A Gentle Introduction to Isogeometric Analysis Part 4: 2-Dimensional Elastostatic Analysis *

Keiji YANASE **

In this study, two-dimensional elastic problems are tackled with the NURBS-based isogeometric analysis. In practice, there is a subtle difference between the conventional FEM approach and the one with NURBS. Accordingly, the knowledge on the finite element analysis is equally applicable to the isogeometric analysis in writing the computer code. Several demonstrations show the superiorities and the interesting properties of the method.

Key Words : B-spline basis function, Finite element method, Stress concentration, MATLAB

1. Formulations with B-spline

Based on the previous studies [1][2][3], we now tackle the numerical simulation in two dimensions. In essence, the geometry of problem can be accurately represented by B-splines and NURBS [1-6]:

$$\mathbf{x}(u,v)=\sum_{i=1}^{I_g} N_i^g(u,v)\mathbf{x}_i \quad \text{or} \quad \begin{bmatrix} x(u,v) \\ y(u,v) \end{bmatrix}=\sum_{I=1}^{I_g} N_i^g(u,v)\begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{1}$$

where $\mathbf{x}_i$ represents the coordinates of control points. The superscript "g" denotes that the basis functions may be different from those describing the displacements.

The strain tensor is rendered as:

$$\boldsymbol{\varepsilon}(u,v) = \begin{bmatrix} \varepsilon_{xx}(u,v) \\ \varepsilon_{yy}(u,v) \\ \gamma_{xy}(u,v) \end{bmatrix} = \begin{bmatrix} \dfrac{\partial u_x(u,v)}{\partial x} \\ \dfrac{\partial u_y(u,v)}{\partial y} \\ \dfrac{\partial u_x(u,v)}{\partial y}+\dfrac{\partial u_y(u,v)}{\partial x} \end{bmatrix} \tag{2}$$

where u_x and u_y are the displacements in the x- and y-direction. The displacement vector can be approximated by B-splines as follows:

$$\mathbf{u}(u,v)=\sum_{i=1}^{I} N_i(u,v)\mathbf{u}_i \quad \text{or} \quad \begin{bmatrix} u_x(u,v) \\ u_y(u,v) \end{bmatrix}=\sum_{i=1}^{I} N_i(u,v)\begin{bmatrix} u_{ix} \\ u_{iy} \end{bmatrix} \tag{3}$$

The use of Eqs. (1) and (3) is sharp contrast to the isoparametric concept of classical finite element method in which the same basis functions are used for the description of geometry and displacement.

To calculate the strain tensor, the derivatives of the displacement vector should be calculated as follows:

$$\begin{bmatrix} \dfrac{\partial u_x(u,v)}{\partial x} \\ \dfrac{\partial u_x(u,v)}{\partial y} \end{bmatrix} = \sum_{i=1}^{I}\begin{bmatrix} \dfrac{\partial N_i(u,v)}{\partial x} \\ \dfrac{\partial N_i(u,v)}{\partial y} \end{bmatrix}\times u_{ix} \tag{4}$$

$$\begin{bmatrix} \dfrac{\partial u_y(u,v)}{\partial x} \\ \dfrac{\partial u_y(u,v)}{\partial y} \end{bmatrix} = \sum_{i=1}^{I}\begin{bmatrix} \dfrac{\partial N_i(u,v)}{\partial x} \\ \dfrac{\partial N_i(u,v)}{\partial y} \end{bmatrix}\times u_{iy} \tag{5}$$

To evaluate Eqs. (4) and (5), we can make use of the following chain rule:

$$\frac{\partial N_i(u,v)}{\partial u} = \frac{\partial N_i(u,v)}{\partial x}\frac{\partial x}{\partial u}+\frac{\partial N_i(u,v)}{\partial y}\frac{\partial y}{\partial u} \tag{6}$$

$$\frac{\partial N_i(u,v)}{\partial v} = \frac{\partial N_i(u,v)}{\partial x}\frac{\partial x}{\partial v}+\frac{\partial N_i(u,v)}{\partial y}\frac{\partial y}{\partial v} \tag{7}$$

Based on Eqs. (6) and (7), we obtain the following equation:

$$\begin{bmatrix} \dfrac{\partial N_i(u,v)}{\partial x} \\ \dfrac{\partial N_i(u,v)}{\partial y} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial x}{\partial u} & \dfrac{\partial y}{\partial u} \\ \dfrac{\partial x}{\partial v} & \dfrac{\partial y}{\partial v} \end{bmatrix}^{-1}\begin{bmatrix} \dfrac{\partial N_i(u,v)}{\partial u} \\ \dfrac{\partial N_i(u,v)}{\partial v} \end{bmatrix} \tag{8}$$

where (cf. Eq. (1)):

$$\begin{bmatrix} \dfrac{\partial x}{\partial u} & \dfrac{\partial y}{\partial u} \\ \dfrac{\partial x}{\partial v} & \dfrac{\partial y}{\partial v} \end{bmatrix} = \begin{bmatrix} \displaystyle\sum_{i=1}^{I_g}\frac{\partial N_i^g(u,v)}{\partial u}x_i & \displaystyle\sum_{i=1}^{I_g}\frac{\partial N_i^g(u,v)}{\partial u}y_i \\ \displaystyle\sum_{i=1}^{I_g}\frac{\partial N_i^g(u,v)}{\partial v}x_i & \displaystyle\sum_{i=1}^{I_g}\frac{\partial N_i^g(u,v)}{\partial v}y_i \end{bmatrix} \tag{9}$$

Now we can evaluate the derivatives of the displacement vector. For convenience, let us rewrite the stain tensor as follows:

*Manuscript received: November 24, 2018

**Department of Mechanical Engineering

$$\boldsymbol{\varepsilon}(u,v) = \begin{bmatrix} \varepsilon_{xx}(u,v) \\ \varepsilon_{yy}(u,v) \\ \gamma_{xy}(u,v) \end{bmatrix} = \begin{bmatrix} [\mathbf{B}_1], [\mathbf{B}_2], \cdots, [\mathbf{B}_1] \end{bmatrix} \begin{bmatrix} [\mathbf{u}_1] \\ [\mathbf{u}_2] \\ \vdots \\ [\mathbf{u}_1] \end{bmatrix} \tag{10}$$

where:

$$[\mathbf{B}_i] = \begin{bmatrix} \dfrac{\partial N_i(u,v)}{\partial x} & 0 \\ 0 & \dfrac{\partial N_i(u,v)}{\partial y} \\ \dfrac{\partial N_i(u,v)}{\partial y} & \dfrac{\partial N_i(u,v)}{\partial x} \end{bmatrix} \quad ; \quad [\mathbf{u}_i] = \begin{bmatrix} u_{ix} \\ u_{iy} \end{bmatrix} \tag{11}$$

For convenience, we rewrite Eq. (10) as:

$$\boldsymbol{\varepsilon}(u,v) = \begin{bmatrix} \varepsilon_{xx}(u,v) \\ \varepsilon_{yy}(u,v) \\ \gamma_{xy}(u,v) \end{bmatrix} = \mathbf{B}\mathbf{u} \tag{12}$$

Based on Hooke's law, we calculate the stress tensor. For plane stress condition, it is rendered as:

$$\boldsymbol{\sigma} = \mathbf{D}\boldsymbol{\varepsilon} \quad \text{or} \quad \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{bmatrix} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-\nu)/2 \end{bmatrix} \begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy} \end{bmatrix} \tag{13}$$

For plain strain condition, it is rendered as:

$$\boldsymbol{\sigma} = \mathbf{D}\boldsymbol{\varepsilon} \quad \text{or} \quad \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{bmatrix} = \frac{E}{1-\nu^2} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & (1-2\nu)/2 \end{bmatrix} \begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy} \end{bmatrix} \tag{14}$$

where E is Young's modulus and ν is Poisson's ratio. Therefore, the strain tensor is rendered as:

$$\boldsymbol{\sigma} = \mathbf{D}\boldsymbol{\varepsilon} = \mathbf{D}\mathbf{B}\mathbf{u} \tag{15}$$

Thus, the stored energy, U, is rendered as:

$$\begin{aligned} U &= \frac{1}{2}\int_V \boldsymbol{\sigma}^T \boldsymbol{\varepsilon}\, dV = \frac{1}{2}\int_V (\mathbf{D}\mathbf{B}\mathbf{u})^T \mathbf{B}\mathbf{u}\, dV \\ &= \frac{1}{2}\int_V \mathbf{u}^T \mathbf{B}^T \mathbf{D}\mathbf{B}\mathbf{u}\, dV \end{aligned} \tag{16}$$

It is noted that $\mathbf{D}^T = \mathbf{D}$. On the other hand, by anchoring the points where the concentrated external force is applied, the external virtual work, W, is rendered as:

$$W = \begin{bmatrix} [\mathbf{u}_1]^T, [\mathbf{u}_2]^T, \cdots, [\mathbf{u}_1]^T \end{bmatrix} \begin{bmatrix} [\mathbf{F}_1] \\ [\mathbf{F}_2] \\ \vdots \\ [\mathbf{F}_1] \end{bmatrix} = \mathbf{u}^T \mathbf{F} \tag{17}$$

where:

$$[\mathbf{u}_i] = \begin{bmatrix} u_{ix} \\ u_{iy} \end{bmatrix} \quad ; \quad [\mathbf{F}_i] = \begin{bmatrix} F_{ix} \\ F_{iy} \end{bmatrix} \tag{18}$$

Thus, the potential energy is rendered as:

$$\begin{aligned} \Pi &= U - W \\ &= \frac{1}{2}\int_V \mathbf{u}^T \mathbf{B}^T \mathbf{D}\mathbf{B}\mathbf{u}\, dV - \mathbf{u}^T \mathbf{F} \end{aligned} \tag{19}$$

Then, the theorem of minimum potential energy renders the following equation:

$$\frac{\partial \Pi}{\partial \mathbf{u}} = 0 \quad \rightarrow \quad \left(\int_V \mathbf{B}^T \mathbf{D}\mathbf{B}\, dV\right)\mathbf{u} = \mathbf{F} \tag{20}$$

We rewrite Eq. (20) as:

$$\mathbf{K}\mathbf{u} = \mathbf{F} \tag{21}$$

Here, $\mathbf{K}$ signifies the global stiffness matrix.

Based on **Fig. 1**, we can express the following vectors in the *x*-*y* coordinate as follows:

$$\overrightarrow{O^*P^*} = \left(\frac{\partial x}{\partial u}du,\ \frac{\partial y}{\partial v}du\right) \quad ; \quad \overrightarrow{O^*Q^*} = \left(\frac{\partial x}{\partial v}dv,\ \frac{\partial y}{\partial v}dv\right) \tag{22}$$

Then, the infinitesimal area, dA, in the *x*-*y* coordinate can be calculated as:

$$\begin{aligned} dA &= \begin{vmatrix} 0 & 0 & 1 \\ \dfrac{\partial x}{\partial u}du & \dfrac{\partial y}{\partial u}du & 0 \\ \dfrac{\partial x}{\partial v}dv & \dfrac{\partial y}{\partial v}dv & 0 \end{vmatrix} = \left(\frac{\partial x}{\partial u}\frac{\partial y}{\partial v} - \frac{\partial x}{\partial v}\frac{\partial y}{\partial u}\right)du(\xi)dv(\eta) \\ &= J_u du(\xi)dv(\eta) \\ &= J_u\left(\frac{du}{d\xi}d\xi\right)\left(\frac{dv}{d\eta}d\eta\right) \\ &= J_u J_\xi d\xi d\eta \end{aligned} \tag{23}$$

where:

$$J_u = \frac{\partial x}{\partial u}\frac{\partial y}{\partial v} - \frac{\partial x}{\partial v}\frac{\partial y}{\partial u} \quad ; \quad J_\xi = \frac{du}{d\xi}\frac{dv}{d\eta} \tag{24}$$

Therefore, by taking advantage of Gauss integration, the stiffness matrix can be evaluated as follows:

$$\begin{aligned}\mathbf{K} &= \int_V \mathbf{B}^T\mathbf{D}\mathbf{B}\, dV \\ &= t\int_A \mathbf{B}^T\mathbf{D}\mathbf{B}\, dA \\ &= t\int_{-1}^{+1}\int_{-1}^{+1}\mathbf{B}^T(\xi,\eta)\mathbf{D}\mathbf{B}(\xi,\eta)J_u J_\xi\, d\xi d\eta \\ &= t\sum_{i=1}^{I}\sum_{j=1}^{J}\mathbf{B}^T(\xi_i,\eta_j)\mathbf{D}\mathbf{B}(\xi_i,\eta_j)J_u J_\xi W_i W_j\end{aligned} \tag{25}$$

where t is the thickness and W_i is the weight for the Gauss integration.

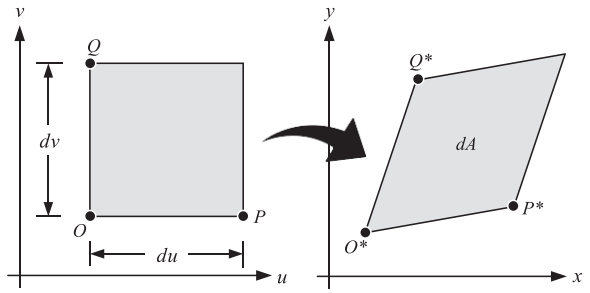


Fig. 1. Infinitesimal area with coordinate transformation.

2.1. Example (1): Rectangular Plate

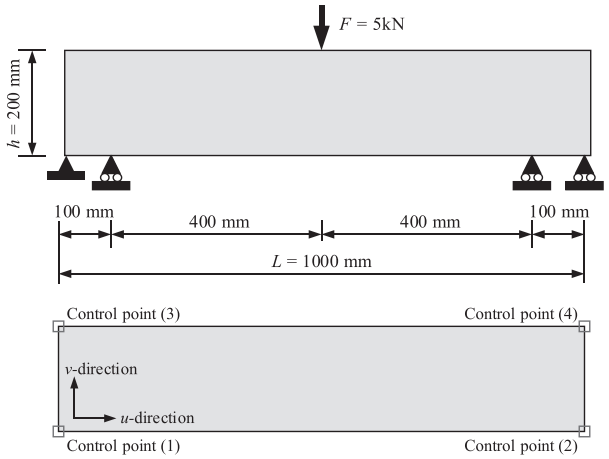


Fig. 2. Geometry of the problem and description of the geometry with one NURBS patch.

Let us consider a simple example to understand the differences between the classical FEM and the NURBS-based analysis. Here, we tackle a plate supported at edges and subjected to a concentrated force, as shown in **Fig. 2** [1]. The plate is made of a carbon steel and the materials properties are E = 206 GPa and $\nu = 0.3$. The plain stress condition is assumed.

For the analysis with B-spline, we first define the geometry by using the linear basis functions (p = 1). The corresponding knot vectors are given as:

$$\Xi_u^g = \Xi_v^g = [0, 0, 1, 1] \tag{26}$$

Thus, based on Eq. (1), the geometry can be expressed as:

$$\begin{bmatrix} x(u,v) \\ y(u,v) \end{bmatrix} = \sum_{I=1}^{4} N_i^g(u,v)\begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{27}$$

where (cf. **Fig. 2**):

$$\begin{aligned} N_1^g(u,v) &= N_{1,1}(u)\times N_{1,1}(v) \\ N_2^g(u,v) &= N_{2,1}(u)\times N_{1,1}(v) \\ N_3^g(u,v) &= N_{1,1}(u)\times N_{2,1}(v) \\ N_4^g(u,v) &= N_{2,1}(u)\times N_{2,1}(v) \end{aligned} \tag{28}$$

$$\begin{aligned} [x_1, x_2, x_3, x_4] &= [0, L, 0, L] \\ [y_1, y_2, y_3, y_4] &= [0, 0, h, h] \end{aligned} \tag{29}$$

By considering the sub-regions, we can write u and v as (**Fig. 3**):

$$\begin{aligned} u &= \left(\frac{1-\xi}{2}\right)u_1^{(s)} + \left(\frac{1+\xi}{2}\right)u_2^{(s)} \\ v &= \left(\frac{1-\eta}{2}\right)v_1^{(s)} + \left(\frac{1+\eta}{2}\right)v_2^{(s)} \end{aligned} \tag{30}$$

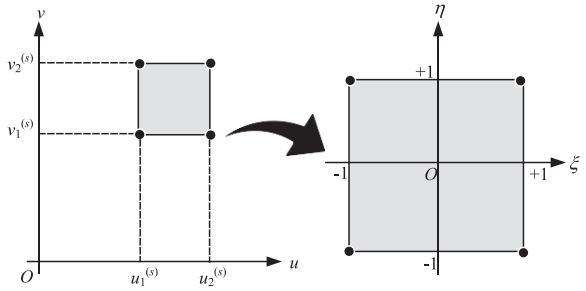
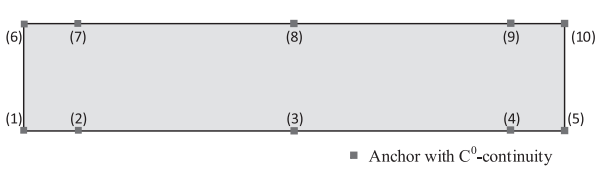
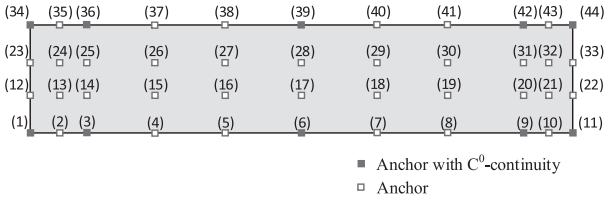


Fig. 3. Coordinate transformation.



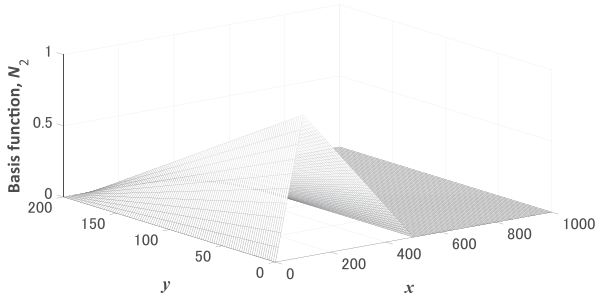
$$\Xi_u = [0, 0, 0.1, 0.5, 0.9, 1, 1], \quad p = 1$$
$$\Xi_v = [0, 0, 1, 1], \quad p = 1$$

(a) Position of anchors

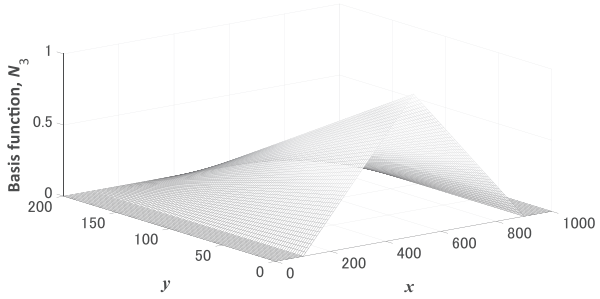


$$\Xi_u = [0, 0, 0, 0.1, 0.1, 0.3, 0.5, 0.5, 0.7, 0.9, 0.9, 1, 1, 1], \quad p = 2$$
$$\Xi_v = [0, 0, 0, 0.5, 1, 1, 1], \quad p = 2$$

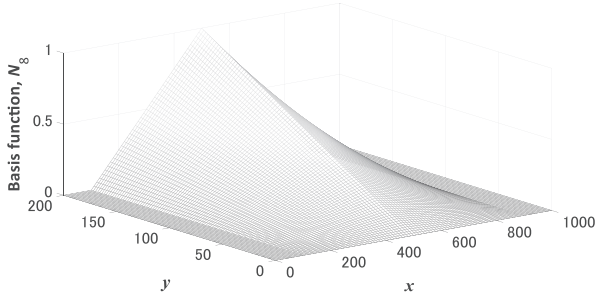
(a) Position of anchors



(a) Basis function, N_2



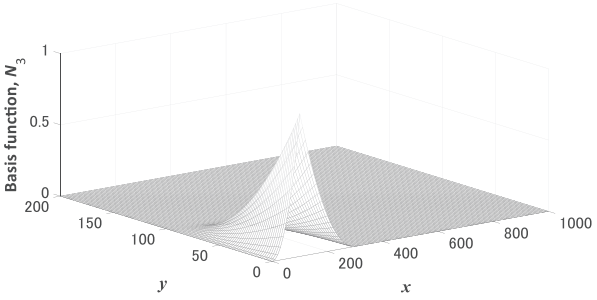
(b) Basis function, N_3



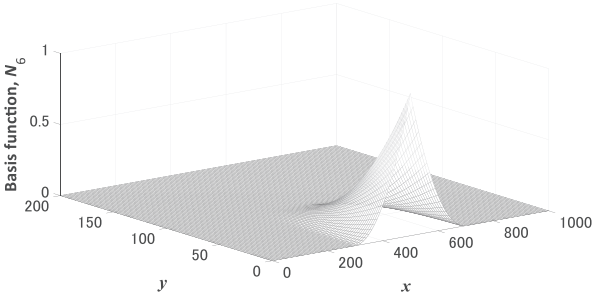
(c) Basis function, N_8

Fig. 4. Before *k*-refinement.

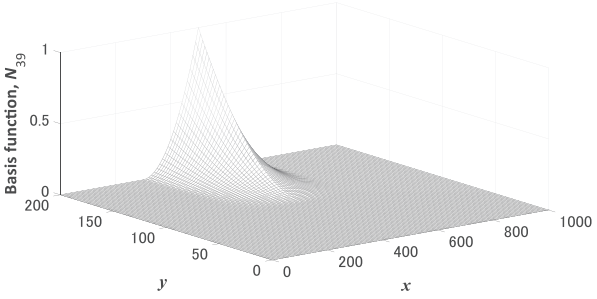
Figs. 4 and **5** show the basis functions in the three-dimensional space. It is noted that the coordinate transformation is applied with Eq. (27). As shown, in conjunction with refinement, the span of basis function becomes smaller.



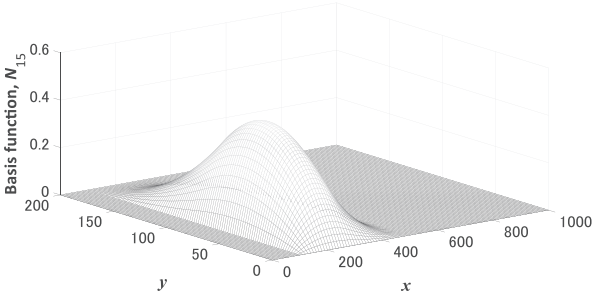
(b) Basis function, N_3



(c) Basis function, N_6



(d) Basis function, N_{39}



(e) Basis function, N_{15}

Fig. 5. After *k*-refinement.

To implement the calculation, we need to prepare the data for the geometry and boundary conditions and the material properties. For example, please see the following program that is related to **Fig. 4**(a):

```
function[Binfo,knotg,knot,t,Dmat] = Input_data3a()

%-------------------------------------------------------------
% Set up the information for boundary conditions

%== Output ==
% Binfo = {Bcode,Bvalue,Dest0,Dest1,Ncp0,Ncp1,Ncp}
%    Bcode   ... Boundary code (0:Newmann BC; 1:Drichlet BC)
%    Bvalue ... Boundary value
%                 (displacement and force at each control point)
%    Dest0   ... Destination vector for Newmann BC (Bcode = 0)
%    Dest1   ... Destination vector for Dirichlet BC (Bcode = 1)
%    Ncp0    ... Number of Newmann BCs (x & y)
%    Ncp1    ... Number of Dirichlet BCs (x & y)
%    Ncp     ... Number of control points

% knotg   ... Geometry data
% knot    ... Knot vector for field variables
% t         ... Thickness
% Dmat    ... D-matrix
%-------------------------------------------------------------

%== Compute D-matrix ==
E   = 206*10^3;   %=>> Young's modulus (MPa)
nu = 0.3;          %=>> Poisson's ratio
Dmat = E/(1-nu^2)*[ 1, nu, 0 ;
                    nu,   1, 0 ;
                     0,   0, (1-nu)/2];

%== Geometry data ==
L = 1000;   %=>> Length (mm)
h = 200;    %=>> Hight (mm)
t = 20;     %=>> Thickness (mm)

knotug = [0,0,1,1]; pug = 1;
%=>> Knot vector and polynomial order in u
knotvg = [0,0,1,1]; pvg = 1;
%=>> Knot vector and polynoimial order in v

xg = [0,L, 0,L];   %=>> Control-points in x-coordinate
yg = [0,0, h,h];   %=>> Control-points in y-coordinate

%== Knot vectors for field variables ==
knotu = [0,0, 0.1, 0.5, 0.9, 1,1]; pu = 1;
knotv = [0,0, 1,1]; pv = 1;
Ncp = (length(knotu)-pu-1) * (length(knotv)-pv-1);
%=>> Number of control points

%== Create the structures for convenience ==
knotg = {knotug,pug, knotvg,pvg, xg,yg};   %=>> For geometry
knot   = {knotu ,pu , knotv ,pv };              %=>> For field variables

%== Boundary conditions ==
Bcode   = zeros(1,2*Ncp);
Bvalue = zeros(1,2*Ncp);

Bcode(2*1-1) = 1;   %=>> Dirichlet BC
Bcode(2*1   ) = 1;
Bcode(2*2   ) = 1;
Bcode(2*4 ) = 1;
Bcode(2*5 ) = 1;

Bvalue(2*8) = -50*10^3;   %=>> Applied force in y-direction (N)

Ncp1 = sum(Bcode);          %=>> Number of Dirichlet BCs (x & y)
Ncp0 = 2*Ncp - Ncp1;       %=>> Number of Newmann BCs (x & y)
Dest1 = zeros(1,Ncp1);     %=>> Destination vector for Dirichlet
BCs
Dest0 = zeros(1,Ncp0);   %=>> Destination vector for Newmann
BCs
J = 1; K = 1;
for I = 1:Ncp
  if Bcode(2*I-1) == 1   %=>> Check BCs in x-direction
    Dest1(J) = 2*I-1; J = J + 1;
  else
    Dest0(K) = 2*I-1; K = K + 1;
  end;

  if Bcode(2*I) == 1   %=>> Check BCs in y-direction
    Dest1(J) = 2*I; J = J + 1;
  else
    Dest0(K) = 2*I; K = K + 1;
  end;
end;

%== Create a structure for convenience ==
Binfo = {Bcode,Bvalue,Dest0,Dest1,Ncp0,Ncp1,Ncp};
```

To calculate the stiffness matrix, the following program is prepared. Since it is a function file, the main program can be effectively simplified, as shown later.

```
function[Kij] = Stiffness_matrix(knotg,knot,t,Dmat,Ng)

%-------------------------------------------------------------
% Compute the stiffness matrix

%== Input ==
% knotg ... Information for geometry data
% knot   ... Knot vector and polynomial order for field variables
% t      ... Thickness
% Dmat   ... D-matrix (stiffness matrix)
% Ng     ... Number of Gauss-points (sampling points)

%== Output ==
% Kij    ... Stiffness matrix
%-------------------------------------------------------------

%== Gauss point and weight ==
Gpt = [          0,          0,          0,          0;
        -0.577350,  0.577350,          0,          0;
        -0.774967,          0,  0.774967,          0;
        -0.861136, -0.339981,  0.339981,  0.861136];
%=>> Gauss point(sampling point)

Gwt = [2.000000,  0.000000, 0.000000, 0.000000 ;
        1.000000,  1.000000, 0.000000, 0.000000 ;
        0.555556,  0.888889, 0.555556, 0.000000 ;
        0.347855,  0.652145, 0.652145, 0.347855];   %=>> Weight

%== Knot vector & polynomial order in u for geometry ==
knotug = knotg{1}; pug = knotg{2};

%== Knot vector & polynomial order in v for geometry ==
knotvg = knotg{3}; pvg = knotg{4};

%== Control points for geometry ==
xg = knotg{5}; yg = knotg{6};

%== Knot vector & polynomial order in u for field variables ==
knotu = knot{1}; pu = knot{2};

%== Knot vector & polynomial order in v for field variables ==
knotv = knot{3}; pv = knot{4};

%== Set up for sub-regions ==
usub = [0,knotu(pu+2)];   %=>> Set up for sub-regions (1)
for I = (pu+3):length(knotu)
  if knotu(I) > max(usub)
    usub = [usub, knotu(I)];
  end;
end;

vsub = [0,knotv(pv+2)];   %=>> Set up for sub-regions (2)
for I = (pv+3):length(knotv)
  if knotv(I) > max(vsub)
    vsub = [vsub, knotv(I)];
  end;
end;

Nsub = (length(usub)-1)*(length(vsub)-1);
%=>> Number of sub-regions
nu  = length(knotu)-pu-1;   %=>> Number of control points in u
nv  = length(knotv)-pv-1;   %=>> Number of control points in v
Ncp = nu*nv;   %=>> Number of control points


%== Control points in sub-regions ==
us = zeros(1,Nsub);   ue = zeros(1,Nsub);
vs = zeros(1,Nsub);   ve = zeros(1,Nsub);

K = 1;
for J = 1:(length(vsub)-1)
  for I = 1:(length(usub)-1)
    us(1,K) = usub(I);      %=>> Start point in u for a sub-region
    ue(1,K) = usub(I+1);    %=>> End point in u
    vs(1,K) = vsub(J);      %=>> Start point in v
    ve(1,K) = vsub(J+1);    %=>> End point in v
    K = K+1;
  end;
end;


Kij = zeros(2*Ncp, 2*Ncp);   %=>> Stiffness matrix
for I = 1:Nsub   %=>> Sub-region loop;

  for J1 = 1:Ng   %=>> Gauss-integration loop (1)
    for J2 = 1:Ng   %=>> Gauss-integration loop (2)
      xsi = Gpt(Ng,J1); W1 = Gwt(Ng,J1);
      %=>> Gauss point & weight in xsi
      eta = Gpt(Ng,J2); W2 = Gwt(Ng,J2);
      %=>> Gauss point & weight in eta

      u = (1-xsi)/2*us(I) + (1+xsi)/2*ue(I);
      %=>> Transformation to u
      v = (1-eta)/2*vs(I) + (1+eta)/2*ve(I);
      %=>> Transformation to v

      %== Let us consider the basis functions for geometry ==
      [Nu, dN1u] = Bsp_deriv(knotug, pug, u);
      [Nv, dN1v] = Bsp_deriv(knotvg, pvg, v);
```

```
    dxdu = [dN1u'.*Nv(1), dN1u'.*Nv(2)]*xg';
    dydu = [dN1u'.*Nv(1), dN1u'.*Nv(2)]*yg';

    dxdv = [Nu'.*dN1v(1), Nu'.*dN1v(2)]*xg';
    dydv = [Nu'.*dN1v(1), Nu'.*dN1v(2)]*yg';

    Jac = [dxdu, dydu ;   %=>> Jacobian matrix
           dxdv, dydv ];

    Ju  = Jac(1,1)*Jac(2,2) - Jac(1,2)*Jac(2,1);
    Jxsi = (ue(l)-us(l))/2*(ve(l)-vs(l))/2;

    %== Let us consider the basis functions for displacement ==
    [Nu, dN1u] = Bsp_deriv(knotu,pu,u);
    [Nv, dN1v] = Bsp_deriv(knotv,pv,v);

    %== Compute the 1st derivatives at each control point ==
    dNdu = dN1u'.*Nv(1);
    dNdv = Nu'.*dN1v(1);
    for K = 2:nv
      dNdu = [dNdu, dN1u'.*Nv(K)];
      dNdv = [dNdv, Nu'.*dN1v(K)];
    end;

    %== Overall weight at a Gauss-point in a sub-region ==
    Weight = W1*W2*Ju*Jxsi;

    Bmat = zeros(3,2*Ncp);   %=> B-matrix at each control point
    for K = 1:Ncp
      Drv = Jac¥[dNdu(K); dNdv(K)];
      dNdx = Drv(1);
      dNdy = Drv(2);
      Bmat(:,2*K-1:2*K) = [dNdx,    0 ;   %=>> B-matrix
                             0   , dNdy;
                           dNdy, dNdx] ;
    end;

    %== Compute the stiffness matrix with Gauss-integration ==
    Kij = (Bmat'*Dmat*Bmat)*Weight*t + Kij;

  end;   %=>> End of Gauss-integration (2)
 end;   %=>> End of Gauss-integration (1)

end;   %=>> End of sub-region-loop
```

With the aid of those function files, the problem shown in **Fig. 2** can be finally simulated by the following main program.

```
%************************************************************
%** Solve the two-dimensional elastostatic problem with IGA **
%************************************************************
%== Obtain the input data ==
[Binfo,knotg,knot,t,Dmat] = Input_data3a();

%== Binfo:{Bcode,Bvalue,Dest0,Dest1,Ncp0,Ncp1,Ncp} ==
Bcode  = Binfo{1}; Bvalue = Binfo{2};
Dest0  = Binfo{3}; Dest1  = Binfo{4};
Ncp0   = Binfo{5}; Ncp1   = Binfo{6}; Ncp  = Binfo{7};

%== Compute the stiffness matrix ==
Ng = 4;   %=>> Number of Gauss-point
[Kij] = Stiffness_matrix(knotg, knot, t, Dmat, Ng);

%== Compute the modified stiffness matrix ==
Kmod = Kij(Dest0, Dest0);

%== Compute the modified force vector ==
Fmod = zeros(Ncp0,1);
for I = 1:Ncp0
  K = Dest0(I);
  Fmod(I) = Bvalue(K);

  for J = 1:Ncp1
    L = Dest1(J);
    Fmod(I) = Fmod(I) - Kij(K,L)*Bvalue(L);
  end;
end;

%== Obtain the solutions ==
umod = Kmod¥Fmod;
usol = zeros(2*Ncp,1);   %=>> Solution of displacement vector
K = 1;
for I = 1:2*Ncp
  if Bcode(I) == 0   %=>> Displacament is originally unknown
    usol(I) = umod(K); K = K + 1;
  else
    usol(I) = Bvalue(I);
  end;
end;
Fsol = Kij*usol;   %=>> Solution for force vector
```

Fig. 6 shows the variations of maximum deflection with the increase of degree of freedom, DOF. As a reference, the solutions with finite element method (FEM) is provided. This FE analysis was conducted with the Excel-VBA program [10]. As clearly shown, the convergence of solution with isogeometric analysis (IGA) exhibits a superior performance than the classical FEM. It is noted that the preparation of geometry data is much simpler in IGA.

2.2. Example (2): Plate with a circular hole under uniform tension

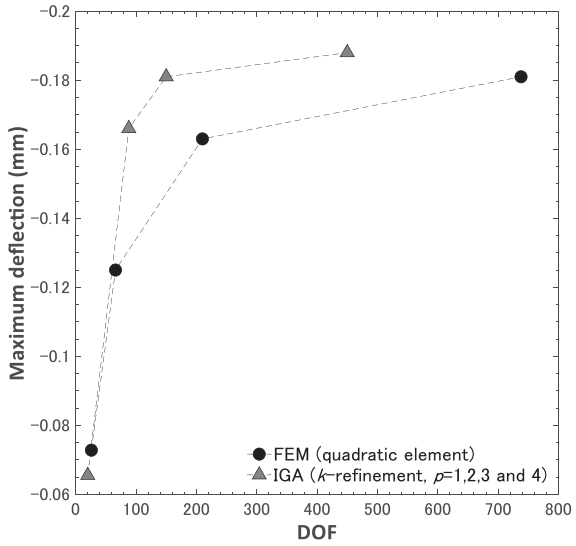


Fig. 6. Convergence of maximum deflection.

In some cases, it is convenient to use more than one NURBS patches to describe the geometry. Since there are no nodal points, the compatibility conditions have to be prescribed appropriately. Accordingly, C^0-compatibility between patches needs to be ensured if the location of anchors match along the line where patches connect. These conditions can be enforced by using incidences for the assembly of coefficient. The incidences refer to the anchors in IGA, instead of nodal points in the classical FEM.

In practice, NURBS are particularly well suited to linear elasticity. It is obvious that the accurate geometrical representation at all levels of discretization should lead to improved accuracy as compared with less geometrically accurate methods. In this two-dimensional example, we use the NURBS-based isogeometric analysis for a problem in solid mechanics having an exact solution: an infinite plate with a circular hole subjected to uniform tension (**Fig. 7**). The exact solution for infinite plate is rendered as [4][5] [11]:

$$\begin{aligned}
\sigma_{rr}(r,\theta) &= \frac{\bar{\sigma}}{2}\left\{1-\frac{R^2}{r^2}-\left(1-4\frac{R^2}{r^2}+3\frac{R^4}{r^4}\right)\cos 2\theta\right\} \\
\sigma_{\theta\theta}(r,\theta) &= \frac{\bar{\sigma}}{2}\left\{1+\frac{R^2}{r^2}+\left(1+3\frac{R^4}{r^4}\right)\cos 2\theta\right\} \\
\sigma_{r\theta}(r,\theta) &= \frac{\bar{\sigma}}{2}\left\{1+2\frac{R^2}{r^2}-3\frac{R^4}{r^4}\right\}\sin 2\theta
\end{aligned} \tag{31}$$

where $\bar{\sigma}$ is the uniform stress applied at infinity and R is the hole radius. For convenience, we analyze a quarter plate by considering the symmetry boundary condition. For analysis, the following input data are used:

Length (L) = 50 mm, height (H) = 50 mm
Thickness (t) = 1 mm, hole radius (R) = 10 mm
E = 206 GPa, ν = 0.3 (carbon steel)

In practice, to deal with the problem of *infinite plate* by using the *finite quarter plate*, the stresses given by Eq. (31) are adopted as Neumann boundary conditions (i.e., prescribed tractions) at the boundary of the finite quarter plate (**Fig. 7**). Under this circumstance, it may be convenient to express the stresses in the x- and y-coordinate. Then, after the coordinate transformation [11], Eq. (31) becomes:

$$\begin{aligned}
\sigma_{xx}(x,y) &= \frac{\bar{\sigma}}{2}\left\{\frac{R^2}{r^2}\cos 2\theta+\left(2\frac{R^2}{r^2}-3\frac{R^4}{r^4}\right)\cos 4\theta\right\} \\
\sigma_{yy}(x,y) &= \frac{\bar{\sigma}}{2}\left\{2+3\frac{R^2}{r^2}\cos 2\theta-\left(2\frac{R^2}{r^2}-3\frac{R^4}{r^4}\right)\cos 4\theta\right\} \\
\sigma_{xy}(x,y) &= \frac{\bar{\sigma}}{2}\left\{-\frac{R^2}{r^2}+\left(4\frac{R^2}{r^2}-6\frac{R^4}{r^4}\right)\cos 2\theta\right\}\sin 2\theta
\end{aligned} \tag{32}$$

where:

$$r(x,y)=\sqrt{x^2+y^2} \quad ; \quad \theta(x,y) = \tan^{-1}\left(\frac{y}{x}\right) \tag{33}$$

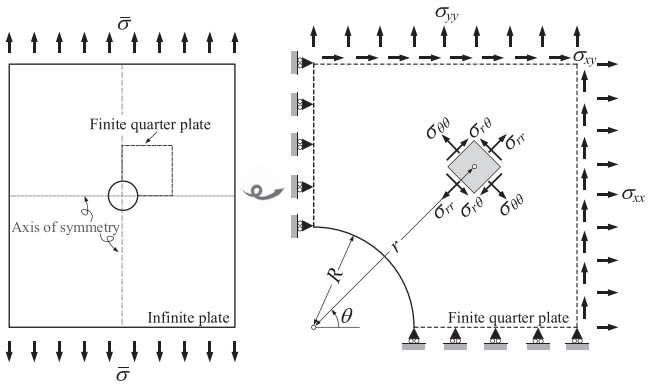


Fig. 7. Infinite plate and finite quarter plate.

We start with the description of the geometry with 2 NURBS patches with 10 control points, as shown in **Fig. 8**. We can reproduce **Fig. 8** with the following MATLAB code.

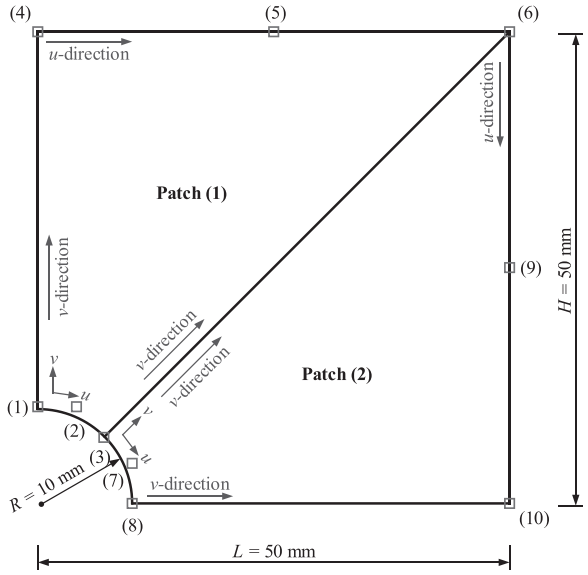
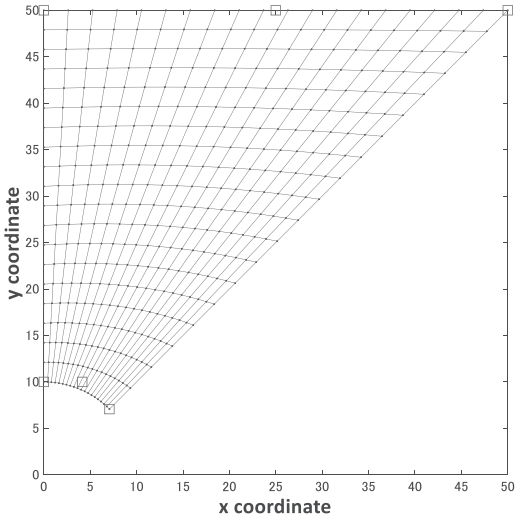


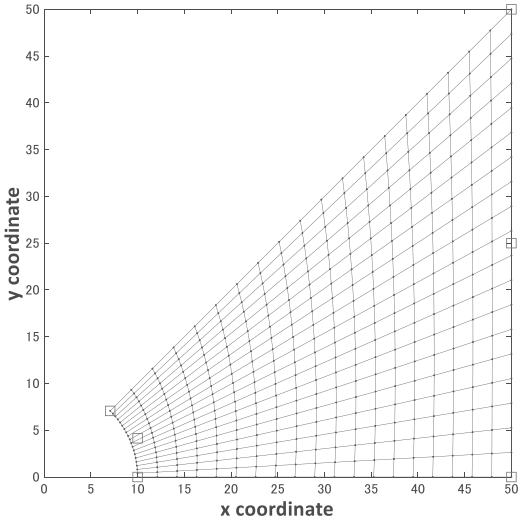
Fig. 8. Description of geometry with two NURBS patches.

```
%== Data for control points ==
% (Homogenized coordinates (coordinates are multiplied by weight)
% Cp = [x; y; z; 1]*weight
H = 50; L = 50; R = 10;
Cp = zeros(4,10);
Cp(:,1)  = [0; R; 0; 1];                    %=>> Control point (1)
Cp(:,2)  = [0.4142*R; R; 0; 1]*0.9239; %=>> Control point (2)
Cp(:,3)  = [0.707*R; 0.7071*R; 0; 1];   %=>> Control point (3)
Cp(:,4)  = [0; H; 0; 1];                    %=>> Control point (4)
Cp(:,5)  = [L/2; H; 0; 1];                  %=>> Control point (5)
Cp(:,6)  = [L; H; 0; 1];                    %=>> Control point (6)
Cp(:,7)  = [R; 0.4142*R; 0; 1]*0.9239; %=>> Control point (7)
Cp(:,8)  = [R; 0; 0; 1];                    %=>> Control point (8)
Cp(:,9)  = [L; H/2; 0; 1];                  %=>> Control point (9)
Cp(:,10) = [L; 0; 0; 1];                    %=>> Control point (10)

%== NURBS patch(1) ==
coefs1 = cat(3, [Cp(:,1),Cp(:,2),Cp(:,3)],[Cp(:,4),Cp(:,5),Cp(:,6)]);
%== NURBS patch(2) ===
coefs2 = cat(3,[Cp(:,3),Cp(:,7),Cp(:,8)],[Cp(:,6),Cp(:,9),Cp(:,10)]);

%== Knot vector for geometry ==
knotug = [0,0,0, 1,1,1];
knotvg = [0,0, 1,1];
knotg  = {knotug,knotvg};

%== Create information for NURBS with NURBS toolbox ==
nurbs1 = nrbmak(coefs1, knotg);  %=>> NURBS patch(1)
nurbs2 = nrbmak(coefs2, knotg);  %=>> NURBS patch(2)

%== Intrinsic coordinate ==
Npoint = 20;
u = linspace(0,1,Npoint);
v = linspace(0,1,Npoint);

%== Create the geometry with NURBS tool box ==
xyz1 = nrbeval(nurbs1,{u,v});  %=>> NURBS patch(1)
xyz2 = nrbeval(nurbs2,{u,v});  %=>> NURBS patch(2)

%== Rearrange the data for output ==
xcoor1 = reshape(xyz1(1,:,:),length(u),length(v));
%=>> x for NURBS patch(1)
ycoor1 = reshape(xyz1(2,:,:),length(u),length(v));
%=>> y for NURBS patch(1)
xcoor2 = reshape(xyz2(1,:,:),length(u),length(v));
%=>> x for NURBS patch(2)
ycoor2 = reshape(xyz2(2,:,:),length(u),length(v));
%=>> y for NURBS patch(2)

%=========================================
%== Plot the figures for NURBS patch ==
%=========================================
FS = 'Fontsize'; FN = 'Fontname'; FW = 'Fontweight';

%---------------------------------------------------------
figure(1) %=>> Plot for NURBS patch(1)
```

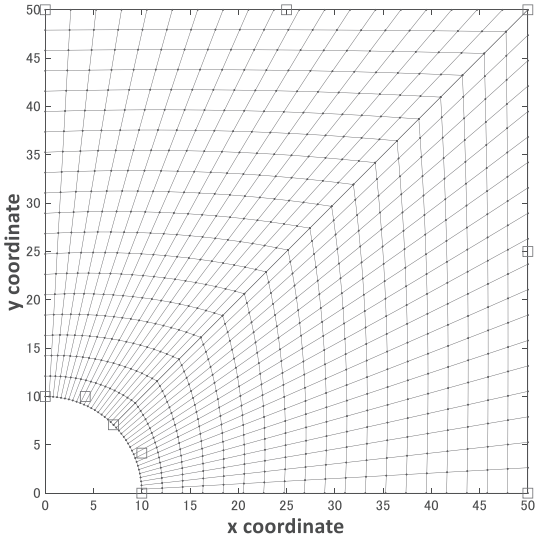
```
for I = 1:length(u)
      plot(xcoor1(I,:), ycoor1(I,:), 'b.-'); hold on;
      plot(xcoor1(:,I), ycoor1(:,I), 'b.-');
end;
plot(Cp(1,1:6)./Cp(4,1:6), Cp(2,1:6)./Cp(4,1:6), 'rs',...
      'Markersize', 15, 'Linewidth',1); hold off;
axis equal; axis([0,L,0,H]); set(gca, 'Fontsize',18);
xx = xlabel('x coordinate', FS,28, FN, 'calibri', FW,'bold');
yy = ylabel('y coordinate', FS,28, FN, 'calibri', FW,'bold');

%-------------------------------------------------------------
figure(2) %=>> Plot for NURBS patch(2);
for I = 1:length(u)
      plot(xcoor2(I,:), ycoor2(I,:), 'b.-'); hold on;
      plot(xcoor2(:,I), ycoor2(:,I), 'b.-');
end;
plot(Cp(1,[3,7,8,6,9,10])./Cp(4,[3,7,8,6,9,10]), ...
       Cp(2,[3,7,8,6,9,10])./Cp(4,[3,7,8,6,9,10]), 'rs',...
      'Markersize', 15, 'Linewidth',1); hold off;
axis equal; axis([0,L,0,H]); set(gca, 'Fontsize',18);
xlabel('x coordinate', FS,28, FN, 'calibri', FW,'bold');
ylabel('y coordinate', FS,28, FN, 'calibri', FW,'bold');

%-------------------------------------------------------------
figure(3) %=>> Plot for NURBS patch(1)&(2);
for I = 1:length(u)
      plot(xcoor1(I,:), ycoor1(I,:), 'b.-'); hold on;
      plot(xcoor1(:,I), ycoor1(:,I), 'b.-');
      plot(xcoor2(I,:), ycoor2(I,:), 'b.-');
      plot(xcoor2(:,I), ycoor2(:,I), 'b.-');
end;
plot(Cp(1,:)./Cp(4,:),Cp(2,:)./Cp(4,:), 'rs',...
      'Markersize', 15, 'Linewidth',1); hold off;
axis equal; axis([0,L,0,H]); set(gca, 'Fontsize',18);
xlabel('x coordinate', FS,28, FN, 'calibri', FW,'bold');
ylabel('y coordinate', FS,28, FN, 'calibri', FW,'bold');
```



(a) NURBS patch (1)



(b) NURBS patch (2)



(c) Assemblage of NURBS patches (1) & (2)

Fig. 9. Created geometry with MATLAB code.

Fig. 9 shows the NURBS patches created with the MATLAB code. Because of C^0-compatibility along the connection of patches (**Figs. 9**(a) and (b)), the assemblage of NURBS patches (**Fig. 9**(c)) can properly reproduce the original geometry.

Since the NURBS basis functions are involved in the analysis, let us consider its first derivatives. The NURBS basis functions are rendered as [4][5]:

$$R_{i,j}^{p,q}(u,v) = \frac{N_{i,p}(u)\cdot N_{j,q}(v)\cdot w_{i,j}}{\sum_{\hat{i}=1}^{I}\sum_{\hat{j}=1}^{J} N_{\hat{i},p}(u)\cdot N_{\hat{j},q}(v)\cdot w_{\hat{i},\hat{j}}} = \frac{N_{i,p}(u)\cdot N_{j,q}(v)\cdot w_{i,j}}{W} \tag{34}$$

For example, concerning the NURBS patch (1) shown in **Fig. 8**, the weights, $w_{i,j}$, are rendered as:

$$\begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \end{bmatrix} = \begin{bmatrix} w_{(1)} & w_{(4)} \\ w_{(2)} & w_{(5)} \\ w_{(3)} & w_{(6)} \end{bmatrix} \tag{35}$$

Further, W in Eq. (34) is rendered for the NURBS patch (1) as follows:

$$\begin{aligned} W = {} & \left(N_{1,p}(u)N_{1,q}(v)w_{1,1}\right)+\left(N_{2,p}(u)N_{1,q}(v)w_{2,1}\right)+\left(N_{3,p}(u)N_{1,q}(v)w_{3,1}\right) \\ & +\left(N_{1,p}(u)N_{2,q}(v)w_{1,2}\right)+\left(N_{2,p}(u)N_{2,q}(v)w_{2,2}\right)+\left(N_{3,p}(u)N_{2,q}(v)w_{3,2}\right) \end{aligned} \tag{36}$$

Eq. (36) can be rewritten as:

$$\begin{aligned} W = {} & \left(N_{1,p}(u)w_{1,1}+N_{2,p}(u)w_{2,1}+N_{3,p}(u)w_{3,1}\right)N_{1,q}(v) \\ & +\left(N_{1,p}(u)w_{1,2}+N_{2,p}(u)w_{2,2}+N_{3,p}(u)w_{3,2}\right)N_{2,q}(v) \end{aligned} \tag{37}$$

Based on Eq. (34), the derivatives of NURBS basis function are rendered as:

$$\frac{dR_{i,j}^{p,q}(u,v)}{du} = \frac{W\left(\frac{dN_{i,p}(u)}{du}\right)N_{j,q}(v)-\left(\frac{dW}{du}\right)N_{i,p}(u)N_{j,q}(v)}{W^2}w_{i,j} \tag{38}$$

$$\frac{dR_{i,j}^{p,q}(u,v)}{dv} = \frac{W\,N_{i,q}(u)\left(\frac{dN_{j,p}(v)}{dv}\right)-\left(\frac{dW}{du}\right)N_{i,p}(u)N_{j,q}(v)}{W^2}w_{i,j} \tag{39}$$

The following MATLAB code calculates the Jacobian, J_u, defined in Eq. (24).

```
function[Jac] = NURBS_deriv(knotg,u,v)

%-------------------------------------------------
% Compute the derivatives of NURBS basis function
%
%== Input ==
% knotg ... Information for geometry data
% u, v   ... Parametric values
%
%== Output ==
% Jac ... Jacobian matrix
%-------------------------------------------------

%== Knot vector & polynomial order in u for geometry ==
knotug = knotg{1}; pug = knotg{2};

%== Knot vector & polynomial order in v for geometry ==
knotvg = knotg{3}; pvg = knotg{4};

%== Coefficients ==
coefs = knotg{5};
xg = coefs(1,:)./coefs(4,:);   %=>> x-coordinate
yg = coefs(2,:)./coefs(4,:);   %=>> y-coordinate
wg = coefs(4,:);   %=>> weight

%== Calculate the derivatives of B-spline basis function ==
[Nu,dNu] = Bsp_deriv(knotug,pug,u);
Iu = length(knotug)-pug-1;
%=>> Number of basis functions for u-coordinate

[Nv,dNv] = Bsp_deriv(knotvg,pvg,v);
Iv = length(knotvg)-pvg-1;
%=>> Number of basis functions for v-coordinate

%== Calculate the derivatives of NURBS basis function ==
W = 0;
dWdu = 0;
dWdv = 0;
for I = 1:Iv
   k1 = Iu*(I-1)+1;   k2 = Iu*I;
   W = (Nu'*wg(k1:k2)')*Nv(I) + W;
   dWdu = (dNu'*wg(k1:k2)')*Nv(I) + dWdu;
   dWdv = (Nu'*wg(k1:k2)')*dNv(I) + dWdv;
end;

dRdu1 = W.*dNu'.*wg(1:Iu).*Nv(1);
%=>> Derivative with respect to u(1)
dRdu2 = dWdu.*Nu'.*wg(1:Iu).*Nv(1);
%=>> Derivative with respect to u(2)
```

```
dRdv1 = W.*Nu'.*wg(1:lu).*dNv(1);
%=>> Derivative with respect to v(1)
dRdv2 = dWdv.*Nu'.*wg(1:lu).*Nv(1);
%=>>   Derivative with respect to v(2)
if lv > 1
  for l = 2:lv
    k1 = lu*(l-1)+1; k2 = lu*l;
    dRdu1 = [dRdu1, W.*dNu'.*wg(k1:k2).*Nv(l)];
    dRdu2 = [dRdu2, dWdu*Nu'.*wg(k1:k2).*Nv(l)];

    dRdv1 = [dRdv1, W.*Nu'.*wg(k1:k2).*dNv(l)];
    dRdv2 = [dRdv2, dWdv.*Nu'.*wg(k1:k2).*Nv(l)];
  end;
end;

dRdu  = (dRdu1-dRdu2)/W^2;  %=>> Derivative with u
dRdv  = (dRdv1-dRdv2)/W^2;  %=>> Derivative with v

dxdu = dRdu*xg';
dydu = dRdu*yg';

dxdv = dRdv*xg';
dydv = dRdv*yg';

Jac = [dxdu, dydu ;   %=>> Jacobian matrix
       dxdv, dydv ];
```

Concerning the unknown field variables, let us consider the following knot vector for each NURBS patch as shown in **Fig. 10**:

$$\begin{aligned} \Xi_{\mathrm{u}} &= [0,0,0,1,1,1], \quad p=2 \\ \Xi_{\mathrm{v}} &= [0,0,0,1,1,1], \quad p=2 \end{aligned} \tag{40}$$

For the assembly process to construct the system equation, we need to define the connectivity between the patches. Thus, we define the following incidence vectors:

$$\begin{aligned} inci(1) &= [1,2,3,4,5,6,7,8,9] \\ inci(2) &= [3,10,11,6,12,13,9,14,15] \end{aligned} \tag{41}$$

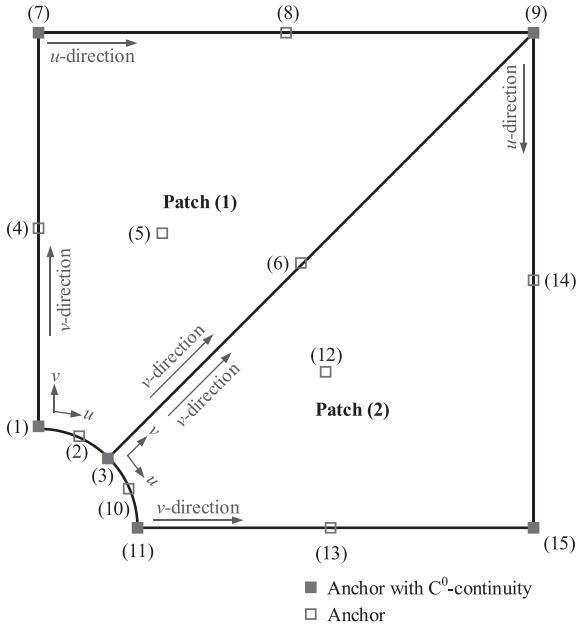


Fig. 10. Anchor points for field variables.

Let us consider the force vector, **F**, associated with the applied traction, as shown in **Fig. 7**. Traction vector for patch (1) is rendered as (**Fig. 10**):

$$t_i = \sigma_{ji} n_j \qquad \text{or} \qquad \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} \sigma_{xx} & \sigma_{yx} & \sigma_{zx} \\ \sigma_{xy} & \sigma_{yy} & \sigma_{zy} \\ \sigma_{xz} & \sigma_{yz} & \sigma_{zz} \end{bmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} \tag{42}$$

where $\sigma_{ji}\,(=\sigma_{ij})$ is the stress tensor and n_j is the unit outward normal vector. For instance, concerning the patch (1), Eq. (42) render the following equation:

$$\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{xy} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{xz} & \sigma_{yz} & \sigma_{zz} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \sigma_{xy} \\ \sigma_{yy} \\ 0 \end{bmatrix} \tag{43}$$

Then, the external virtual work, W, is calculated as:

$$\begin{aligned} W &= t\int (u_x t_x + u_y t_y)\, d\Gamma\Big|_{y=h} \\ &= t\int (u_x \sigma_{xy} + u_y \sigma_{yy})\, d\Gamma\Big|_{y=h} \\ &= t\int_{Patch(1)} \{N_7 u_{7x} + N_8 u_{8x} + N_9 u_{9x}\}\, \sigma_{xy} \bar{J}\, du\Big|_{v=1} \\ &\quad + t\int_{Patch(1)} \{N_7 u_{7x} + N_8 u_{8x} + N_9 u_{9x}\}\, \sigma_{yy} \bar{J}\, du\Big|_{v=1} \end{aligned} \tag{44}$$

where:

$$\bar{J} = \sqrt{\left(\frac{dx}{du}\right)^2 + \left(\frac{dy}{du}\right)^2} \tag{45}$$

To make use of the Gauss integration, we evaluate Eq. (45)

with the ξ-coordinate as follows:

$$W = \left[u_{7x}, u_{8x}, u_{9x}\right]\cdot t\int_{-1}^{1}\begin{bmatrix}N_7(\xi)\\ N_8(\xi)\\ N_9(\xi)\end{bmatrix}\sigma_{xy}(\xi)\bar{J}(\xi)\left(\frac{du(\xi)}{d\xi}\right)d\xi\Big|_{v=1} + \left[u_{7y}, u_{8y}, u_{9y}\right]\cdot t\int_{-1}^{1}\begin{bmatrix}N_7(\xi)\\ N_8(\xi)\\ N_9(\xi)\end{bmatrix}\sigma_{yy}(\xi)\bar{J}(\xi)\left(\frac{du(\xi)}{d\xi}\right)d\xi\Big|_{v=1}$$

$$\cong \left[u_{7x}, u_{8x}, u_{9x}\right]\cdot t\sum_{g=1}^{NG}\begin{bmatrix}N_7(\xi_g)\\ N_8(\xi_g)\\ N_9(\xi_g)\end{bmatrix}\sigma_{xy}(\xi_g)\bar{J}(\xi_g)\left(\frac{du(\xi_g)}{d\xi}\right)W_g\Big|_{v=1} + \left[u_{7y}, u_{8y}, u_{9y}\right]\cdot t\sum_{g=1}^{NG}\begin{bmatrix}N_7(\xi_g)\\ N_8(\xi_g)\\ N_9(\xi_g)\end{bmatrix}\sigma_{yy}(\xi_g)\bar{J}(\xi_g)\left(\frac{du(\xi_g)}{d\xi}\right)W_g\Big|_{v=1} \tag{46}$$

Finally, based on the theorem of minimum potential energy, the force vector is rendered as:

$$F_i = \frac{\partial W}{\partial u_i} \rightarrow \begin{bmatrix}F_{7x}\\ F_{8x}\\ F_{9x}\end{bmatrix} = t\sum_{g=1}^{NG}\begin{bmatrix}N_7(\xi_g)\\ N_8(\xi_g)\\ N_9(\xi_g)\end{bmatrix}\sigma_{xy}(\xi_g)\bar{J}(\xi_g)\left(\frac{du(\xi_g)}{d\xi}\right)W_g\Big|_{v=1} \tag{47}$$

$$F_i = \frac{\partial W}{\partial u_i} \rightarrow \begin{bmatrix}F_{7y}\\ F_{8y}\\ F_{9y}\end{bmatrix} = t\sum_{g=1}^{NG}\begin{bmatrix}N_7(\xi_g)\\ N_8(\xi_g)\\ N_9(\xi_g)\end{bmatrix}\sigma_{yy}(\xi_g)\bar{J}(\xi_g)\left(\frac{du(\xi_g)}{d\xi}\right)W_g\Big|_{v=1} \tag{48}$$

The following program is for providing the input data in

```
function[Binfo,knotg,knot,t,Dmat,Sbar] = Input_data_Plate_1a()

%-------------------------------------------------------------
% Set up the information for boundary conditions etc

%== Output ==
% Binfo = {Bcode,Bvalue,Dest0,Dest1,Ncp0,Ncp1,Ncp}
%     Bcode    ... Boundary code (0:Newmann BC; 1:Drichlet BC)
%     Bvalue ... Boundary value (displacemnt and force at each
control point)
%     Dest0    ... Destination vector for Newmann BC (Bcode = 0)
%     Dest1    ... Destination vector for Dirichlet BC (Bcode = 1)
%     Ncp0     ... Number of Newmann BCs (x & y)
%     Ncp1     ... Number of Dirichlet BCs (x & y)
%     Ncp       ... Number of control points

% knotg    ... Geometry data
% knot     ... Knot vector for field variables
% t          ... Thickness
% Dmat     ... D-matrix
%-------------------------------------------------------------

%== Compute D-matrix ==
E   = 206*10^3;   %=>> Young's modulus (MPa)
nu = 0.3;           %=>> Poisson's ratio
t    = 1;           %=>> Plate thickness
Dmat = E/(1-nu^2)*[ 1, nu, 0 ;
                    nu,   1, 0 ;
                     0,   0, (1-nu)/2];

%== Geometry data ==
% Homogenized coordinates (coordinates are multiplied by weight)
% Cp = [x; y; z; 1]*weight
H = 50; L = 50; R = 10;
Cp = zeros(4,10);
Cp(:,1)  = [0; R; 0; 1];                    %=>> Control point (1)
Cp(:,2)  = [0.4142*R; R; 0; 1]*0.9239; %=>> Control point (2)
Cp(:,3)  = [0.707*R; 0.7071*R; 0; 1];   %=>> Control point (3)
Cp(:,4)  = [0; H; 0; 1];                    %=>> Control point (4)
Cp(:,5)  = [L/2; H; 0; 1];                  %=>> Control point (5)
Cp(:,6)  = [L; H; 0; 1];                    %=>> Control point (6)
Cp(:,7)  = [R; 0.4142*R; 0; 1]*0.9239; %=>> Control point (7)
Cp(:,8)  = [R; 0; 0; 1];                    %=>> Control point (8)
Cp(:,9)  = [L; H/2; 0; 1];                  %=>> Control point (9)
Cp(:,10) = [L; 0; 0; 1];                    %=>> Control point (10)

%== NURBS patch ==
coefs1 = [Cp(:,1),Cp(:,2),Cp(:,3), Cp(:,4),Cp(:,5),Cp(:,6)] ; %=>>
Patch (1)
coefs2 = [Cp(:,3),Cp(:,7),Cp(:,8),Cp(:,6),Cp(:,9),Cp(:,10)];
%=>> Patch (2)

knotug = [0,0,0, 1,1,1]; pug = 2;
%=>> Knot vector and polynomial order in u
knotvg = [0,0,1,1]; pvg = 1;
%=>> Knot vector and polyoimial order in v

%== Create a strucuture for each NURBS patch ==
knotg1 = {knotug,pug, knotvg,pvg, coefs1};   %=>> Patch(1)
knotg2 = {knotug,pug, knotvg,pvg, coefs2};   %=>> Patch(2)
knotg   = {knotg1, knotg2};


%-------------------------------------------------------------
%== Knot vectors for field variables ==
knotu = [0,0,0, 1,1,1]; pu = 2;
knotv = [0,0,0, 1,1,1]; pv = 2;
```

```
Inci1 = [1,2,3, 4,5,6, 7,8,9];
Inci2 = [3,10,11, 6,12,13, 9,14,15];
Ncp = max([Inci1,Inci2]);
%=>> Maximum incidence number = Number of control points

%== Create structures for each NURBS patch ==
knot1   = {knotu,pu, knotv,pv, Inci1 };   %=>> Patch(1)
knot2   = {knotu,pu, knotv,pv, Inci2 };   %=>> Patch(2)
knot    = {knot1, knot2};

%== Boundary conditions ==
Bcode   = zeros(2*Ncp,1);
Bvalue = zeros(2*Ncp,1);

%== Compute the force vector at the boundaries ==
Binci = [7,8,9; 9,14,15]; %=>> Boundary incidence vector
Sbar = 100; %=>> 100MPa
[Bvalue] = Traction_BC(Binci,Bvalue,knotg,knot,t,Sbar,R);

%== Prescribed displacelement ==
Bcode(2*11) = 1;   Bvalue(2*11) = 0; %=>> Dirichlet BC(1)
Bcode(2*13) = 1;   Bvalue(2*13) = 0; %=>> Disp. in y is fixed
Bcode(2*15) = 1;   Bvalue(2*15) = 0;

Bcode(2*1-1) = 1; Bvalue(2*1-1) = 0; %=>> Dirichlet BC(2)
Bcode(2*4-1) = 1; Bvalue(2*4-1) = 0; %=>> Disp. in x is fixed
Bcode(2*7-1) = 1; Bvalue(2*7-1) = 0;

%== Set up the destination vectors ==
Ncp1 = sum(Bcode);          %=>> Number of Dirichlet BCs (x & y)
Ncp0 = 2*Ncp - Ncp1;      %=>> Number of Newmann BCs (x & y)
Dest1 = zeros(1,Ncp1);    %=>> Destination vector for Dirichlet
BCs
Dest0 = zeros(1,Ncp0);   %=>> Destination vector for Newmann
BCs
J = 1; K = 1;
for I = 1:Ncp
  if Bcode(2*I-1) == 1   %=>> Check BCs in x-direction
    Dest1(J) = 2*I-1; J = J + 1;
  else
    Dest0(K) = 2*I-1; K = K + 1;
  end;

  if Bcode(2*I) == 1   %=>> Check BCs in y-direction
    Dest1(J) = 2*I; J = J + 1;
  else
    Dest0(K) = 2*I; K = K + 1;
  end;
end;

%== Create a structure for convenience ==
Binfo = {Bcode,Bvalue,Dest0,Dest1,Ncp0,Ncp1,Ncp};
```

The following program is to compute the force vector associated with applied traction, as shown in **Figs. 7** and **9**.

```
function[Bvalue] = traction_BC(Binci,Bvalue,knotg,knot,t,Sbar,R)

%---------------------------------------------------------------
% Compute the force vector associated with distributed traction

%== Input ==
% Binci   ... Boundary incidence vector for prescribed traction
% Bvalue ... Boundary value
% knotg   ... Information for geometry (knot vector etc)
% knot    ... Approximation for field variable (knot vector etc)
% t         ... Plate thickness
% Sbar    ... Prescribed stress (uniform far-field stress)
% R         ... Hole radius

%== Output ==
% Bvalue ... Force vector with traction is added to original Bvalue
%---------------------------------------------------------------

%== 4-point Gauss integration ==
Gpt = [-0.861136, -0.339981, 0.339981, 0.861136];
%=>> Sampling point
Gwt = [ 0.347855,   0.652145, 0.652145, 0.347855];
%=>> Weight

%=========================================
%== Consider the subregions for u-coordinate, patch(1) ==
%=========================================
knot1 = knot{1};
knotu = knot1{1}; pu = knot1{2};
usub   = [0, knotu(pu+2)];
for I = (pu+3):length(knotu)
  if knotu(I) > max(usub)
    usub = [usub, knotu(I)];
  end;
end;
Nsub = length(usub)-1;   %=>> Number of subregions

%== Geometry data for patch(1) ==
```

```
knotg1 = knotg{1};
knotug = knotg1{1}; pug = knotg1{2};
knotvg = knotg1{3}; pvg = knotg1{4};
coefs  = knotg1{5};
xg = coefs(1,:)./coefs(4,:); xg = xg([4,5,6]);
yg = coefs(2,:)./coefs(4,:); yg = yg([4,5,6]);


%== Compute the force vector ==
Fx = zeros(length(Binci(1,:)),1);
Fy = zeros(length(Binci(1,:)),1);
for I = 1:Nsub   %=>> Subregion loop
  for J = 1:4    %=>> Gauss-integration loop

    xsi = Gpt(J); Wxsi = Gwt(J);   %=>> Gauss point & weight
    u = (1-xsi)/2*usub(I) + (1+xsi)/2*usub(I+1);
    %=>> Transformation to u
    v = 1;
    Nu = Bsp_deriv(knotug,pug,u);
    x = Nu'*xg';   y = yg(1);   %=>> Compute x & y coordinate
    r = sqrt(x^2+y^2); th = atan(y/x);   %=>> Polar coordinate

    %== Compute the normal stress in y-direction
    Sy = Sbar/2*(2+3*R^2/r^2*cos(2*th)-(2*R^2/r^2-3*R^4/r^4)*cos(4*th));
    Sxy = Sbar/2*(-R^2/r^2+(4*R^2/r^2-6*R^4/r^4)*cos(2*th))*sin(2*th);

    [Jac] = NURBS_deriv(knotg1,u,v);
    Jbar  = sqrt( Jac(1,1)^2+Jac(1,2)^2 );
    dudxsi = ( usub(I+1) - usub(I) )/2;
    Weight = Wxsi*Jbar*dudxsi;
    [Nu] = Bsp_deriv(knotu,pu,u);

    Fx     = t*Sxy*Nu.*Weight + Fx;
    Fy     = t*Sy*Nu.*Weight  + Fy;

  end;
end;

%== Update the force vector (1) ==
Bvalue(2*Binci(1,:)-1,1) = Fx + Bvalue(2*Binci(1,:)-1,1);
Bvalue(2*Binci(1,:)  ,1) = Fy + Bvalue(2*Binci(1,:)  ,1);


%=======================================
%== Consider the subregions for u-coordinate, patch(2) ==
%=======================================
knot2 = knot{2};
knotu = knot2{1}; pu = knot2{2};
usub  = [0, knotu(pu+2)];
for I = (pu+3):length(knotu)
  if knotu(I) > max(usub)
    usub = [usub, knotu(I)];
  end;
end;
Nsub = length(usub)-1;   %=>> Number of subregions

%== Geometry data for patch(2) ==
knotg2 = knotg{2};
knotug = knotg2{1}; pug = knotg2{2};
knotvg = knotg2{3}; pvg = knotg2{4};
coefs  = knotg2{5};
xg = coefs(1,:)./coefs(4,:); xg = xg([4,5,6]);
yg = coefs(2,:)./coefs(4,:); yg = yg([4,5,6]);


%== Compute the force vector ==
Fx = zeros(length(Binci(2,:)),1);
Fy = zeros(length(Binci(2,:)),1);
for I = 1:Nsub   %=>> Subregion loop
  for J = 1:4    %=>> Gauss-integration loop

    xsi = Gpt(J); Wxsi = Gwt(J);   %=>> Gauss point & weight
    u = (1-xsi)/2*usub(I) + (1+xsi)/2*usub(I+1);
    %=>> Transformation to u
    v = 1;
    Nu = Bsp_deriv(knotug,pug,u);
    x = xg(1);   y = Nu'*yg';   %=>> Compute x & y coordinate
    r = sqrt(x^2+y^2); th = atan(y/x);   %=>> Polar coordinate

    %== Compute the normal stress in x-direction

Sx=Sbar/2*(R^2/r^2*cos(2*th)+(2*R^2/r^2-3*R^4/r^4)*cos(4*th) );
Sxy=Sbar/2*(-R^2/r^2+(4*R^2/r^2-6*R^4/r^4)*cos(2*th))*sin(2*th);

    [Jac] = NURBS_deriv(knotg2,u,v);
    Jbar  = sqrt( Jac(1,1)^2+Jac(1,2)^2 );
    dudxsi = ( usub(I+1) - usub(I) )/2;
    Weight = Wxsi*Jbar*dudxsi;
    [Nu] = Bsp_deriv(knotu,pu,u);
    Fx     = t*Sx*Nu.*Weight  + Fx;
    Fy     = t*Sxy*Nu.*Weight + Fy;

  end;
end;

%== Update the force vector (2) ==
Bvalue(2*Binci(2,:)-1,1) = Fx + Bvalue(2*Binci(2,:)-1,1);
Bvalue(2*Binci(2,:)  ,1) = Fy + Bvalue(2*Binci(2,:)  ,1);
```

The following program is for computing the stiffness matrix for each NURBS patch:

```
function[Kij] = Stiffness_matrix_patch(knotg,knot,NcpG,t,Dmat,Ng)

%-------------------------------------------------------------
% Compute the stiffness matrix for NURBS patch

%== Input ==
% knotg ... Information for geometry data
% knot   ... Knot vector and polynomial order for field variables
% t      ... Thickness
% Dmat   ... D-matrix (stiffness matrix)
% Ng     ... Number of Gauss-points (sampling points)
% NcpG   ... Number of anchors in global

%== Output ==
% Kij    ... Stiffness matrix
%-------------------------------------------------------------

%== Gauss point and weight ==
Gpt = [          0,              0,          0,          0;
        -0.577350,   0.577350,          0,          0;
        -0.774967,              0, 0.774967,          0;
        -0.861136, -0.339981, 0.339981, 0.861136];
%=>> Gauss point (sampling point)

Gwt = [2.000000,   0.000000, 0.000000, 0.000000 ;
        1.000000,   1.000000, 0.000000, 0.000000 ;
        0.555556,   0.888889, 0.555556, 0.000000 ;
        0.347855,   0.652145, 0.652145, 0.347855];
%=>> Weight

%== Knot vector & polynomial order in u for geometry ==
knotug = knotg{1}; pug = knotg{2};

%== Knot vector & polynomial order in v for geometry ==
knotvg = knotg{3}; pvg = knotg{4};

%== Control points for geometry ==
% coefs = knotg{5};
% xg = coefs(1,:)./coefs(4,:);
% yg = coefs(2,:)./coefs(4,:);
% wg = coefs(4,:);

%== Knot vector & polynomial order in u for field variables ==
knotu = knot{1}; pu = knot{2};

%== Knot vector & polynomial order in v for field variables ==
knotv = knot{3}; pv = knot{4};

%== Incidence vector ==
Inci = knot{5};

%== Set up for sub-regions ==
usub = [0,knotu(pu+2)];   %=>> Set up for sub-regions (1)
for I = (pu+3):length(knotu)
  if knotu(I) > max(usub)
    usub = [usub, knotu(I)];
  end;
end;

vsub = [0,knotv(pv+2)];   %=>> Set up for sub-regions (2)
for I = (pv+3):length(knotv)
  if knotv(I) > max(vsub)
    vsub = [vsub, knotv(I)];
  end;
end;

Nsub = (length(usub)-1)*(length(vsub)-1);   %=>> Number of
sub-regions
nu   = length(knotu)-pu-1;   %=>> Number of control points in u
nv   = length(knotv)-pv-1;   %=>> Number of control points in v
Ncp = nu*nv;   %=>> Number of control points


%== Control points in sub-regions ==
us = zeros(1,Nsub);   ue = zeros(1,Nsub);
vs = zeros(1,Nsub);   ve = zeros(1,Nsub);

K = 1;
for J = 1:(length(vsub)-1)
  for I = 1:(length(usub)-1)
    us(1,K) = usub(I);      %=>> Start point in u for a sub-region
    ue(1,K) = usub(I+1);   %=>> End point in u
    vs(1,K) = vsub(J);      %=>> Start point in v
    ve(1,K) = vsub(J+1);   %=>> End point in v
    K = K+1;
  end;
end;


Kij = zeros(2*NcpG, 2*NcpG);   %=>> Stiffness matrix
for I = 1:Nsub   %=>> Sub-region loop;

  for J1 = 1:Ng   %=>> Gauss-integration loop (1)
```

```
  for J2 = 1:Ng   %=>> Gauss-integration loop (2)
    xsi = Gpt(Ng,J1); W1 = Gwt(Ng,J1);
    %=>> Gauss point & weight in xsi
    eta = Gpt(Ng,J2); W2 = Gwt(Ng,J2);
    %=>> Gauss point & weight in eta

    u = (1-xsi)/2*us(l) + (1+xsi)/2*ue(l);
    %=>> Transformation to u
    v = (1-eta)/2*vs(l) + (1+eta)/2*ve(l);
    %=>> Transformation to v

    %== Let us consider the basis functions for geometry ==
    Jac = NURBS_deriv(knotg,u,v);

    Ju   = Jac(1,1)*Jac(2,2) - Jac(1,2)*Jac(2,1);
    Jxsi = (ue(l)-us(l))*(ve(l)-vs(l))/4;

    %== Let us consider the basis functions for displacement ==
    [Nu, dN1u] = Bsp_deriv(knotu,pu,u);
    [Nv, dN1v] = Bsp_deriv(knotv,pv,v);

    %== Compute the 1st derivatives at each control point ==
    dNdu = dN1u'.*Nv(1);
    dNdv = Nu'.*dN1v(1);
    if nv > 1
      for K = 2:nv
        dNdu = [dNdu, dN1u'.*Nv(K)];
        dNdv = [dNdv, Nu'.*dN1v(K)];
      end;
    end;

    %== Overall weight at a Gauss-point in a sub-region ==
    Weight = W1*W2*Ju*Jxsi;

    Bmat = zeros(3,2*NcpG);   %=> B-matrix at each control
point
    for K = 1:Ncp
      Drv = Jac¥[dNdu(K); dNdv(K)];
      dNdx = Drv(1);
      dNdy = Drv(2);
      Km = Inci(K);   %=>> Change it to the global number
      Bmat(:,2*Km-1:2*Km) = [dNdx,    0 ;   %=>> B-matrix
                             0   , dNdy;
                            dNdy, dNdx] ;
    end;

    %== Compute the stiffness matrix with Gauss-integration ==
    Kij = (Bmat'*Dmat*Bmat)*Weight*t + Kij;

  end;   %=>> End of Gauss-integration (2)

  end;   %=>> End of Gauss-integration (1)

end;   %=>> End of sub-region-loop
```

The following is the main program.

```
%************************************************************
%** Solve the two-dimensional elastostatic problem with IGA **
%************************************************************
%== Obtain the input data ==
[Binfo,knotg,knot,t,Dmat,Sbar] = Input_data_Plate_1a();

%== Binfo:{Bcode,Bvalue,Dest0,Dest1,Ncp0,Ncp1,Ncp} ==
Bcode   = Binfo{1}; Bvalue = Binfo{2};
Dest0   = Binfo{3}; Dest1   = Binfo{4};
Ncp0    = Binfo{5}; Ncp1    = Binfo{6}; NcpG   = Binfo{7};

%== Compute the stiffness matrix ==
Ng = 4;   %=>> Number of Gauss-point
[Kij1] = Stiffness_matrix_patch(knotg{1},knot{1},NcpG,t,Dmat,Ng);
[Kij2] = Stiffness_matrix_patch(knotg{2},knot{2},NcpG,t,Dmat,Ng);
Kij = Kij1 + Kij2;

%== Compute the modified stiffness matrix ==
Kmod = Kij(Dest0, Dest0);
%== Compute the modified force vector ==
Fmod = zeros(Ncp0,1);
for I = 1:Ncp0
  K = Dest0(I);
  Fmod(I) = Bvalue(K);

  for J = 1:Ncp1
    L = Dest1(J);   %=>> Displacement is prescribed
    Fmod(I) = Fmod(I) - Kij(K,L)*Bvalue(L);
  end;
end;

%== Obtain the solutions ==
umod = Kmod¥Fmod;
usol = zeros(2*NcpG,1);   %=>> Solution of displacement vector
K = 1;
for I = 1:2*NcpG
  if Bcode(I) == 0   %=>> Displacement is originally unknown
    usol(I) = umod(K); K = K + 1;
  else
    usol(I) = Bvalue(I);
  end;
end;
```

```
Fsol = Kij*usol;   %=>> Solution for force vector

%== Compute the stress distribution ==
[Sigma,xcoor] = Comp_Stress(knotg{2},knot{2},Dmat,Sbar,usol);
```

conjunction with **Figs. 8** and **10**.

The machine and structural components have various geometrical discontinuities (e.g., hole, keyway and groove) that trigger the stress concentration and they can serve as the crack initiation sites [7][8][9]. The crack initiation and growth is a major concern for the reliability of the machine and its structural components. Therefore, understanding and identifying stress concentration sites is essential for engineers [10][11]. Correspondingly, the following program is prepared to compute the stress distribution at $y = 0$. Accordingly, only the NURBS patch (2) is considered for

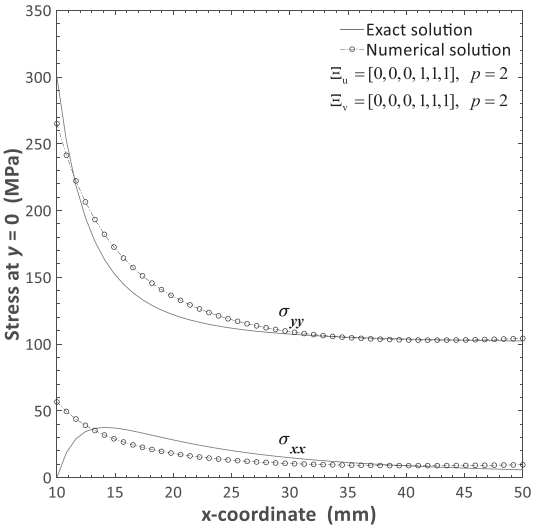
```
function[Sigma,xcoor] = Comp_Stress(knotg, knot, Dmat, Sbar, usol)

%-----------------------------------------------
% Compute the stress distribution

%== Input ==
% knotg   ... Geometrical data for patch(2)
% knotu   ... Approximation for displacement for patch(2)
% Dmat    ... Stiffness matrix
% usol    ... Displacement

%== Output ==
% Sigma ... Stress field at y = 0
%-----------------------------------------------
%== Geometrical data ==
knotug = knotg{1}; pug = knotg{2};
knotvg = knotg{3}; pvg = knotg{4};
coefs  = knotg{5};   %=>> Coefficient for control points

R = coefs(1,3)./coefs(4,3);   %=>> Hole radius

knotu = knot{1}; pu = knot{2};
knotv = knot{3}; pv = knot{4};
inci  = knot{5};   %=>> Incidence vector
xg = coefs(1,:)./coefs(4,:);   %=>> x-coordinate
yg = coefs(2,:)./coefs(4,:);   %=>> y-coordinate

u = 1.0;
v = linspace(0,1,50);
Sigma = zeros(3,length(v));
Sy = zeros(1,length(v));
Sx = zeros(1,length(v));
xcoor = zeros(1,length(v));

for l = 1:length(v)

  x = (1-v(l))*xg(3) + v(l)*xg(6);
  r = x;
  th = 0;
  %== Analytical solution for infinite plate ==
  Rr = R/r;
  Sy(l) = Sbar/2*( 2+3*Rr^2*cos(2*th) - (2*Rr^2-3*Rr^4)*cos(4*th) );
  Sx(l) = Sbar/2*( Rr^2*cos(2*th) + (2*Rr^2-3*Rr^4)*cos(4*th) );
  xcoor(l) = x;

  %== Numerical solution ==
  Jac = NURBS_deriv(knotg,u,v(l));
  %==> Consideration for geometry

  dxdu = Jac(1,1); dydu = Jac(1,2);
  dxdv = Jac(2,1); dydv = Jac(2,2);

  [Nu,dNu] = Bsp_deriv(knotu,pu,u);
  [Nv,dNv] = Bsp_deriv(knotv,pv,v(l));
  dNdu = dNu'.*Nv(1);
  dNdv = Nu'.*dNv(1);
  nv = length(knotv)-pv-1;
  %=>> Number of basis functions for v-direction
  if nv > 1
    for K = 2:nv
      dNdu = [dNdu, dNu'.*Nv(K)];
      dNdv = [dNdv, Nu'.*dNv(K)];
    end;
  end;

  Ncp = length(inci);   %=>> Number of control points in patch(2)
  Bmat = zeros(3,2*Ncp);
  umod = zeros(2*Ncp,1);

  for K = 1:Ncp   %=>> Compute B-matrix
    Drv = Jac¥[dNdu(K); dNdv(K)];
    dNdx = Drv(1);
    dNdy = Drv(2);
    Bmat(:,2*K-1:2*K) = [dNdx,    0   ;
                           0 , dNdy ;
                         dNdy, dNdx ];
    umod(2*K-1) = usol(2*inci(K)-1);   %=>> ux
```

```
    umod(2*K   ) = usol(2*inci(K)   );   %=>> uy
  end;   %=>>

  Sigma(:,l) = Dmat*Bmat*umod;   %=>> (3*1) matrix
end;   %=>> End of v-loop


%== Plot the results ==
FS = 'Fontsize'; FW = 'Fontweight'; FN = 'Fontname';
plot(xcoor,Sy, 'r-', 'Linewidth',1); hold on;
plot(xcoor,Sigma(2,:), 'ko-.', 'Markersize',6);
plot(xcoor,Sx, 'r-', 'Linewidth',1);
plot(xcoor,Sigma(1,:), 'ko-.'); hold off;

axis square;
axis([R,50,0,350]);

leg = legend('Exact solution', 'Numerical solution');
legend('boxoff')
set(gca,'Fontsize',18,FN,'calibri','Xminortick','on','Yminortick','on');
set(leg,'Fontsize',18);

xlabel('x-coordinate   (mm)',        FS,28, FN,'calibri', FW,'bold');
ylabel('Stress at y = 0   (MPa)', FS,28, FN,'calibri', FW,'bold');

text(29,123, '¥sigma_{yy}', FS,22, FW,'bold');
text(29, 27, '¥sigma_{xx}', FS,22, FW,'bold');
```

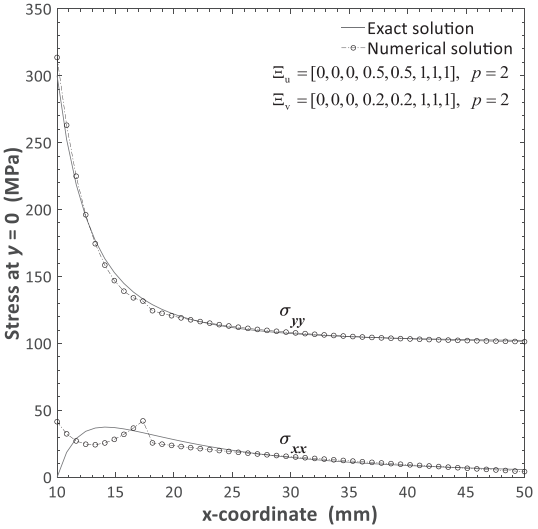
calculation (cf. **Figs. 7**, **8** and **9**).

For this problem (cf. **Fig. 7**), the stress concentration factor is 3. The analysis was performed with $\bar{\sigma} = 100$ MPa , thus the maximum stress of σ_{yy} should be rendered as = $\sigma_{yy\_\max}$ = 3×100 MPa = 300 MPa at the edge of hole. It is noted that the numerical integration for obtaining the stiffness matrix was performed with 4×4 integration points for each sub-region. **Figs. 11** and **12** show the stress distribution at y = 0. As shown, by using the refinement strategies, the numerical solution converges to the exact solution. As expected, *k*-refinement shows a superior result over *h*-refinement in terms of accuracy (cf. **Fig. 11**(c) and **Fig. 12**(c)).

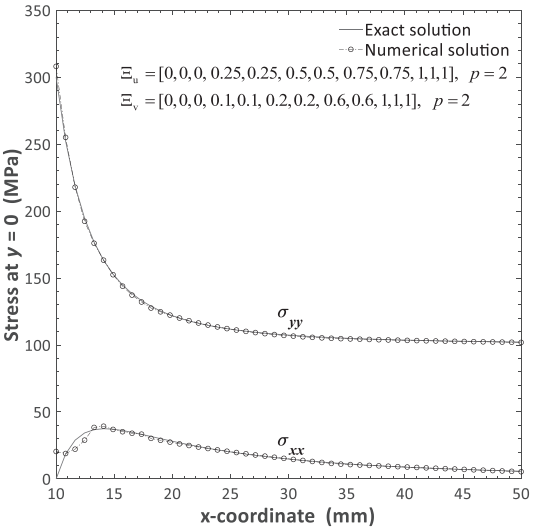
Figs. 13 and **14** show the effect of Gauss quadrature rule on the numerical solution. As shown, when the number of Gauss points is insufficient, the numerical solution has a large oscillation that is not observed for the exact solution. In these simulations, 3×3 Gauss quadrature rule can provide a good result, irrespective of polynomial order, p.



(a) Number of control points = 9 in a NURBS patch

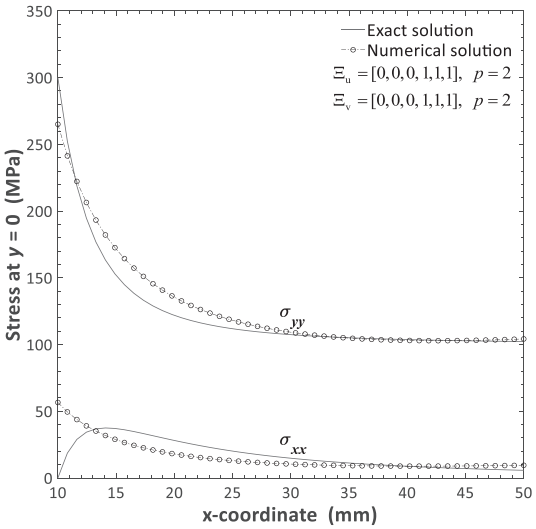


(b) Number of control points = 25 in a NURBS patch

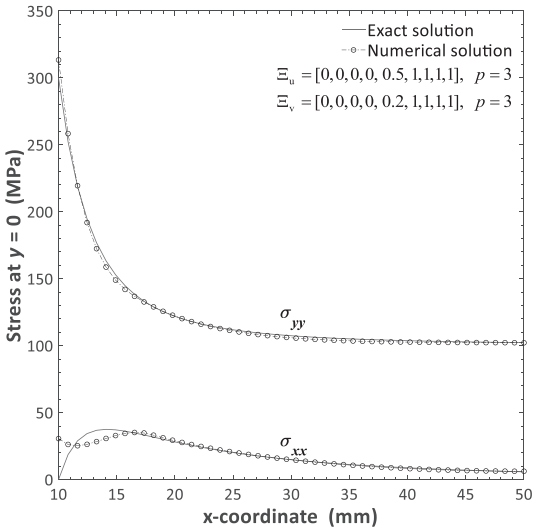


(c) Number of control points = 81 in a NURBS patch

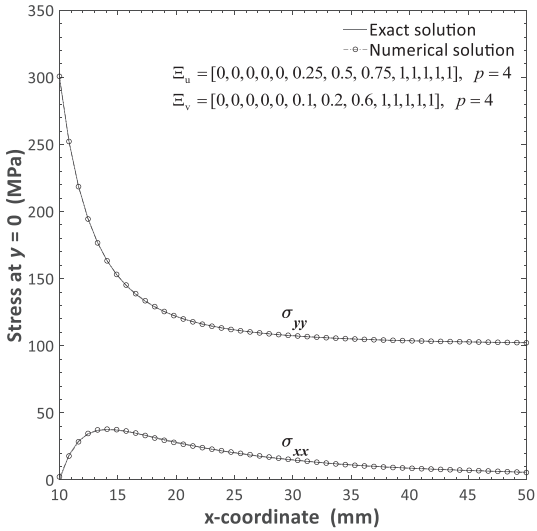
Fig. 11. *h*-refinement.



(a) Number of control points = 9 in a NURBS patch

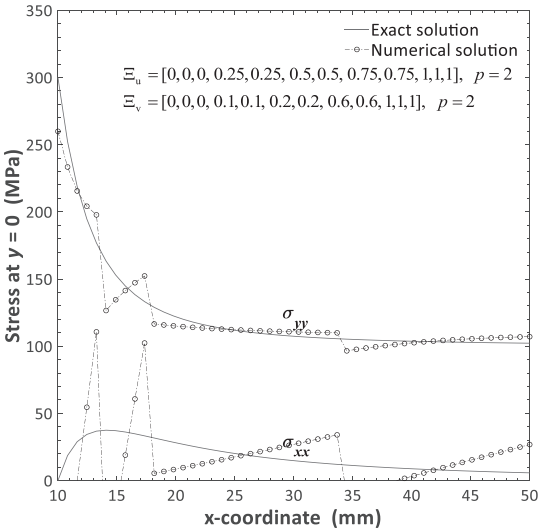


(b) Number of control points = 25 in a NURBS patch

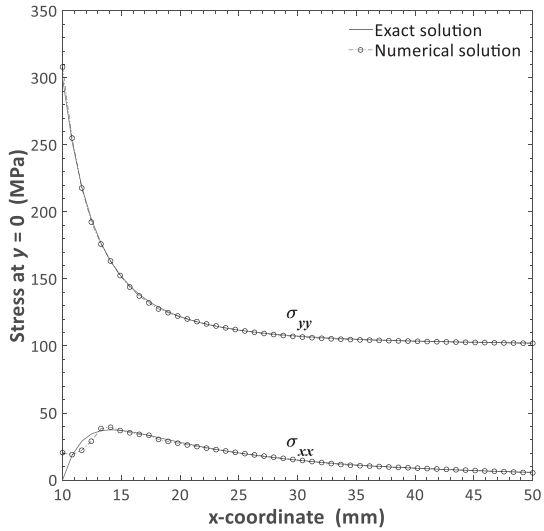


(c) Number of control points = 64 in a NURBS patch

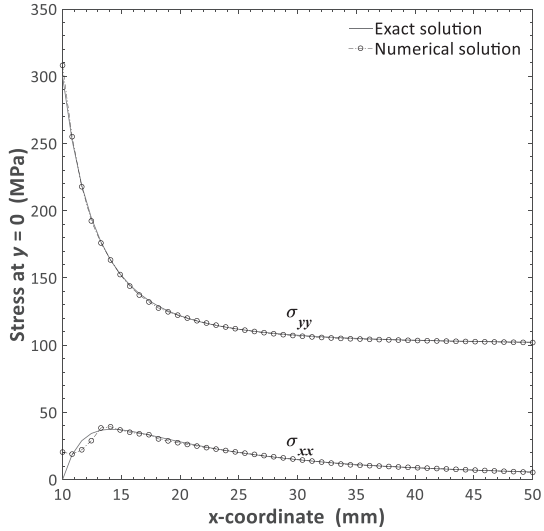
Fig. 12. *k*-refinement.



(a) 2×2 Gauss points

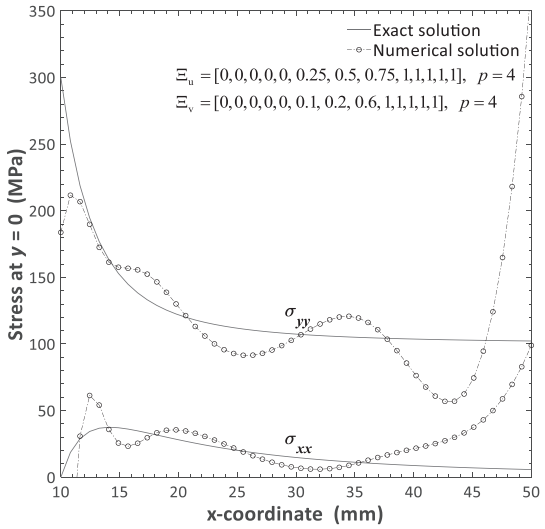


(b) 3×3 Gauss points

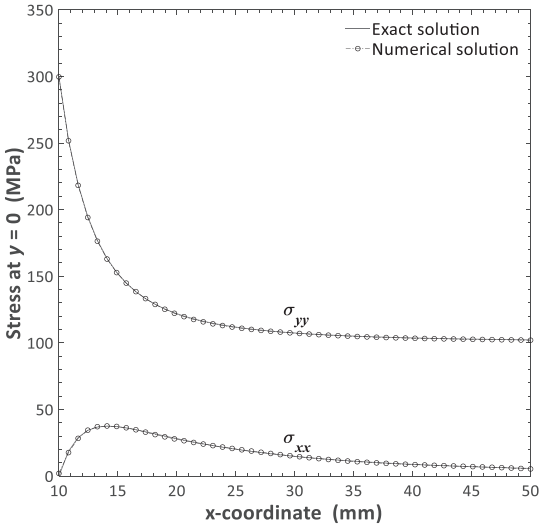


(c) 4×4 Gauss points

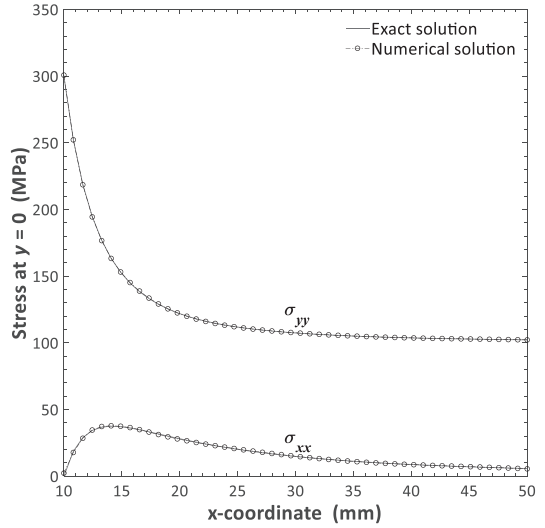
Fig. 13. Effect of Gauss quadrature in Fig. 11(c).



(a) 2×2 Gauss points



(b) 3×3 Gauss points



(c) 4×4 Gauss points

Fig. 14. Effect of Gauss quadrature in Fig. 12(c).

Conclusions

In this study, two-dimensional elastic problems are tackled with the NURBS-based isogeometric analysis. It is noted that the knowledge in the finite element analysis is equally applicable to the isogeometric analysis in writing the computer code. Several demonstrations show the superiorities and the interesting properties of the method.

References

[1] K. Yanase, A gentle introduction to isogeometric analysis: Part 1 B-spline curve & 1D finite element analysis, *Fukuoka University Review of Technological Sciences*, Vol. 98, pp.1-9, 2017.

[2] K. Yanase, A gentle introduction to isogeometric analysis: Part 2 NURBS curve and surface, *Fukuoka University Review of Technological Sciences*, Vol. 99, pp.1-11, 2017.

[3] K. Yanase, A gentle introduction to isogeometric analysis: Part 3 Elastostatic analysis for Euler-Bernoulli beam, *Fukuoka University Review of Technological Sciences*, Vol. 100, pp.1-10, 2018.

[4] G. Beer, *Advanced numerical simulation methods*, CRC Press, 2015.

[5] J.A. Cottrell, T.J.R. Hughes and Y. Bazilevs, *Isogeometric analysis: Toward integration of CAD and FEA*, John Wiley & Sons, 2009.

[6] L. Piegl and W. Tiller, *The NURBS book* (2nd edition), Springer, 1997.

[7] K. Yanase, K. Shojima and C. Ogata, High-cycle fatigue threshold behaviors in notched plate, *International Journal of Fatigue*, Vol. 22(7), pp. 1006-1022, 2013.

[8] B.M. Schönbuer, K. Yanase and M. Endo, Influences of small defects on torsional fatigue limit of 17-4PH stainless steel, *International Journal of Fatigue*, Vol. 100, pp. 540-548, 2017.

[9] Y. Nishimura, K. Yanase, Y. Ikeda, Y. Tanaka, N. Miyamoto, S. Miyakawa and M. Endo, Fatigue strength of spring steel with small scratches, *Fatigue & Fracture of Engineering Materials & Structures*, Vol. 41(7), pp. 1514-1528, 2018.

[10] K. Yanase, An introduction to FE analysis with Excel-VBA, *Computer Applications in Engineering Education*, Vol. 25, pp.311-319, 2017.

[11] Y. Murakami, *Theory of elasticity and stress concentration*, John Wiley & Sons, 2017.