse Matrix Solver(희박행렬 해법)

ABSTRACT

This paper presents physical coordinate-based 3-node triangular, 4-node rectangular, and 4-node quadrilateral FE codes on the Kirchhoff thin plate theory. Isoparametric 4-node quadrilateral and 4-node plate shell FE codes based on natural coordinates are also presented. The calculation time needed to solve a system equation was compared using direct, iterative, and full matrix and sparse matrix solvers including Gauss elimination method, CG, DGESV, DSGESV, SOR, PCG, GMRES, FGMRES, MGMRES, PARDISO, etc. The full matrix and the sparse matrix were applied to compare the computing time of the system equation using OCTAVE or the MATLAB intrinsic operator '\'. The computation times for PCG methods using four types of preconditioners were also compared. The computation time of the finite element method is compared with that of the PCG method after processing the system matrix with the full matrix and sparse matrix forms. We showed that PARDISO and sparse MATLAB or OCTAVE operator '\' could be useful and effective methods for conducting large-scale finite element analyses using personal computers.

† Corresponding Author ; Member, Dept. of Academic Affairs, Chungbuk Health and Science University
E-mail : stpark@chsu.ac.kr

A part of this paper was presented at the KSNVE 2019 Annual Spring Conference.

‡ Recommended by Editor Won Ju Jeon

요 약

이 논문에서는 키르히호프 평판 이론에 기초하여 물리적 좌표기반 3절점 삼각형 유한요소, 4절점 직사각형 유한요소, 4절점 사각형 유한요소 코드들을 제시하였다. Natural coordinates에 기반한 isoparametric 4절점 사각형 유한요소 코드와 4절점 plate shell 유한요소 코드도 제시하였다. 직접법, 반복법 및 full matrix 해법들과 희박행렬 해법들을 이용하여 시스템 방정식을 푸는데 필요한 계산시간을 비교하였다. Gauss 소거법, CG, DGESV, DSGESV, SOR, PCG, GMRES, FGMRES, MGMRES, PARDISO 법 등을 비교 검토하였다. OCTAVE와 MATLAB에 내장된 연산자 '\'를 full matrix 형태와 희박행렬 형태 시스템 방정식에 적용하여 계산시간을 비교하였다. 4가지 종류의 preconditioners를 채택한 PCG 방법들에 대한 계산시간도 비교 검토하였다. 또한, 유한요소법에서 시스템 행렬을 full matrix와 희박행렬로 처리한 후에 각각의 경우에 PCG 방법을 적용했을 때의 계산시간도 비교하였다. PC에서도 PARDISO와 sparse MATLAB 연산자 '\'를 사용하여 대규모의 유한요소 해석을 수행할 수 있음을 보였다.

1. History and Research Motivation of Thin Plate Analysis

In this paper, the development of structural displacement finite element method (FEM) codes and the computational time comparisons of system equation using several solvers were described. For the first topic, we dealt with the Kirchhoff flat plate theory in Section 2, and described the formulations of the finite element (FE) in Section 3. We verified the validity of the developed FE codes in Section 4. As a second theme, in Section 5, we compared the computation times obtained by various solving methods of the system equation within structural static displacement FE codes, and proposed an appropriate analysis methods. Acoustic BEM (boundary element method) code and structural vibration FEM code were needed to perform structural-acoustical coupling analysis by developing high-speed computations of large degrees of freedom without using commercial software.

The reason for developing the structural displacement analysis FEM program was that Holmström[1] omitted the FEM program for vibration analysis with blackbox in the program presented in the vibration-acoustic coupled analysis paper. (1) In this paper, the first developed FEM codes were the same as rectangular FE codes presented by Irvine[2~4]. It was a physical coordinate based rectangular FE codes that could be applied only when the four sides were all parallel to the x or y axis[2], developed according to the assumptions used in the Kirchhoff plate theory (thin plate theory). (2) Semie suggested a physical coordinate based 3-node triangular plate FE codes that could model arbitrary two-dimensional shapes and analyzed the displacement of a thin plate[5]. He investigated two cases where four sides of the square plate were simply supported or fixed (clamped). However, the analytical results were significantly deviated from the values of the commercial software ANSYS® and Kirchhoff thin plate's theory. In this paper, we developed a physical coordinate based triangular FE codes according to the methodology suggested by Semie, and found that there would be some errors in the FE codes presented by Semie. (3) We have also developed a physical coordinate based 4-node quadrilateral FE codes that can model any two-dimensional shape in order to complement the disadvantages of FEM using rectangular FEs parallel to the x- or y-axis. (4) On the other hand, the above three FE codes were assumed to ignore transverse shear strain effects because of the Kirchhoff thin plate theory. Ferreira proposed an isoparametric quadrilateral FE codes based on the Mindlin-Reissner plate theory, which considered transverse shear deformation[6]. We also developed the natural coordinates based isoparametric quadrilateral FE codes which were based on the thick plate theory, and compared with the displacement

analysis results of the Kirchhoff plate theory. (5) The displacement analysis FEM codes based on the Kirchhoff plate and Mindlin-Reissner plate theories were able to only analyze the z-direction displacement in the normal direction of the 2-D coplanar plate. Kwon proposed a 3-dimensional isoparametric plate shell FE codes for static displacement analysis in 3-D space using MATLAB® language[7].

We have translated this codes into FORTRAN ones to improve memory scalability and computational speed. Consequently, four 2-D coplanar structural FE codes and one 3-D structural displacement FE codes were described in this paper. The differences between Kirchhoff theory, Mindlin-Reissner plate theory and three dimensional plate shell element were described for two dimensional plate. In order to evaluate the performance of FEM system equation, which was a system matrix equation $Ax = b$, the computation times of MATLAB®, OCTAVE® and FORTRAN codes were compared in both direct and iterative methods. We also compared the computation time for full matrix solvers that processed the coefficient matrix A of system equation as a full matrix form and sparse matrix solvers for the sparse matrix one. The computation times for PCG methods using four kinds of preconditioners were also compared in sparse system matrix. The finite element method was compared with the computation time of PCG method after processing the system matrix with full matrix and sparse matrix form, respectively.

2. Kirchhoff Thin Plate Theory

According to Semie[5], in 1776 Euler first attempted to solve the free vibration problem of the plate. Germain developed differential equations for flat plates without considering warping term, and Lagrange proposed general plate differential equations by considering warping term. Poisson proposed the Germain-Lagrange plate equation assuming a constant flexural rigidity for the plate subjected to static loads. Navier has established a flat bending theory that uses flexural rigidity as a function of plate thickness and proposed a method to obtain exact solution using Fourier trigonometric series. Kirchhoff proposed the theory of plate bending using basic assumptions known as 'Kirchhoff's hypothesis.' Kirchhoff's thin plate's hypothesis is expressed: the cross sections of the plate are remained as straight and unstretched and also normal to midsurface even after the plate is deformed. That is, $\gamma_{xz} = 0$, $\gamma_{yz} = 0$, $\sigma_{zz} = 0$ and $\varepsilon_{zz} = 0$. Let the reference in the z direction be the middle of the plate thickness, and the displacements in the x and y directions in the plate u, v respectively, and the displacement in the z direction w (Fig. 1).

$$u = -z\theta_x = -z\frac{\partial w}{\partial x} \tag{1}$$

$$v = -z\theta_y = -z\frac{\partial w}{\partial y} \tag{2}$$

For inplane strains, strain-displacement relationships:

$$\varepsilon_x = \frac{\partial u}{\partial x} = -z\frac{\partial^2 w}{\partial x^2} \tag{3}$$

$$\varepsilon_y = \frac{\partial v}{\partial y} = -z\frac{\partial^2 w}{\partial y^2} \tag{4}$$

$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} = -2z\frac{\partial^2 w}{\partial x \partial y} \tag{5}$$

The relationship between plane stress and plane strain for isotropic materials according to Kirchhoff's assumptions is as follows. Stress-strain relationships:

$$\sigma_x = \frac{E}{1-\upsilon^2}(\varepsilon_x + \upsilon\varepsilon_y) \tag{6}$$

$$\sigma_y = \frac{E}{1-\upsilon^2}(\varepsilon_y + \upsilon\varepsilon_x) \tag{7}$$

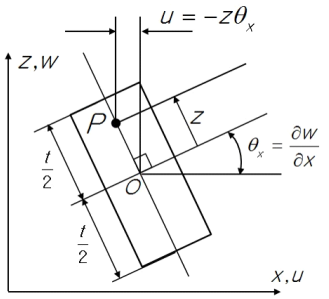

Fig. 1 Slice cut of plate normal to y axis

$$\tau_{xy} = G\gamma_{xy} \tag{8}$$

Where, shear modulus $G = \frac{E}{2(1+\upsilon)}$, Young's modulus E, Poisson's ratio υ

Bending moments[7] along the x, y, and xy edge (Fig. 2):

$$M_x = \int_{-t/2}^{t/2} z\sigma_x dz = -D\left(\frac{\partial^2 w}{\partial x^2} + \upsilon\frac{\partial^2 w}{\partial y^2}\right) \tag{9}$$

$$M_y = \int_{-t/2}^{t/2} z\sigma_y dz = -D\left(\frac{\partial^2 w}{\partial y^2} + \upsilon\frac{\partial^2 w}{\partial x^2}\right) \tag{10}$$

$$M_{xy} = \int_{-t/2}^{t/2} z\tau_{xy} dz = -D(1-\upsilon)\frac{\partial^2 w}{\partial x \partial y} \tag{11}$$

Where, bending rigidity of the plate $D = \frac{Et^3}{12(1-\upsilon^2)}$ and t is thickness of the plate.

By the moment equilibrium,

$$\frac{\partial Q_x}{\partial x} + \frac{\partial Q_y}{\partial y} + q = 0 \tag{12}$$

$$\frac{\partial M_x}{\partial x} + \frac{\partial M_{xy}}{\partial y} - Q_x = 0 \tag{13}$$

$$\frac{\partial M_y}{\partial y} + \frac{\partial M_{xy}}{\partial x} - Q_y = 0 \tag{14}$$

Inserting Eq. (9) ~ Eq. (11) into Eq. (13) and Eq. (14),

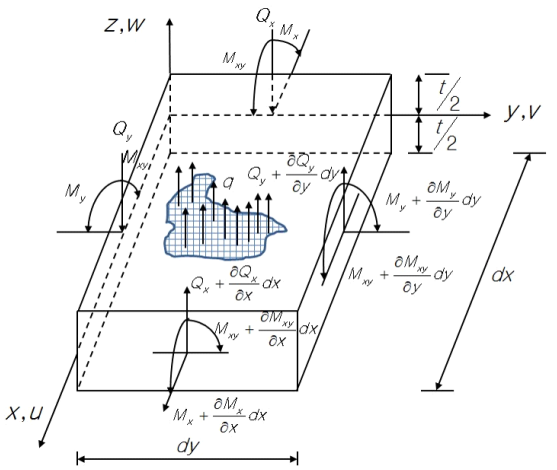

Fig. 2 Free body diagram of the plate element

after $\frac{\partial(\text{Eq.}(13))}{\partial x} + \frac{\partial(\text{Eq.}(14))}{\partial y} + (\text{Eq.}(12))$:

$$D\left(\frac{\partial^4 w}{\partial x^4} + 2\frac{\partial^4 w}{\partial x^2 \partial y^2} + \frac{\partial^4 w}{\partial y^4}\right) = q \tag{15}$$

This is the governing equation for Kirchhoff thin plate bending theory with transversely loaded.

2.1 Analytical Solution of Thin Plate

Boundary conditions for a simple supported rectangular plate: width and length of the plate are a and b, respectively.

$$W = 0, \quad \frac{\partial^2 w}{\partial x^2} = 0, \quad \text{at } x = 0, a$$

$$W = 0, \quad \frac{\partial^2 w}{\partial y^2} = 0, \quad \text{at } y = 0, b$$

Using the infinite double Fourier trigonometric series satisfying the boundary condition in the method proposed by Navier[5,8], out of plate displacement $w(x, y)$ can be expressed as Eq. (16).

$$w(x,y) = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty} w_{mn} \sin\frac{m\pi x}{a}\sin\frac{m\pi y}{b} \tag{16}$$

For sinusoidal load q,

$$q = p(x,y) = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty} p_{mn} \sin\frac{m\pi x}{a}\sin\frac{m\pi y}{b} \tag{17}$$

Where,

$$p_{mn} = \frac{4}{ab}\int_0^a\int_0^b p(x,y)\sin\frac{m\pi x}{a}\sin\frac{n\pi y}{b}dxdy \tag{18}$$

Equation (16) and Eq. (17) can be inserted into Eq. (15) and solved $w(x, y)$, but Levy[8] proposed a simple series method and got the maximum displacement at the center of the square plate ($a = b$).

$$w_{\max} = 0.00406\frac{p_0 a^4}{D} \tag{19}$$

The displacement due to concentrated load P at the center of the square plate is followed[8].

$$w_{\max} = 0.01160\frac{Pa^2}{D} \tag{20}$$

The maximum displacements at the center of the square plate with respect to the distributed loads p_0 and concentrated loads P for the fixed boundary conditions are given by Eq. (21) and Eq. (22)[8].

$$w_{\max} = 0.00126\frac{p_0 a^4}{D} \quad \text{for } p_0 \tag{21}$$

$$w_{\max} = 0.00560\frac{Pa^2}{D} \quad \text{for } P \tag{22}$$

3. Formulations of Various Structural Displacement Finite Elements

In this paper, three-node triangular, four-node rectangular and quadrilateral FE codes based on physical coordinates and Kirchhoff thin plate theory were presented and also natural coordinates based isoparametric four-node quadrilateral FE codes according to Reissner-Mindlin thick plate theory and 3-D plate shell FE codes. According to Kirchhoff's thin plate theory, the transverse shear strain energy is negligible compared to the bending energy. Therefore, the total strain energy of the plate[2] is as Eq. (23).

$$\begin{aligned} U &= \frac{1}{2}\int_v \{\sigma\}^t\{\varepsilon\}dv \\ &= \frac{D}{2}\iint \begin{bmatrix} \left(\frac{\partial^2 w}{\partial x^2}\right)^2 + \left(\frac{\partial^2 w}{\partial y^2}\right)^2 \\ +2\upsilon\left(\frac{\partial^2 w}{\partial x^2}\right)\left(\frac{\partial^2 w}{\partial y^2}\right) \\ +2(1-\upsilon)\left(\frac{\partial^2 w}{\partial x \partial y}\right)^2 \end{bmatrix} dxdy \end{aligned} \tag{23}$$

$$\{\sigma\}^t = \begin{bmatrix} \sigma_x & \sigma_x & \tau_{xy} \end{bmatrix} \tag{24}$$

$$\{\varepsilon\}^t = \begin{bmatrix} \varepsilon_x & \epsilon_x & \gamma_{xy} \end{bmatrix} \tag{25}$$

Total potential energy Π, potential energy of external forces V

$$\Pi = U - V \tag{26}$$

Where, $V = \int_v w^t f dv$

3.1 Triangular Finite Element Based on Physical Coordinates

Batoz proposed a DKT (discrete Kirchhoff theory) element[9] with the origin of the local coordinate of the geometric centroid of the triangle. But, in this paper, the origin of the local coordinates was defined as in Fig. 3. In Fig. 3, let the longest side of the triangular element coincide with the x-axis of the local coordinates. The local triangle with new coordinate values was constructed by translational and rotational movement so that the vertex facing the longest side was in the first quadrant. FE codes using 9 degrees of freedom (DOF) triangular element with 3 DOF per the triangular node were developed. The node numbering order was counterclockwise. Local FEs constructed from local coordinates were converted to their original global coordinates by rotational movement. The global total stiffness matrix was constructed by assembling each global FEs. W was out of plane displacement in z coordinate and was expressed by physical coordinates like as Eq. (27)[5].

$$\begin{aligned} w(x,y,t) &= a_1(t) + a_2(t)x + a_3(t)y + a_4(t)x^2 \\ &\quad + a_5(t)xy + a_6(t)y^2 + a_7(t)x^3 \\ &\quad + a_8(t)(x^2y + xy^2) + a_9(t)y^3 \\ &= \{A(t)\}^t\{Z\} \end{aligned} \tag{27}$$

$$\{A(t)\}^t = \begin{bmatrix} a_1(t) & \cdots a_9(t) \end{bmatrix} \tag{28}$$

$$\{Z\}^t = \begin{bmatrix} 1 & x & y & x^2 & xy & y^2 & x^3 & x^2y + xy^2 & y^3 \end{bmatrix} \tag{29}$$

For node i,

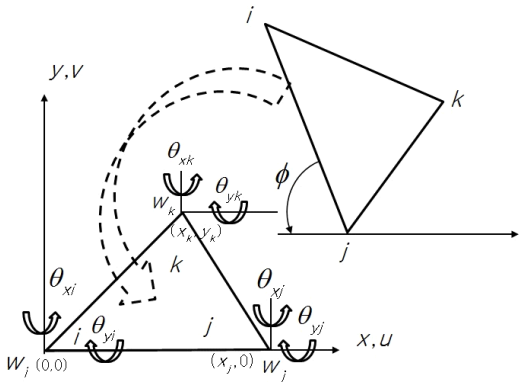

Fig. 3 Local and global coordinates

$$w(0,0)=w_i=a_1$$

$$\theta_{xi}=\partial w/\partial x=a_2,\quad \theta_{yi}=\partial w/\partial y=a_3$$

After applying the same processes for node j and node k,

$$\{w\}_e^t=\left[w_i\ \theta_{xi}\ \theta_{yi}\ w_j\ \theta_{xj}\ \theta_{yj}\ w_k\ \theta_{xk}\ \theta_{yk}\right] \tag{30}$$

$$\{w\}_e=[B]\{A\} \tag{31}$$

That is,

$$\begin{Bmatrix} w_i \\ \theta_{xi} \\ \theta_{yi} \\ w_j \\ \theta_{xj} \\ \theta_{yj} \\ w_k \\ \theta_{xk} \\ \theta_{yk} \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & x_j & 0 & x_j^2 & 0 & 0 & x_j^3 & 0 & 0 \\ 0 & 1 & 0 & 2x_j & 0 & 0 & 3x_j^2 & 0 & 0 \\ 0 & 0 & 1 & 0 & x_j & 0 & 0 & x_j^2 & 0 \\ 1 & x_k & y_k & x_k^2 & x_k y_k & y_k^2 & x_k^3 & x_k^2 y_k + x_k y_k^2 & y_k^3 \\ 0 & 1 & 0 & 2x_k & y_k & 0 & 3x_k^2 & 2x_k y_k + y_k^2 & 0 \\ 0 & 0 & 1 & 0 & x_k & 2y_k & 0 & x_k^2 + 2x_k y_k & 3y_k^2 \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \\ a_9 \end{Bmatrix} \tag{32}$$

$$\begin{aligned}\{A\} &= [B]^{-1}\{w\}_e \\ &= [C]\{w\}_e\end{aligned} \tag{33}$$

Where, $[C]=[B]^{-1}$

Equation (23) can be rewritten as Eq. (34).

$$\begin{aligned}U &= \frac{D}{2}\{w\}_e^t[C]^t\iint S(x,y)dxdy[C]\{w\}_e \\ &= \frac{1}{2}\{w\}_e^t[k]_e\{w\}_e\end{aligned} \tag{34}$$

Local element stiffness matrix $[k]_e$,

$$[k]_e = D[C]^t\iint S(x,y)dxdy[C] \tag{35}$$

$$\begin{aligned}S(x,y)_{9x9} &= \left\{\frac{\partial^2 Z}{\partial x^2}\right\}\left\{\frac{\partial^2 Z}{\partial x^2}\right\}^t+\left\{\frac{\partial^2 Z}{\partial y^2}\right\}\left\{\frac{\partial^2 Z}{\partial y^2}\right\}^t \\ &+\upsilon\left[\left\{\frac{\partial^2 Z}{\partial x^2}\right\}\left\{\frac{\partial^2 Z}{\partial y^2}\right\}^t+\left\{\frac{\partial^2 Z}{\partial y^2}\right\}\left\{\frac{\partial^2 Z}{\partial x^2}\right\}^t\right] \\ &+2(1-\upsilon)\left\{\frac{\partial^2 Z}{\partial x\partial y}\right\}\left\{\frac{\partial^2 Z}{\partial y\partial x}\right\}^t\end{aligned} \tag{36}$$

Where, $\beta=2(1-\upsilon)$

Analytical area integration (Fig. 4) for S_{47} = $12x$,

$$\begin{aligned}\iint S_{47}dxdy &= \int_0^{y_k}\int_{y/p_1}^{(y-q)/p_2}12xdxdy \\ &= 2y_k\left(3q^2-3qy_k+y_k^2\right)/p_2^2-2y_k^3/p_1^2\end{aligned}$$

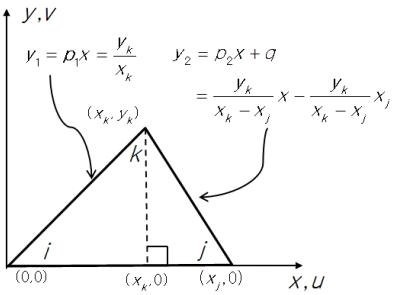

Fig. 4 Analytical integration

Where, $p_1=y_k/x_k,\ p_2=y_k/(x_k-x_j),\ q=-x_jp_2$

Global element stiffness matrix $[k]_g$ was obtained like as follows.

$$[k]_g=[T]^t[k]_e[T] \tag{37}$$

Where, rotation matrix $[T]=\begin{bmatrix} T_1 & 0 & 0 \\ 0 & T_1 & 0 \\ 0 & 0 & T_1 \end{bmatrix}$

$$T_1=\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(-\phi) & -\sin(-\phi) \\ 0 & \sin(-\phi) & \cos(-\phi) \end{bmatrix}$$

By variational approaches for Eq. (26),

$$[k]_g\{w\}_g=\{f\}_g \tag{38}$$

Total global stiffness matrix, displacement vector and force vector were followed, respectively.

$$[K]=\sum[k]_g \tag{39}$$

$$\{W\}=\sum\{w\}_g \tag{40}$$

$$\{F\}=\sum\{f\}_g \tag{41}$$

System equation was expressed as Eq. (42).

$$[K]\{W\}=\{F\} \tag{42}$$

3.2 Rectangular Finite Element Based on Physical Coordinates

According to Tom Irvine[2], the out of displacement W can be expressed as physical coordinates as 4 DOF at each of four vertices of a rectangle.

$$\begin{aligned}w(x,y,t) &= a_1(t)+a_2(t)x+a_3(t)y+a_4(t)x^2 \\ &+a_5(t)xy+a_6(t)y^2+a_7(t)x^3 \\ &+a_8(t)x^2y+a_9(t)xy^2 \\ &+a_{10}(t)y^3+a_{11}(t)x^3y+a_{12}(t)xy^3\end{aligned} \tag{43}$$

The FE could be obtained in the same way as Sec. 3.1. For $S_{47} = 12x$,

$$\iint S_{47}dxdy = \int_0^b \int_0^a 12xdxdy = 6a^2b$$

Where, a and b was side length of rectangular element.

Analytical area integration was simple compared to triangular FE, but rectangular FE suggested by Irvine could be applied only if all sides were parallel to x or y axis. However, in this paper, even though the local rectangular element was not parallel to the x or y axis, we have modified the FE codes so that it could be applicable to any orientation of rectangle FE by applying the rotational transformation as shown in Eq. (37).

3.3 Quadrilateral Finite Element Based on Physical Coordinates

Batoz developed the DKQ (discrete Kirchhoff quadrilateral) element by extending the DKT concept. However, the results of the convergence rates during displacement analysis were not good compared to McNeal or LORA by Robinson and Haggenmacher[10]. In this paper, quadrilateral FE were developed by the same physical coordinate based methodology as the triangular FE mentioned in Sec. 3.1. The FE formulation process was the same as the triangular FE in Sec. 3.1, but the analytical area integration $\iint S(x,y)dxdy$ was very lengthy and was obtained using software.

3.4 Isoparametric Quadrilateral Finite Element (Mindlin-Reissner Thick Plate Theory) Based on Natural Coordinates

Mindlin-Reissner, Naghdi plate theory[6,7,11~13] used a generalization of the Kirchhoff hypothesis: points of the plate which are on the normal to the undeformed middle surface remain on a straight line even after deformations are proceeded but, which are not necessarily normal to the deformed middle surface[11]. That means $\gamma_{xz} \neq 0$, $\gamma_{yz} \neq 0$, $\sigma_{zz} = 0$ and $\varepsilon_{zz} = 0$. Therefore, shear deformation term was added to traditional Kirchhoff plate theory, Eq. (23) as a second term.

$$U = \frac{1}{2}\int_v \sigma^t\{\varepsilon\}dv + \frac{\alpha}{2}\int_v \{\sigma_c\}^t\{\varepsilon_c\}dv \tag{44}$$

$$\{\sigma_c\}^t = [\tau_{xz} \ \ \tau_{yz}] \tag{45}$$

$$\{\varepsilon_c\}^t = [\gamma_{xz} \ \ \gamma_{yz}] \tag{46}$$

Where, α is shear correction factor[6], 5/6, $\{\sigma_c\}$ and $\{\varepsilon_c\}$ are stress and strain due to shear deformations. Transverse shear strain-displacement relationship[6]:

$$\gamma_{xz} = \frac{\partial w}{\partial x} + \frac{\partial u}{\partial z} = \frac{\partial w}{\partial x} + \theta_x \tag{47}$$

$$\gamma_{yz} = \frac{\partial w}{\partial y} + \frac{\partial v}{\partial z} = \frac{\partial w}{\partial y} + \theta_y \tag{48}$$

$$\sigma = D\varepsilon \tag{49}$$

$$\sigma_c = D_c\varepsilon_c \tag{50}$$

Where, $D = \frac{E}{1-\upsilon^2}\begin{bmatrix} 1 & \upsilon & 0 \\ \upsilon & 1 & 0 \\ 0 & 0 & \frac{1-\upsilon}{2} \end{bmatrix}$, $D_c = \begin{bmatrix} G & 0 \\ 0 & G \end{bmatrix}$

With the shape function,

$$\begin{aligned} w &= \sum_{i=1}^{4} N_i(\zeta,\eta)w_i, \\ \theta_x &= \sum_{i=1}^{4} N_i(\zeta,\eta)\theta_{xi}, \\ \theta_y &= \sum_{i=1}^{4} N_i(\zeta,\eta)\theta_{yi} \end{aligned} \tag{51}$$

$$\{\varepsilon\} = -z[B]\{w\}_e \tag{52}$$

$$\{\varepsilon_c\} = [B_c]\{w\}_e \tag{53}$$

$$[B] = \begin{bmatrix} 0 & \frac{\partial N_1}{\partial x} & 0 & 0 & \frac{\partial N_2}{\partial x} & 0 & 0 & \frac{\partial N_3}{\partial x} & 0 & 0 & \frac{\partial N_4}{\partial x} & 0 \\ 0 & 0 & \frac{\partial N_1}{\partial y} & 0 & 0 & \frac{\partial N_2}{\partial y} & 0 & 0 & \frac{\partial N_3}{\partial y} & 0 & 0 & \frac{\partial N_4}{\partial y} \\ 0 & \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} & 0 & \frac{\partial N_4}{\partial y} & \frac{\partial N_4}{\partial x} \end{bmatrix} \tag{54}$$

$$[B_c] = \begin{bmatrix} \frac{\partial N_1}{\partial x} & N_1 & 0 & \frac{\partial N_2}{\partial x} & N_2 & 0 & \frac{\partial N_3}{\partial x} & N_3 & 0 & \frac{\partial N_4}{\partial x} & N_4 & 0 \\ \frac{\partial N_1}{\partial y} & 0 & N_1 & \frac{\partial N_2}{\partial y} & 0 & N_2 & \frac{\partial N_3}{\partial y} & 0 & N_3 & \frac{\partial N_4}{\partial y} & 0 & N_4 \end{bmatrix} \tag{55}$$

$$\{w\}_e^t = [w_i \ \theta_{xi} \ \theta_{yi} \ w_j \ \theta_{xj} \ \theta_{yj} \ w_k \ \theta_{xk} \ \theta_{yk} \ w_l \ \theta_{xl} \ \theta_{yl}] \tag{56}$$

$$\begin{aligned} U &= \frac{1}{2}\{w\}_e^t \int_A [B]^t \frac{t^3}{12} D[B] dA \{w\}_e \\ &\quad + \frac{1}{2}\{w\}_e^t \int_A \alpha t [B_c]^t D_c [B_c] dA \{w\}_e \\ &= \frac{1}{2}\{w\}_e^t [k]_e \{w\}_e \end{aligned} \tag{57}$$

$$\begin{aligned} [k]_e &= \frac{t^3}{12} \int_A [B]^t \frac{t^3}{12} D[B] dA \\ &\quad + \int_A \alpha t [B_c]^t D_c [B_c] dA \end{aligned} \tag{58}$$

$$\{f\}_e = \int_{-1}^{1}\int_{-1}^{1} [N]\{P\} |J| d\xi d\eta \tag{59}$$

$$[k]_e \{w\}_e = \{f\}_e \tag{60}$$

3.5 Plate Shell Finite Element Based on Natural Coordinates

Kwon proposed MATLAB® FE codes using plate shell finite element, 5 DOFs per node, which combined membrane element and plate bending element using natural coordinates based isoparametric elements[7]. Since the inplane rotational term was not included in plate shell element, it left null or zero values in the stiffness matrix. So-called drilling DOF incurred singularity in structural stiffness matrix if all the elements were coplanar[7]. To solve this problem, Knight[14] suggested that a very small value be specified for the stiffness of the drilling DOF so that the contribution to the strain energy equation from this term would be zero[7]. The local stiffness matrix in the natural local coordinates was transformed into the global coordinates and assembled to obtain the total global stiffness matrix.

4. Validation

The FE codes suggested in Section 3 were validated using a square plate with four sides fixed. The plate specifications and the analysis conditions used in the analysis were as followed. Young's modulus $E = 2 \times 10^{11}$ Pa, Poisson ratio $\upsilon = 0.285$, density $\rho =$ 7874 kg/m^3 and width × length, 1 m × 1 m with thickness 0.1 m. A square plate with a uniform distribution load of 50 000 Pa was modeled on a 10 × 10, 20 × 20, 40 × 40 grids on a flat plate. Theoretical calculations, results calculated from the Semie model[5], ANSYS® shell 63, ANSYS® shell 163 and LISA® software are shown in Table 1. The results using the five FE codes suggested in Section 3 are also presented. The displacement at the center of the square plate calculated by the theoretical formula (Eq. (21)) was –3.47 μm. Semie model using triangular FE showed a displacement of –4.81 μm when modeling a square plate at 100 × 100. It was 38.6 % error compared with the value predicted by the theoretical Eq. (21), and it was also reported that the error was –42.4 % and 384.15 %, respectively, when using ANSYS® shell 63 and ANSYS® shell 163 element (5). However, in Sec. 3.1, the result using the triangle FE codes formulated by the methodology proposed by Semie was –3.50 μm, which was within 0.9 % of the theoretical value. The FE codes presented in Sec. 3.1 ~ Sec. 3.3 which were formulated according to the Kirchhoff thin plate theory showed a maximum error of 0.9 % with the theoretically predicted value. Therefore, there would be some errors in the FE codes programmed by Semie, and it was assured that the analys using ANSYS® software was also applied incorrectly. Consequently, it showed that the

Table 1 Comparison of displacement at center of square plate under distributed load 50 000 Pa with fixed boundary condition by several methods (Unit: μm)

	Elem × elem	10 × 10	20 × 20	40 × 40
1	Theory, Eq. (21)	–3.47	–3.47	–3.47
2	Semi[5]	-	-	–4.81
3	ANSYS 63[5]	-	-	–2.00
4	ANSYS 163[5]	-	-	–16.80
5	LISA	–4.01	–4.02	–4.03
6	Triangular FE	–3.65	–3.54	–3.50
7	Rectangular FE	–3.56	–3.51	–3.49
8	Quadri. FE	–3.56	–3.51	–3.49
9	Mindlin-Reissner	–3.74	–3.76	–3.77
10	Plate shell	–4.12	–4.13	–4.13

FEM codes formulated according to the Kirchhoff thin plate theory were suitable for use. The analysis result by the commercial software LISA® version 8.1.0 beta version was 16.14 %. In case of Mindlin-Reissner thick plate theory considering shear deformation, it showed 8.6 % error with Kirchhoff thin plate theory. On the other hand, 3 dimensional plate shell FE codes showed 19.02 % error with Kirchhoff thin plate theory.

5. Comparison of Computation Time for System Matrix According to Solving Methods

It can be classified by two regimes to obtain the solution of the system equation, $Ax = b$. One is full matrix solving methods with $A(i, j)$ form of square coefficient matrix A and the other sparse matrix solving methods with coefficient matrix A as sparse form.

5.1 Full Matrix Solving Methods

The FE codes developed in Section 3 initially were coded in OCTAVE® language because of necessity of graphic postprocessing in the coding stage. OCTAVE® intrinsic operator '\' was adopted to obtain the vector x in the system equation $Ax = b$. But, it needed a long computation time, and codes were translated to FORTRAN language in order to improve its computing performance. In order to find the solution vector in the system equation $Ax = b$, which was the form $[K]\{W\} = \{F\}$ used in the structural displacement FE codes in Section 3, and compare computing performance, the following eight full matrix solving methods including four methods described in reference[15] were examined.

(1) Gauss elimination method with partial pivoting algorithm, direct solving method.

(2) Successive over-relaxation (SOR($\omega = 1.95$)) algorithm, iterative solving method.

(3) Conjugate gradient (CG) Algorithm, iterative solving methods, tolerance = 10^{-11}.

(4) Preconditioned conjugate gradient algorithm with diagonal preconditioner (PCG-D), iterative solving method, tolerance = 10^{-11}.

(5) Direct Gauss elimination method with partial pivoting (DGESV), FORTRAN MKL® library.

(6) Iterative Gauss elimination method with partial pivoting (DSGESV), FORTRAN MKL® library.

(7) OCTAVE® '\', full matrix solver (OCT-F).

(8) MATLAB® '\', full matrix solver (MAT-F).

Methods (1) to (4) showed that the same algorithmic codes were simply translated in OCTAVE®, MATLAB®, and FORTRAN languages, respectively, rather than using specialized built-in functions in each language. There were two classes of iteration methods[16]. One was stationary iteration methods or fixed-point iteration methods like as Jacobi, Gauss-Seidel, SOR and the other was projection methods, which was called Krylov subspace method, like as CG, PCG, generalized minimum residual method (GMRES). Projection methods were required that approximation solution x extracted from the subspace K should make residual $r = b - Ax$ orthogonal to the subspace L. In the first stage, the system coefficient matrix A used in the simulation was a symmetric positive definite (SPD) matrix, a tridiagonal matrix with a bandwidth of $1 + 2 \times nx$, with one diagonal with four off-diagonal terms added[17]. Nx was the distance between diagonal term and element locations and n was the size of matrix A.

$$nx = \sqrt{n}$$

$$A(i,i) = 4 \times (1+nx)^2$$

$$\begin{aligned} A(i,i+1) &= A(i+1,i) = A(i,i+nx) \\ &= A(i+nx,i) = -(1+nx)^2, \ i = 1,2,...,n \end{aligned}$$

The vector b is obtained by assuming $x(1{:}n) = 1$ in $Ax = b$. The personal computer (PC) used was Intel® Core i7 4790K@4.0 GHz processor, 32 GB RAM, and operating on Windows 7 64 bit. FORTRAN codes were compiled using Intel® Parallel Studio XE 2015 MKL

compiler 64 bit. In the case of 2000 DOF, the calculation time ratio was 27 883 sec : 144.24 sec : 32.26 sec = 864 : 4 : 1 when the Gauss elimination method (solid line) was excuted with OCTAVE®, MATLAB®, and FORTRAN language, respectively (Fig. 5). In the case of the PCG-D algorithm, it showed OCTAVE® : MATLAB® : FORTRAN = 21 436 sec : 70.31 sec : 1.12 sec = 19 139 : 63 : 1. When the Gauss method and the SOR method were used, the calculation time ratio of OCTAVE® / MATLAB® was 193 and 197, CG method and PCG-D was 355 and 305, respectively. That is, it showed that the computing speed was 193 to 355 times faster to operate in MATLAB® than in OCTAVE® for same algorithmic codes. In addition, executing similar codes in FORTRAN was 4 to 63 times faster than running in MATLAB®, and 831 to 2269 times faster than in OCTAVE® between 200 DOF and 10 000 DOF. The PCG-D method in FORTRAN showed the fastest calculation time.

Figure 6 showed the computing time when the operator '\' built in OCTAVE® and MATLAB®, the FORTRAN MKL library direct solver DGESV and iterative solver DSGESV were used. The shortest computation time for 2000 DOF was 0.051 sec for the direct method DGESV and 21 436 sec (≒ 10 hours) for the slowest case, OCTAVE®-PCG-D (marker: blue ●). It showed OCTAVE®-PCG-D : MATLAB®-PCG-D : FORTRAN-PCG-D : OCTAVE '\' : MATLAB '\' : DGESV = 21 436 : 70.31 : 1.12 : 0.637 : 0.065 : 0.051 = 420 314 : 1379 : 22 : 12.5 : 1.27 : 1. In the case of MKL FORTRAN library, iteration method was faster than direct method at 5000 DOF or more. In view of above discussions, it showed that the program language and solving method to be used were very important in computing time of system equation. The intrinsic operator '\' used in MATLAB® was shown to be an effective method. Using the MATLAB® '\' (-◆-) showed 1080 times faster than the computing time of the PCG-D programmed in MATLAB® language(◆) in case of 2000 DOF. For 10 000 DOF system equation, FORTRAN-PCG-D was 69.3 sec (Fig. 5), DGESV 4.5 sec, OCTAVE® '\' 104.4 sec, and MATLAB® '\' 3.3 sec (Fig. 6), respectively.

So far we have compared the computation times for the cases where the coefficient matrix A was ideally diagonally banded SPD matrix. Gauss and SOR method, which was stationary iteration method with slow computation speed, were excluded from comparison. Using the data obtained from the FEM co-

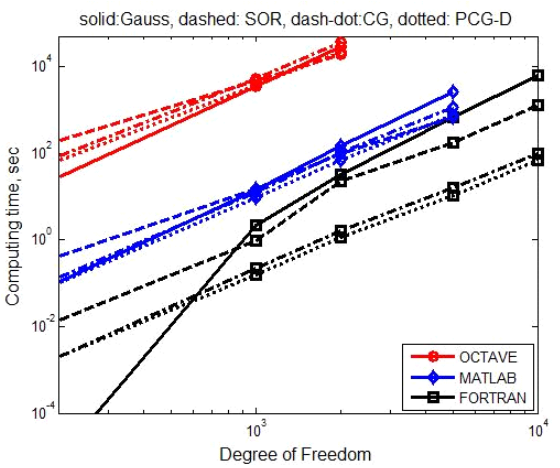

Fig. 5 Comparison of computing time using full matrix solvers for OCTAVE®, MATLAB® and FORTRAN codes for 5 model cases, marker: ○: OCTAVE®, ◇: MATLAB®, □: FORTRAN, linestyle: -: Gauss, --: SOR, -.: CG, ▫▫: PCG-D, range: 200 DOF ~ 10 000 DOF (Unit: sec)

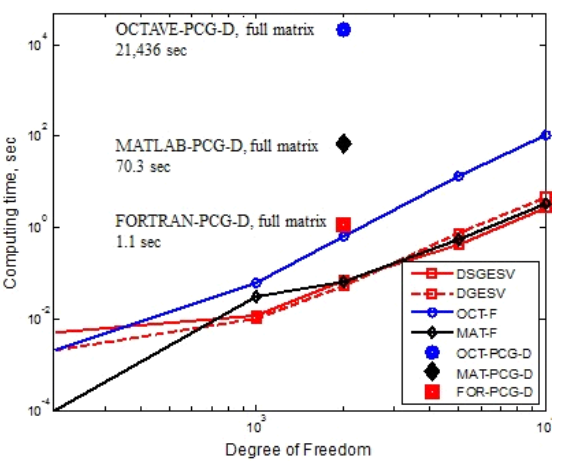

Fig. 6 Comparison of computing time using full matrix solvers, marker-line: ●: OCTAVE®-PCG-D, ○-: OCTAVE® '\', ◆: MATLAB®-PCG-D, ◇-: MATLAB® '\', ■: FORTRAN-PCG-D, □-: DSGESV, □--: DGESV, range: 200 DOF ~ 10 000 DOF (Unit: sec)

des in Section 3.2, the computational speeds for the system equations were compared. In the formal FEM, the stiffness matrix $[K]$ typically exhibited the characteristics of SPD matrix[18]. The stiffness matrix and the force vector were obtained using FEM codes using the rectangular FE codes in Sec. 3.2. The square plate were modeled to 20×20, 40×40, 80×80, 100×100, 140×140 FE meshes, respectively, which was called Ex1, Ex2, Ex3, Ex4 and Ex5. Since the DOF per node was 3, each model size was 1323, 5043, 19 683, 30 603, 59 643 DOF, respectively. The specifications, boundary conditions and force conditions of the square plate were described in Section 4. The stiffness matrices and force vectors were obtained by applying fixed boundary conditions to the four sides of the square plate. The stiffness matrices and force vectors for the five data sizes were output to data files and then used in each solving method programs. Figure 7 and Fig. 8 showed the stiffness matrices before and after applying the boundary condition to the system equation of the case of square plate (4×4) (75 DOF). The modified stiffness matrix pattern (Fig. 8) was a general matrix form and obtained by applying fixed boundary conditions to the stiffness matrix pattern in Fig. 7.

Table 2 showed the calculation times obtained using the following seven solving methods for the five matrix size models. Methods (1) ~ (4) were compiled with Intel® Parallel Studio XE 2015 FORTRAN compiler. Figure 9 showed the computation time results for each solver.

(1) CG codes[15], iterative method.

(2) PCG-D codes[15], diagonal preconditioner, iterative method.

(3) DSGESV, iterative Gauss elimination method with partial pivoting.

(4) DGESV, direct Gauss elimination method with partial pivoting.

(5) OCTAVE® operator '\', OCT-F.

(6) MATLAB® operator '\', MAT-F.

(7) LISA® software, version 8.1.0 beta version.

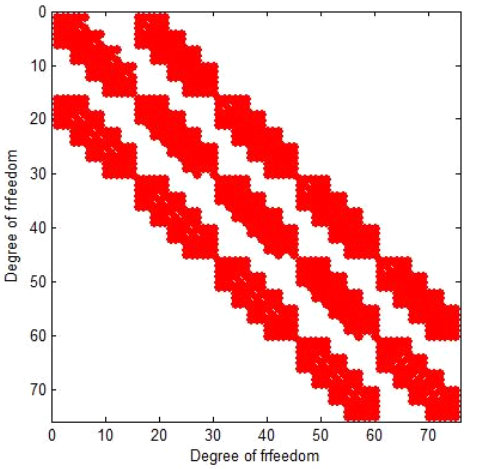

Fig. 7 Pattern of stiffness matrix K of 75×75 before applied boundary condition, white part: '0' value

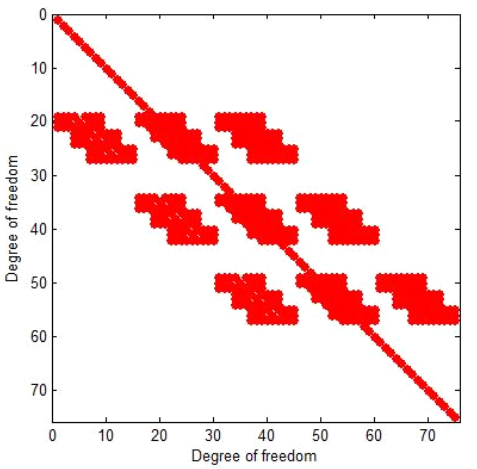

Fig. 8 Pattern of stiffness matrix K of 75×75 after applied boundary condition, white part: '0' value

Table 2 Comparison of computing time using full matrix solvers for 5 models (Unit: sec)

DOF	Ex1	Ex2	Ex3	Ex4	Ex5
	1323	5043	19 683	30 603	59 643
(1) CG	1.5	133.9	5657.6	13 543.5	
(2) PCG-D	0.32	12.06	836.8	3078.7	
(3) DSGESV	0.11	1.65	55.8	202.0	31 235.7
(4) DGESV	0.02	0.75	31.5	120.8	1218.4
(5) OCTAVE	0.27	14.84	2580.9		
(6) MATLAB	0.06	1.12	47.1	168.2	3442.8
(7) LISA	-	1.0	13.0	26.0	106.0

*Blank spaces mean omitted because of expecting excessive computing times.
*LISA®, 4 node quadrilateral element

In the direct solution method DGESV (□ dashed line), it showed that the calculation time was nearly proportional to DOF^3. In MATLAB® operator '\', it was 0.065 sec at 1323 DOF and 3443 sec at 59 643 DOF, the computation time was also proportional to DOF^3. In Fig. 6, iterative method DSGESV showed faster calculation speed than direct method DGESV, but in Fig. 9, iterative method showed slower calculation speed. This was because the coefficient matrix A applied in Fig. 6 was a sparse SPD matrix and the matrix A used in Fig. 9 was a non-SPD matrix and a relatively dense matrix. In OCTAVE® and MATLAB®, the same operator '\' was used. As the DOF increases, the calculation time difference between MATLAB® and OCTAVE® becomes larger. When calculating the FE model of 19 683 DOF (Ex3), the computation time ratio was 55 times as 47 seconds in MATLAB® and 2581 seconds in OCTAVE®. In the case of 59 643 DOF (Ex5), it was 3443 sec for MATLAB® '\' and 1218 sec for DGESV routine. There was a rapid increment in computing time at 59 643 DOF for DSGESV. The reason why the computing time using DSGESV, an iterative method, surged compared with the direct method DGESV, was that it was caused by a reduction in efficiency caused by using virtual memory due to lack of CPU memory. Unlike DGESV, DSGESV, an iterative method, required additional workspace memory equal in size to system matrix A. To calculate an Ex5 model using DSGESV, we needed at least 56 GB of memory, but used PC had hardware specifications with 32 GB of RAM, so it needed virtual memory and so, it utilized hard disk as a virtual memory source, which resulted in a slower overall memory accessing time than using only CPU memory in direct solver DGESV. That is, by using hard disk memory rather than CPU memory as swap memory, the DOF was increased 1.9 times in Ex5 compared to Ex4, but the calculation time was 31 236 sec which was 155 times larger than 202 sec in Ex4. Therefore, in the case of Ex5, 59 643 DOF, DSGESV was needed 8 hours 41 minutes (31 236 sec), MATLAB® solver '\' 57 minutes (3443 sec) and DGESV took 20 min (1218 sec). In all cases, the direct solver, the DGESV routine showed the fastest calculation speed. For the commercial software LISA®, the solving time was 106 sec. A comparative analysis of Table 2, Fig. 5 and Fig. 6 revealed interesting facts. It was summarized the calculation time for 5000 DOF in Table 3.

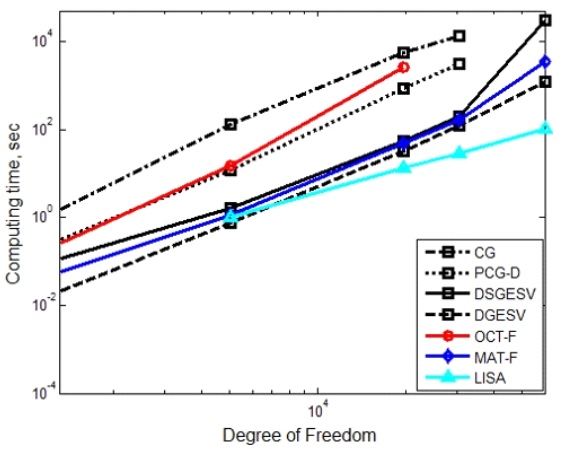

Fig. 9 Comparison of computing time using full matrix solvers for 7 methods, marker-line: □-: DSGESV, □--: DGESV, □-.: CG, □▫▫: PCG-D, ○-: OCTAVE® '\', ◆- : MATLAB® '\', ▲-: LISA®, range: 1323 DOF ~ 59 643 DOF (Unit: sec)

Direct solver DGESV showed almost the same calculation time in both cases. The iterative solvers, CG and DSGESV, showed 4-8 times faster computing time with the SPD coefficient matrix in system equation. This was because the DOF was almost the same, but the SPD matrix was a sparser matrix than Ex2. Even when the MATLAB® operator '\' was used, the calculation time was half shorter in the sparse matrix SPD. As an interesting result, the direct method DGESV in Ex2 showed faster calcu-

Table 3 Comparison of computing time using full matrix solvers for general matrix (Ex2) and sparse SPD matrix (Unit: sec)

DOF	Ex2	SPD	Ratio, Ex2/SPD
	5043	5000	
CG	133.85	15.90	8.42
PCG-D	12.06	10.62	1.14
DSGESV	1.65	0.41	3.99
DGESV	0.75	0.73	1.02
MATLAB® '\'	1.12	0.54	2.07

lation speed than the iterative method DSGESV, but the iterative method showed faster calculation time than the direct method in sparse matrix SPD. That is, when the number of non-zero coefficients was small, the iteration method showed faster calculation speed than the direct method.

5.2 Sparse Matrix Solving Method

In Section 5.1, full-matrix solving methods were used to examine the computing times in system equation for the SPD coefficient matrix (Figs. 5 ~ 6, 200 ~ 10 000 DOF) and for the stiffness matrix obtained in the structural displacement FEM (Table 2, 1323 ~ 59 643 DOF). Practically, there were limitations on the size of matrix that could be solved due to full matrix memory space. Also, since the matrix operation was performed on the '0' values in the coefficient matrix A (Figs. 7 ~ 8), lots of calculation time was required for full matrix solvers. If there were many '0' values in coefficient matrix A, using a sparse solver would be benefit from memory savings and computation speed. In order to use the sparse matrix solving methods, the system matrix A must be converted from full matrix form to a sparse matrix one. In OCTAVE® and MATLAB®, the sparse matrix solver used the '\' operator, which was the same form as the full matrix solver. However, the system matrix A must be transformed into a sparse matrix form. In this paper, it was adopted in order to make the system coefficient matrix A a sparse matrix form, one-based indexing CSR (compressed sparse row) format[19~21] in FORTRAN codes and CSC (compressed sparse column) format in MATLAB® and OCTAVE® codes, respectively. NNZ (number of non-zero) was defined the number of non-zero coefficients in the coefficient matrix A whose size was $n \times n$ is. Sparsity (%) is defined as sparsity (%) = NNZ / ($n \times n$) × 100. Table 4 and Table 5 showed the DOF, NNZ, and sparsity(%) of the data to be used in this section and the amount of memory required to store matrix A in full matrix and sparse matrix. Table 4 showed that the 50 000 DOF (#7) model required 20 GB of memory to process a full matrix of coefficients A, but a sparse matrix required only 2 MB, which was 0.01 % of the original memory requirement. 300 000 DOF (#10) required 720 GB of memory for the full matrix, but it could not be handled by general PCs. However, if sparse matrix form is used, only 12 MB of 0.002 % of raw data is needed. Table 6 compared computing times of full matrix solvers and sparse matrix solvers for from #1 to #5 data models.

When using the full matrix solver, operator '\' in models with a range of 200 DOF ~ 10 000 DOF, it showed computing time ratio of OCTAVE® / MATLAB® was 2 ~ 32. However, using sparse solver showed

Table 4 Comparison of memory sizes of full and sparse for SPD matrix, NNZ, sparsity (%)

	DOF	Full	NNZ	Sparse	Spar. (%)
#1	200	.32 MB	970	0.01 MB	2.43
#2	1000	8 MB	4934	0.04 MB	0.49
#3	2000	32 MB	9908	0.08 MB	0.25
#4	5000	200 MB	24 856	0.20 MB	0.10
#5	10 000	800 MB	49 798	0.40 MB	0.05
#6	30 000	7.2 GB	149 652	1.20 MB	0.02
#7	50 000	20 GB	249 550	2.00 MB	0.01
#8	0.1 M	80 GB	499 366	3.99 MB	5×10^{-3}
#9	0.2 M	320 GB	999 104	7.99 MB	3×10^{-3}
#10	0.3 M	720 GB	1 498 902	11.99 MB	1×10^{-3}
#11	0.5 M	2 TB	2 498 584	19.99 MB	5×10^{-4}
#12	1 M	8 TB	4 997 998	39.98 MB	5×10^{-4}
#13	2 M	32 TB	9 997 170	79.98 MB	3×10^{-4}
#14	5 M	200 TB	24 995 526	0.2 GB	1×10^{-4}
#15	10 M	800 TB	49 993 674	0.4 GB	5×10^{-5}
#16	20 M	3.2 PB	99 991 054	0.8 GB	3×10^{-5}

Table 5 Comparison of memory sizes of full and sparse matrix for FEM stiffness matrix K, NNZ, sparsity (%)

	DOF	Full	NNZ	Sparse	Spar. (%)
Ex1	1323	14 MB	29 453	0.24 MB	1.68
Ex2	5043	203 MB	123 381	0.99 MB	0.49
Ex3	19 683	3.10 GB	505 325	4.04 MB	0.13
Ex4	30 603	7.49 GB	790 713	6.33 MB	0.08
Ex5	59 643	28.46 GB	1 564 440	12.52 MB	0.04

OCTAVE® / MATLAB® ≈ 1. The method using sparse solver '\' showed the same performance of MATLAB® and OCTAVE®. In the 5000 DOF model (#4), the sparse solver (0.006 sec) of MATLAB® and OCTAVE® was 122 times faster than the full matrix direct solver DGESV (0.732 sec). Sparse PCG-D (0.016 sec) showed 664 times faster than full matrix PCG-D (10.62 sec). Sparse solvers were used to calculate system equation with the SPD coefficient matrix from 200 to 20 000 000 DOF using eight sparse matrix methods.

(1) PARDISO[19,22~24]: PARallel sparse Direct and multi-recursive Iterative linear SOlvers

(2) OCTAVE® '\', OCT-S.

(3) MATLAB® '\', MAT-S.

(4) CG[15,16,21,24]

(5) PCG-D[15,16,21,24].

(6) PCG-split-D[15,16,21,24], PCG with diagonal split preconditioner.

(7) PCG-IC(0)[15,16,21,24,25]: PCG with incomplete Cholesky decomposition with no fillin preconditioner

(8) PCG-ILU(0)[15,16,21,24]: PCG with incomplete LU decomposition with no fillin preconditioner

Table 6 Comparison of computing time of $Ax = b$ using full matrix solvers and sparse matrix solvers, (Unit: sec)

	DOF		OCT.	MAT.	PCG-D	DSG.	DGE.
#1	200	F	2.0×10^{-3}	0.	2.0×10^{-3}	5.0×10^{-3}	0.002
		S	1.0×10^{-3}	1.0×10^{-3}	0.		
#2	1000	F	6.3×10^{-2}	3.0×10^{-2}	1.6×10^{-1}	1.2×10^{-2}	0.01
		S	1.0×10^{-3}	1.0×10^{-3}	0.		
#3	2000	F	6.4×10^{-3}	6.5×10^{-2}	1.1	6.9×10^{-2}	0.05
		S	2.0×10^{-3}	2.0×10^{-3}	1.6×10^{-2}		
#4	5000	F	1.3×10^{1}	5.4×10^{-1}	10.6	4.1×10^{-1}	0.73
		S	6.0×10^{-3}	6.0×10^{-3}	1.6×10^{-2}		
#5	10 000	F	1.0×10^{2}	3.3	63.4	2.5	4.52
		S	1.4×10^{-2}	1.1×10^{-2}	4.7×10^{-2}		

*F and S in column 3 mean full matrix solver and sparse matrix solver, each.
*PCG-D: PCG with diagonal preconditioner, full matrix codes are from Ref. (26) and sparse matrix codes are modified from Ref. (25).
*DSGESV and DGESV are full matrix solver.

In the case of the 2000 DOF mentioned in Section 5.1, the PCG-D (marker: black ●) implemented with OCTAVE® language was 21 436 sec, and the OCTAVE® full matrix solver '\' (marker: red ●), 0.637 sec and OCTAVE® sparse solver '\' (-●-), 0.001 8 sec, respectively (Fig. 10). The calculation time ratio was 12 000 000 : 425 : 1. OCTAVE® sparse solver '\' showed twelve million times faster calculation time than PCG-D translated by OCTAVE® language. The computing time using the sparse solver PCG-D compiled with FORTRAN was 3.3×10^{-3} sec, which was 1.8 times slower than the OCTAVE® sparse solver '\'. PCG-D adopted the diagonal terms of the coefficient matrix A as the preconditioner M and PCG-split-D adopted the preconditioner as the (diagonal terms)$^{1/2}$ of the coefficient matrix A, respectively. Figure 10 and Fig. 11 showed that PCG-split-D was more efficient than PCG-D because the iteration number in PCG-split-D is 40 % smaller than in PCG-D above 500 000 DOF model. PCG-IC(0) showed similar calculation times as PCG-D and PCG-split-D, even though the number of iterations was 1/2 to 1/3 times that of PCG-D or

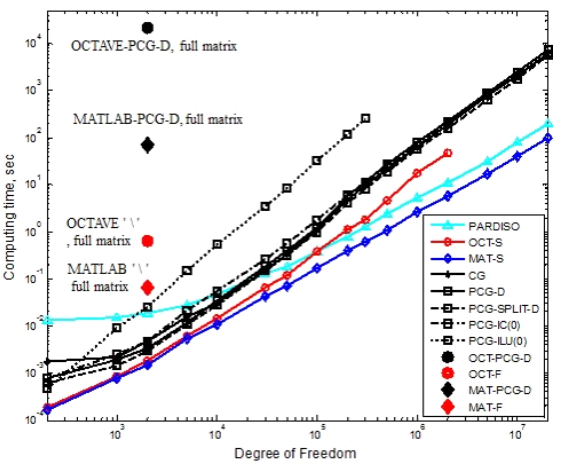

Fig. 10 Comparison of computing time according to sparse solvers for SPD matrix, marker-line: Δ-: PARDISO, ○-: OCT-S '\', ◇-: MAT-S '\', +-: CG, □-: PCG-D, □--: PCG-split-D., □-.: PCG-IC(0), □▫▫: PCG-ILU(0), ●: OCTAVE®-PCG-D, full matrix solver, ●: OCTAVE® '\' -full matrix solver, ◆: MATLAB®-PCG-D, full matrix solver, ◆: MATLAB® '\' -full matrix solver, range: 200 DOF ~ 20 000 000 DOF (Unit: sec)

PCG-split-D (Fig. 11). Thus, it meant that the computation amount per iteration of PCG-D or PCG-split-D was smaller than PCG-IC(0). In the SPD coefficient matrix data, iteration number in PCG-D showed 140 % greater than the number of iteration in PCG-split-D and needed 140 % of the calculation time. In the case of 2 000 000 DOF (#13), the calculation time in OCTAVE® '\' was 47.8 sec, but the time to read the data file took 6301 sec. It took 114 sec for MATLAB® and 5.2 sec for FORTRAN PCG to read the data file #13. Saad showed that when the coefficient matrix A was the SPD matrix and the preconditioner M was incomplete Cholesky product, $A \approx M = LL^t$, it was equal to the number of iterations when using M as the preconditioner or when splitting it into L and L^t for PCG method[21].

On the other hand, when the coefficient matrix A was a general matrix, a preconditioner was obtained with incomplete LU factorization instead of incomplete Cholesky decomposition. Split GMRES using L and U as preconditioners in GMRES instead of PCG was most effective when the coefficient matrix A was nearly symmetric and was not effective from using preconditioner $M = LU$ in other cases[21]. In this paper, GMRES was applied to FEM data, which showed that it was inferior to PCG-D or PCG-split-D in terms of calculation speed. Especially, when using

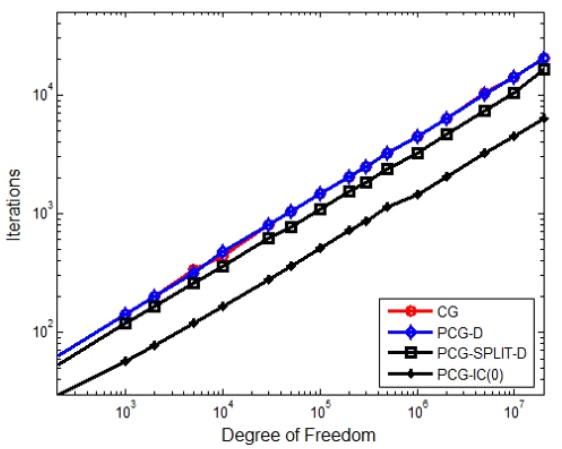

Fig. 11 Comparison of iterations according to sparse solvers for SPD matrix, marker-line: ○-: CG , ◇-: PCG-D, □-: PCG-split-D, +-: PCG-IC(0), range: 200 DOF ~ 20 000 000 DOF, (Unit: number)

FEM data, it showed a tendency to diverge above 1323 DOF (Ex1 model). FGMRES (RCI Flexible Generalized Minimal RESidual method with ILUT Preconditioner)[26] and MGMRES(restarted GMRES) suggested by Burkardt[27] were less likely to diverge than GMRES. However, according to the characteristics of the coefficient matrix A, the user had to determine the inner loop and the outer loop number in advance, and the divergence or convergence was determined according to these values. Also, if the number of inner loop was large, the solution converged rapidly but the memory to be memorized increased and the memory efficiency decreased. Here, interestingly, Saad[21] described the difference in computational complexity in PCG when choosing preconditioner using incomplete Cholesky decomposition for SPD matrix. In this paper, we showed that the PCG could also be applied to non-SPD matrices and the same description applied to the SPD matrix when we adopted diagonal preconditioner or split diagonal preconditioner and it could be especially emphasized to non-SPD coefficient matrix. The PCG method with incomplete Cholesky decomposition preconditioner was known to apply only to SPD matrix[28]. If the coefficient matrix was non-SPD matrix, Cholesky decomposition could not be applied; instead, LU decomposition applicable to a general matrix could be applied. The PCG-IC(0) could be applied only when the coefficient matrix was SPD, so it could not be applied to the FEM system matrix equation applied the actual boundary condition as in Fig. 8.

In Table 5 (1323 ~ 59 643 DOF), the PCD-IC(0) could not be applied because the matrix was non-SPD. In order to solve the general matrix in the PCG method, the preconditioner could be obtained by incomplete LU decomposition instead of incomplete Cholesky decomposition. We could use PCG-ILU(0), which used L and U obtained by incomplete LU decomposition with no fillin as the preconditioner, to perform the same size LU decomposition using the coefficient matrix A. The preconditioner ILU(0) was efficient in terms of memory because the size and position of the matrix were the same as the coeffi-

cient matrix A, but there was a restriction on its use because it tended to diverge. In case of non-SPD coefficient matrix as in Fig. 12, PCG-ILU(0) showed divergence when the size of the matrix was larger than 1323 DOF. As an alternative, we could use PCG-ILU(τ) by obtaining L and U with Crout incomplete LU decomposition. However, the size of L and U obtained by Crout incomplete LU decomposition was large, which was not useful in terms of memory. The Crout method could be used to solve the ILU when using the Crout ILU method. However, when the Crout method was used to obtain the preconditioner LU, it showed NNZ (ILU(τ)) >> NNZ(A) (Table 7). Therefore, it could not be applied to a large size non-SPD coefficient matrix. On the other hand, as shown in the appendix, PCG-D or PCG-split-D could perform very fast convergences if coefficient matrix A was not asymmetric. In Fig. 12, the full matrix direct solver DGESV (marker: red ▼) was 1218 sec and the MATLAB® '\' -full matrix solver (marker: blue ◆) was 3443 sec in the case of 59 643 DOF.

The MATLAB® sparse matrix solver '\' (marker: blue -◇-) was 0.592 sec. Therefore, when using the MATLAB® operator '\', computing time ratio of the full matrix solver / sparse matrix solver = 5816 times. That is, the sparse MATLAB® solver was about six thousand times faster than the full matrix MATLAB® solver.

Ratio of number of iterations in the sparse PCG method was PCD-D / PCD-split-D = 1.4 in the SPD matrix (Fig. 11). When the coefficient of the matrix was non-SPD (Fig. 13), it showed ratio of number of iterations of PCD-D / PCD-split-D = 4 and the convergence speed of PCG-split-D was 4 times faster than PCG-D. That is, in the case of the asymmetric matrix, PCG-split-D showed shorter computation time than PCG-D. The PCG-split-D, 4.805 sec, was 4.7 times faster than the PCG-D, 22.792 sec, in the case of the 59 643 DOF, where the coefficient matrix was

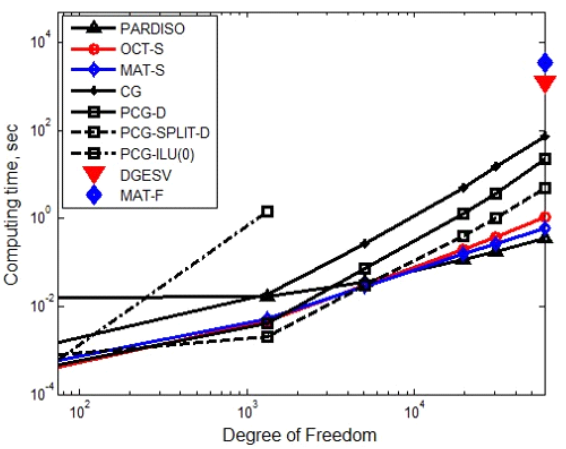

Fig. 12 Comparison of computing time using sparse solvers for non-SPD matrix, marker-line: Δ-: PARDISO, ○-: OCTAVE® '\', ◇-: MATLAB® '\', +-: CG, □-: PCG-D, □--: PCG-split-D., □▫▫: PCG-ILU(0), ◆: MATLAB® '\' - full matrix solver, ▼: DGESV-full matrix solver, range: 75 DOF ~ 59 643 DOF (Unit: sec)

Table 7 Comparison of number of non-zero of coefficient matrix A, incomplete LU with no fillin (ILU(0)) and incomplete LU with Crout ILU(τ), τ: drop tolerance, τ=10^{-9}

	DOF	NNZ (A)	NNZ (ILU(0))	NNZ (ILU(τ))
Ex1	1323	29 453	29 453	132 910
Ex2	5043	123 381	123 381	922 478
Ex3	19 683	505 325	505 325	6 100 455
Ex4	30 603	790 713	790 713	11 030 223
Ex5	59 643	1 564 440	1 564 440	25 895 749

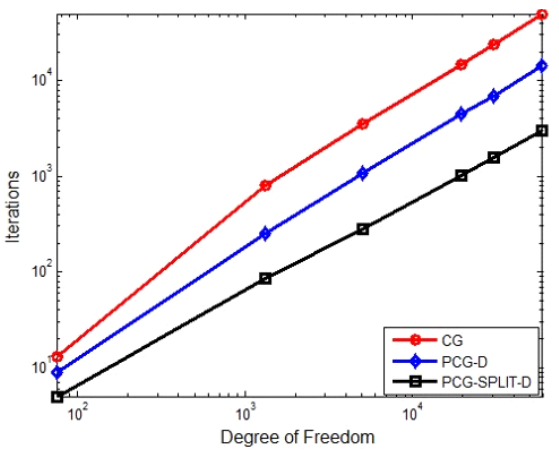

Fig. 13 Comparison of iterations using sparse solvers for non-SPD matrix, marker-line: ○-: CG, ◇-: PCG-D, □-: PCG-split-D, range: 75 DOF ~ 59 643 DOF, (Unit: number)

a little asymmetric matrix (Fig. 12). When the coefficient matrix was SPD (Fig. 10), PCG-split-D was compared with 40 % faster than PCG-D. In sparse solvers, the speed of calculation depended on the non-zero number of the coefficient matrix and the non-SPD matrix, not on the size of the matrix. In Fig. 14, it was compared the computing times according to NNZ of SPD and non-SPD coefficient matrices. In SPD coefficient matrix 20 000 000 DOF, PARDISO (mark: ▼) and MATLAB® sparse '\' (mark: ▲) solvers showed 193 sec and 98 sec, respectively. Using the FEM stiffness matrix, non-SPD matrix, for example, Ex5 (59 643 DOF, NNZ = 1 564 440), system equation was solved by PCG-split-D, resulting in 4.805 sec, which was nearly equivalent time to solve 200 000 DOF SPD matrix (#9) problem. In other words, the sparse solver was not dependent on the size of the coefficient matrix A but on the number of non-zero of the coefficient matrix A. When NNZ was the same, SPD matrix converged faster than the non-SPD matrix in CG method and PCG-D converges equally to SPD or non-SPD matrix. However, in case of equal NNZs of coefficient matrices, it showed that PCG-split-D showed faster convergence in non-SPD matrices than SPD ones.

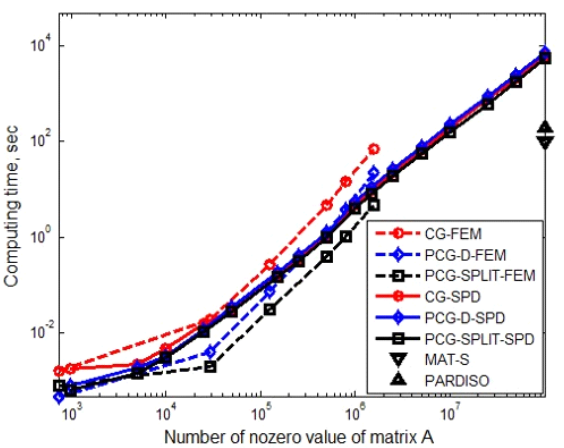

Fig. 14 Comparison of computing time according to NNZ of sparse solvers for SPD and non-SPD matrix, marker-line: ○-: CG-SPD, ◇-: PCG-D-SPD, □-: PCG-split-D-SPD, ○--: CG-FEM, ◇--: PCG-D-FEM, □--: PCG-split-D-FEM, range: 200 DOF ~ 20 000 000 DOF (Unit: sec)

The NNZ of the non-SPD matrix Ex5 (59 643 DOF, NNZ: 1 564 440) was equal to the NNZ of SPD matrix #10 (300 000 DOF, NNZ: 1 498 902). However, the time required for calculating using Ex5 using PCG-split-D was equal to the time for solving #9 (200 000 DOF, NNZ: 999 104). The reason why was that the number of iterations when using PCG-split-D for SPD matrix #9 and #10 was 1533 and 1846, respectively. However, when applying PCG-split-D to non-SPD matrix Ex5, the number of iterations was 2929. The non-SPD matrix Ex5 showed 48 828 and 14 475 iterations, respectively, when CG or PCG-D was used to solve system equation. Consequently, the computing time was the shortest since the number of iterations was smaller than CG or PCG-D when solving non-SPD matrix Ex5 with PCG-split-D.

6. Conclusions

(1) In this paper, we proposed physical coordinated based 3-node triangular FE, 4-node rectangular FE, and 4-node quadrilateral FE codes according to the Kirchhoff thin plate theory.

(2) FE codes using isoparametric quadrilateral FE and 3-D plate shell FE elements based on the Mindlin-Reissner thick plate theory were also presented.

(3) We found that there would be some errors in the FE codes presented by Semie[5] through validation test using a three-node triangular FE codes developed in this paper.

(4) The displacements were compared at the center point of a square plate with four sides fixed and under constant pressure. Three FE codes, which were physical coordinated based 3-node triangular FE, 4-node rectangular FE, and 4-node quadrilateral FE program developed based on the Kirchhoff plate theory, showed within a deviation of 0.9% from the theoretical value obtained by the Levy method. Therefore, three developed FEM codes were proved to be adequate to use.

(5) We also presented results using natural coordinates based isoparametric quadrilateral FE and 3-D plate shell FE codes based on the Mindlin-Reissner

thick plate theory. The displacement was greater than 8 % of the value obtained by the Kirchhoff thin plate theory. The commercial software LISA analysis results were also compared and it was 16% higher than Kirchhoff's theory without taking shear strain into account.

(6) The coefficient matrix from the actual FEM analysis was a non-SPD matrix. In case of 59 643 DOF non-SPD coefficient matrix, it showed that the calculation time was 1218 sec of the MKL library full-matrix direct solver DGESV, 4.805 sec in sparse PCG-split-D and 0.590 sec in sparse MATLAB '\', respectively, indicating that the sparse solvers were superior to the full matrix solvers. Especially, sparse PARDISO took 0.345 sec.

(7) Sparse solvers showed excellent memory saving and computation speed when coefficient matrix was sparse when solving large system equations.

(8) FGMRES and MGMRES were less likely to diverge than GMRES when solving system equations with non-SPD coefficient matrix. However, according to the characteristics of the coefficient matrix, the number of inner loop and the number of outer loop must be predetermined by the user, and divergence or convergence was determined according to these values. Also, if the iteration number of inner loop was large, the solution vector converged rapidly but the memory to be memorized increased and the memory efficiency decreased. Calculation time was needed more than sparse PCG-D or PCG-split-D.

(9) In the case of the SPD coefficient matrix, the calculation performance of PCG-D and PCG-split-D was similar, but in the case of non-SPD coefficient matrix, PCG-split-D showed faster computation time than PCG-D. In the non-SPD matrix, 59 643 DOF, PCG-split-D was five times faster than PCG-D due to faster convergence.

(10) OCTAVE® showed that reading data files was very slower than in MATLAB® or FORTRAN.

(11) It showed that PCG-D and PCG-split-D almost were required the same calculation amount per iteration to solve system equations with SPD coefficient matrix.

(12) If the coefficient matrix was asymmetric, it showed that it may be happened to inaccurate results or divergency in PCG-D, PCG-split-D and PCG ILU(0). The PCG-ILU(τ) using Crout ILU was accurate, but required more memory than ILU(0), which may not be suitable for large-size problem.

(13) Sparse solver was not dependent on the size of the coefficient matrix A but on the number of non-zero of the coefficient matrix A.

(14) In case of system equation with large non-SPD coefficient matrices, it showed that computing times were ordered with PARDISO, MATLAB® sparse operator '\' and sparse PCG-split-D in short time order. But, in case of large SPD matrix, for example, 20 000 000 DOF, sparse MATLAB® '\', PARDISO, and sparse PCG-split-D exhibited 98 sec, 193 sec, and 5632 sec, respectively.

References

(1) Holmström, F., 2001, Structure-acoustic Analysis Using BEM-FEM implementation in MATLAB, M.S. Thesis, Lund University, Lund, Sweden.

(2) Irvine, T., 2011, Plate Bending Frequencies via the Finite Element Method with Rectangular Elements, Revision A.

(3) Irvine, T., 2011, The Natural Frequency of a Rectangular Plate with Fixed-free-fixed-free Boundary Conditions.

(4) Irvine, T., 2011, The Natural Frequency of a Rectangular Plate with Fixed-fixed-fixed-fixed-fixed Boundary Conditions Revision B.

(5) Semie, A. G., 2010, Numerical Modelling of Thin Plates Using the Finite Element Method, M.S. Thesis, Addis Ababa University, Addis Ababa, Ethiopia.

(6) Ferreira, A. J. M., 2009, MATLAB Codes for Finite Element Analysis, New York, NY, Springer.

(7) Kwon, Y. W. and Bang, H. 2000, The Finite Element Method Using MATLAB (2nd edition), Boca Raton, FL, CRC Press.

(8) Timoshenko, S. P. and Woinowsky-Krieger, S. 1989, Theory of Plates and Shells, New York, NY, McGraw-Hill.

(9) Batoz, J.-L., Bathe, K.-J. and Ho, L.-W., 1980, A Study of Three-node Triangular Plate Bending Elements, International Journal for Numerical Methods in Engineering, Vol. 15, No. 12, pp. 1771~1812.

(10) Batoz, J. L. and Tahar, M. B., 1982, Evaluation of a New Quadrilateral Thin Plate Bending Element, International Journal for Numerical Methods in Engineering, Vol. 18, No. 11, pp. 1655~1677.

(11) Bischoff, M., 2008, Modeling of Shells with Three-dimensional Finite Elements, Proceedings of the 6th International Conference on Computation of Shell and. Spatial Structures, pp. 12~15.

(12) Petyt, M., 1990, Introduction to Finite Element Vibration Analysis, Cambridge, England, Cambridge University Press.

(13) Reddy, J. N., 1997, Mechanics of Laminated Composite Plates, New York, NY, CRC Press.

(14) Knight Jr., N. F., 1997, Raasch Challenge for Shell Elements, AIAA Journal, Vol. 35, No. 2, pp. 375~381.

(15) Burden, R. L. and Fairs, J. D., 2011, Numerical Analysis (9th edition), Pacific Grove, CA, Brooks/Cole.

(16) Du, L. and Liu, S., 2015, Study on Preconditioned Conjugate Gradient Methods for Solving Large Sparse Matrix in CSR Format.

(17) Burgerscentrum, J. M., 2012, Iterative Solution Methods.

(18) Nikishkov, G. P., 2004, Introduction to the Finite Element Method (Lecture Notes), Aizu-Wakamatsu, Japan, University of Aizu.

(19) Intel, 2014, Intel® Math Kernel Library (Version 11.2) Reference Manual.

(20) Frydendall, J., 2009, A Parallel Implementation of a Finite Element Solver, IMM-Technical Report-2009-10, Lyngby, Denmark, DTU Informatics.

(21) Saad, Y., 2003, Iterative Methods for Sparse Linear Systems (2nd edition), Philadelphia, PA, Society for Industrial and Applied Mathematics.

(22) Schenk, O. and Gärtner, K., 2018, Parallel Sparse Direct and Multi-recursive Iterative Linear Solvers User's Guide Version 6.1.0.

(23) Shenk O. and Gärtner, K., 2003, Solving Unsymmetric Sparse Systems of Linear Equations with PARDISO, Preprint Submitted to Elsevier Preprint.

(24) W. Layton and M. Sussman, 2014, Numerical Linear Algebra, Morrisville, NC, Lulu Press.

(25) Stahel, A., 2014, Numerical Methods.

(26) http://sep.stanford.edu/sep/claudio/Research/Prst_ExpRefl/ShtPSPI/intel/mkl/10.0.3.020/examples/solver/source/dcsrilut_exampl2.f.

(27) Burkardt, J., 2019, MGMRES: Restarted GMRES Solver for Sparse Linear Systems, https://people.sc.fsu.edu/~jburkardt/m_src/mgmres/mgmres.html.

(28) Mathworks, 2018, pcg (Preconditioned Conjugate Gradients Method), https://kr.mathworks.com/help/matlab/ref/pcg.html#f93-998415.

Appendix

A study on the applicability of PCG method by the asymmetry of coefficient matrix in system equation

We applied PCG method to SPD coefficient matrix #1 ~ #16 (Table 4, Fig. 10) and non-SPD matrix Ex1 ~ Ex5 (Table 5, Fig. 12) similar to symmetric matrix as a whole. And we got solutions within an acceptable accuracy. However, CG or PCG methods should be used with caution when solving system equations with non-SPD matrices other than SPD matrices. We investigated two examples like as followed.

Example 1: In the system equation, $Ax = b$, if the coefficient matrix A is a general matrix with strong asymmetry.

$$A = \begin{bmatrix} 5 & 3 & 1 & 7 & 6 \\ 2 & 6 & 0 & 0 & 4 \\ 0 & 2 & 7 & 3 & 0 \\ 0 & 0 & 0 & 4 & 5 \\ 1 & 0 & 3 & 0 & 7 \end{bmatrix},$$

$$x = \{\, 1 \quad 1 \quad 1 \quad 1 \quad 1 \,\}^t,$$

$$b = \{\, 22 \quad 12 \quad 12 \quad 9 \quad 11 \,\}^t$$

CG, PCG with ILU(0) or PCG with diagonal preconditioner (PCG-D) and PCG with split diagonal preconditioner (PCG-split-D) showed incorrect results but only PCG with Crout version showed correct results (Table A1).

Example 2: If the coefficient matrix is not asym-

metric in the system equation $Ax = b$,

$$A = \begin{bmatrix} 5 & 3 & 1 & 7 & 6 \\ 3 & 6 & 2 & 0 & 4 \\ 0 & 2 & 7 & 3 & 3 \\ 7 & 0 & 3 & 4 & 2 \\ 6 & 4 & 3 & 2 & 7 \end{bmatrix},$$

$$x = \{ 1 \quad 1 \quad 1 \quad 1 \quad 1 \}^t,$$

$$b = \{ 22 \quad 15 \quad 15 \quad 16 \quad 22 \}^t$$

CG only showed the wrong results (Table A1).

Table A1 Analysis results of PCG method according to characteristics of coefficient matrix A

		X(1)	X(2)	X(3)	X(4)	X(5)
Exact sol.		1.	1.	1.	1.	1.
Ex1	CG	3473	-772	1143	-575	-1044
	PCG-D	10 565	-3604	3729	-135 31	3484
	PCG-split-D	10 565	-3604	3729	-135 31	3484
	PCG-ILU(0)	1.012	0.95	0.97	1.12	0.93
	PCG-Crout	1.000	1.000	1.000	1.000	1.000
Ex2	CG	0.998	1.001	1.00	1.01	1.000
	PCG-D	1.000	1.000	1.000	1.000	1.000
	PCG-split-D	1.000	1.000	1.000	1.000	1.000
	PCG-ILU(0)	1.000	1.000	1.000	1.000	1.000
	PCG-Crout	1.000	1.000	1.000	1.000	1.000

Seok-Tae Park, B.S. in Hanyang University, Mechanical Eng., 1984., M.S. in KAIST, Mechanical Eng., 1986., Ph.D. in Ajou University, Systems Eng., 1999. 1986 ~ 1989., KAIST (KIST) Mechanical Eng., researcher. 1989 ~ 1992., Ssangyong Motor Co., senior researcher. 1993 ~ 1999., IAE Automotive Technical Lab. principal researcher. 2000 ~ present, in Chungbuk Health & Science University Assistant Professor. His principal interest is the fields of sound quality index codes programming, noise control, and subjective evaluation, loudspeaker, horn and microspeaker system analysis and design, boundary integral analysis in Acoustics.