Chapter 1

A two-dimensional Biharmonic problem with the Bell triangular finite element

In this document, we demonstrate how to solve a two-dimensional Biharmonic equation using triangular finite elements within `oomph-lib`.

First, let consider the Biharmonic equation which is mathematically expressed as follow

$$\frac{\partial^4 u}{\partial x_1^4} + 2\frac{\partial^4 u}{\partial x_1^2 \partial x_2^2} + \frac{\partial^4 u}{\partial x_2^4} = f(x_1, x_2), \quad \mathbf{x} = (x_1, x_2) \in \Re^2. \quad (1)$$

In this study, we consider the homogeneous Biharmonic equation with the exact solution

$$u(x_1, x_2) = \cos(x_1)e^{x_2}.$$

The problem will be implemented in the rectangular domain $\Omega = \{(x_1, x_2) | 0 \leq x_1 \leq 2, 0 \leq x_2 \leq 1\}$, and we apply Dirichlet boundary conditions such that the exact solution is satisfied.

In order to formulate the finite element implementation of the Biharmonic equation, the weak formulation must be derived. The reader is referred to another document for detailed descriptions of weak formulations.

After introducing the weight or test function $\phi(x_i)$ and changing the differential equation (1) into the integral form, we have

$$0 = r = \int_\Omega \left(\frac{\partial^4 u}{\partial x_1^4} + 2\frac{\partial^4 u}{\partial x_1^2 \partial x_2^2} + \frac{\partial^4 u}{\partial x_2^4} - f(x_i) \right) \phi(x_i) d\Omega. \quad (2)$$

Integrating by parts twice and using of the divergence theorem give

$$0 = r = \int_\Omega \left(\frac{\partial^2 u}{\partial x_1^2}\frac{\partial^2 \phi}{\partial x_1^2} + 2\frac{\partial^2 u}{\partial x_1^2}\frac{\partial^2 \phi}{\partial x_2^2} + \frac{\partial^2 u}{\partial x_2^2}\frac{\partial^2 \phi}{\partial x_2^2} - f(x_i)\phi(x_i) \right) d\Omega$$
$$+ \int_{\partial\Omega} \nabla_n^3 u \phi(x_i) d\partial\Omega - \int_{\partial\Omega} \nabla^2 u \frac{\partial \phi}{\partial n}(x_i) d\partial\Omega, \quad (3)$$

where $\partial\Omega$ denotes the domain boundary and $\partial/\partial n$ the outward normal derivative. Since the test functions satisfy homogeneous boundary conditions, $\phi|_{\partial\Omega} = 0 = \left.\frac{\partial \phi}{\partial n}\right|_{\partial\Omega}$, the integrals for the last two terms of equation (3) vanish.

Therefore, an alternative version of the weak form of (1) is given by

$$0 = r = \int_\Omega \left(\frac{\partial^2 u}{\partial x_1^2}\frac{\partial^2 \phi}{\partial x_1^2} + 2\frac{\partial^2 u}{\partial x_1^2}\frac{\partial^2 \phi}{\partial x_2^2} + \frac{\partial^2 u}{\partial x_2^2}\frac{\partial^2 \phi}{\partial x_2^2} - f(x_i)\phi(x_i) \right) d\Omega. \quad (4)$$

It is noteworthy that the Galerkin method is employed in this study so that the choice of function spaces for the solution, $u(x_i)$, and the test function, $\phi(x_i)$, is the same. Therefore, both the solution and the test function can be approximated by the same set of shape function, ψ_l as

$$u(x_i) = \sum_{l=1}^{N} U_l \psi_l(x_i), \quad (5)$$

and,

$$\phi(x_i) = \sum_{k=1}^{N} \Phi_k \psi_k(x_i), \quad (6)$$

where N is the dimension of discrete approximating to function space.

Substituting the derivatives of (6) into (4), we have

$$0 = r = \int_\Omega \left(\frac{\partial^2 u}{\partial x_1^2}\sum_{k=1}^{N}\Phi_k\frac{\partial^2 \psi_k}{\partial x_1^2} + 2\frac{\partial^2 u}{\partial x_1^2}\sum_{k=1}^{N}\Phi_k\frac{\partial^2 \psi_k}{\partial x_2^2} + \frac{\partial^2 u}{\partial x_2^2}\sum_{k=1}^{N}\Phi_k\frac{\partial^2 \psi_k}{\partial x_2^2} - f(x_i)\sum_{k=1}^{N}\Phi_k\psi_k(x_i) \right) d\Omega,$$

$$0 = r = \sum_{k=1}^{N} \Phi_k r_k(U_1, U_2, ..., U_N), \quad (7)$$

where

$$r_k(U_1, U_2, ..., U_N) = \int_\Omega \left(\frac{\partial^2 u}{\partial x_1^2}\frac{\partial^2 \psi_k}{\partial x_1^2} + 2\frac{\partial^2 u}{\partial x_1^2}\frac{\partial^2 \psi_k}{\partial x_2^2} + \frac{\partial^2 u}{\partial x_2^2}\frac{\partial^2 \psi_k}{\partial x_2^2} - f(x_i)\psi_k(x_i) \right) d\Omega. \quad (8)$$

We have that (7) must hold for any value of the coefficients Φ_k, so the coefficients U_j must satisfy the equations

$$r_k(U_1, U_2, ..., U_N) = 0, \quad \text{for } k = 1, 2, ..., N.$$

After substituting the derivatives of (5) into (8), the weak form of (1) can be expressed as

$$0 = r_k = \int_\Omega \left(\sum_{l=1}^{N} U_l \frac{\partial^2 \psi_l}{\partial x_1^2}\frac{\partial^2 \psi_k}{\partial x_1^2} + 2\sum_{l=1}^{N} U_l \frac{\partial^2 \psi_l}{\partial x_1^2}\frac{\partial^2 \psi_k}{\partial x_2^2} + \sum_{k=1}^{N} U_l \frac{\partial^2 \psi_l}{\partial x_2^2}\frac{\partial^2 \psi_k}{\partial x_2^2} - f(x_i)\psi_k \right) dx_1 dx_2. \quad (9)$$

Since we solve the Biharmonic equation with the Dirichlet boundary conditions in this study, the boundary terms in (9) vanish. This is a consequence of every test functions have to satisfy the homogeneous conditions at boundaries. Now, it can be seen that the weak formulation of the equation contains the second-order derivatives of the approximation, $\psi(x_i)$ of the solution. Hence, in order to make the integral exists, the approximation and its first-order derivative are required to be continuous. Therefore, the C^1-interpolations are required.

Hence, the `BellElement` is introduced in this document to provide the continuously differentiable shape functions to interpolate the solution of a Biharmonic equation. This kind of element arises in order to satisfy C^1 continuity of the solutions of a fourth-order problem.

Specifically, we shall provide

- a general descriptions of the Bell triangular finite element
- numerical results of solving the Biharmonic equation with the `BellElement`

and

- how to implement the `BellElement` in `oomph-lib` using the Biharmonic equation as a case study.

1.1 Overview of the BellElement

The constructed `BellElement` provides a discretisation of the variational principle (9) with two-dimensional, three-node triangular finite elements. In this element, the subparametric idea is employed. This means that the degree of geometric shape functions (shape function approximated a geometry) is defined to be lower than the degree of polynomials for parametric shape functions (shape function approximated a variable). In this study, a geometry is approximated by the linear Lagrange polynomials while the Bell shape functions are employed to interpolate variables.
The two-dimensional linear Lagrange shape functions defined over a triangle are parametrised by the local coordinates $s_1, s_2 \in [0, 1]$ and are defined as

$$\psi_1(s_1, s_2) = s_1, \quad \psi_2(s_1, s_2) = s_2, \quad \psi_3(s_1, s_2) = 1 - s_1 - s_2. \quad (10)$$

Therefore, a position $\mathbf{x} = (x_1, x_2) \in \Re^2$ is approximated by

$$x_i(s_1, s_2) = \sum_{j=1}^{3} X_{ij}\ \psi_j(s_1, s_2) \quad \text{for } i = 1, 2, \qquad (11)$$

where X_{ij} are the nodal positions in the direction i^{th} at node j.
Similar to a geometric approximation, a variable can be approximated as

$$u_i(x_1, x_2) = \sum_{l=1}^{3} \sum_{k=1}^{6} U_{ilk}\ \varphi_{lk}(\lambda_1, \lambda_2, \lambda_3),$$

where $x_1, x_2 \in \Omega$ are the element's 2D global coordinates. The function φ_{lk} are the two-dimensional Bell shape functions parametrised by area coordinates $\lambda_1, \lambda_2, \lambda_3$ which we will describe in the next two subsections.

1.1.1 Area coordinates

The area coordinate system or Barycentric coordinates are defined on a triangle which is expressed in terms of its three vertices. We consider a triangle defined by three vertices $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ as

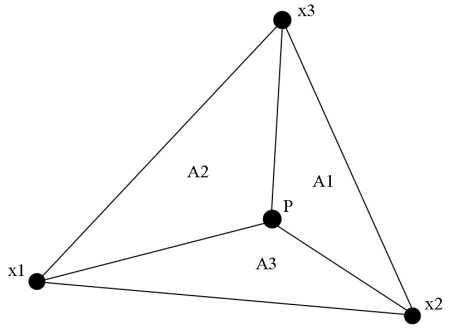

Figure 1.1 The geometry of a triangle which has three nodes.

The barycentric coordinates of point $\mathbf{p} = (x, y)$ in the triangle are simply defined to be ratios of triangular areas as,

$$\lambda_1 = \frac{A_1}{A}, \quad \lambda_2 = \frac{A_2}{A}, \quad \lambda_3 = \frac{A_3}{A}. \quad (12)$$

Area A is the total area of the triangle and can be computed as

$$A = \frac{1}{2}\left((x_2 y_3 - x_3 y_2) + (x_3 y_1 - x_1 y_3) + (x_1 y_2 - x_2 y_1)\right).$$

Also, $A_j, j = 1, 2, 3$ are the sub-area of the triangle described in the figure. They can be computed as

$$A_1 = \frac{1}{2}\left((x_2 y_3 - x_3 y_2) + (x_3 y - x y_3) + (x y_2 - x_2 y)\right),$$

$$A_2 = \frac{1}{2}\left((xy_3 - x_3y) + (x_3y_1 - x_1y_3) + (x_1y - xy_1)\right),$$

$$A_3 = \frac{1}{2}\left((x_2y - xy_2) + (xy_1 - x_1y) + (x_1y_2 - x_2y_1)\right),$$

where x_i, y_i denotes the $x-$ and $y-$ coordinates of the point $\mathbf{x}_i; i = 1, 2, 3$. These ratios of areas form dimensionless coordinates in the plane defined by points $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$.
A barycentric combination of three points takes the form: $\mathbf{p} = \lambda_1\mathbf{x}_1 + \lambda_2\mathbf{x}_2 + \lambda_3\mathbf{x}_3$ where $\lambda_1 + \lambda_2 + \lambda_3 = 1$. The third area coordinate λ_3 can be expressed as $\lambda_3 = 1 - \lambda_1 - \lambda_2$.
The three vertices have the barycentric coordinates as

$$\mathbf{x}_1 = (1, 0, 0), \quad \mathbf{x}_2 = (0, 1, 0), \quad \mathbf{x}_3 = (0, 0, 1).$$

1.1.2 The Bell shape functions

The shape functions of the Bell element are defined over a triangle and derived from the quintic polynomials with 18 degrees of freedom. The element comprises three nodes with six degrees of freedom at each corner node including the value, and all first and second derivatives; $u, u_x, u_y, u_{xx}, u_{yy}, u_{xy}$. The graphical description of the Bell element is illustrated below.

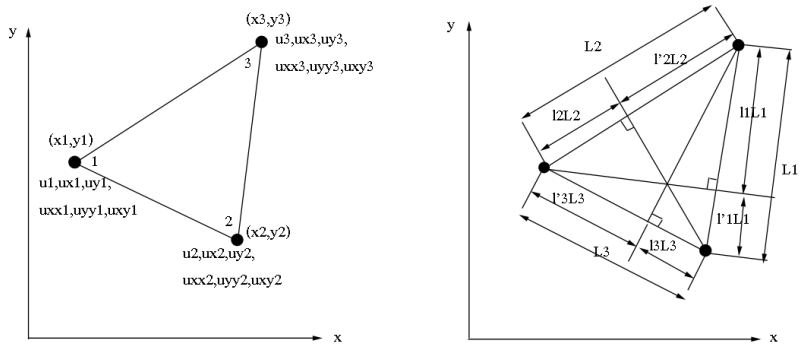

Figure 1.2 (Left)The Bell element with 18 degrees of freedom; 6 dofs at vertices. (Right) The description of all parameters utilised within the derivation of the Bell shape functions.

The shape functions of the Bell element are parametrised by area coordinates, λ_j. The subscripts ik of the shape functions ψ_{ik} mean that they are evaluated at node i with k^{th} degree of freedom where $k = 1$ denotes degree of freedom that corresponds to the unknown value, $k = 2, 3$ correspond to the first derivatives with respect to the first and second global coordinates, $k = 4, 5$ correspond to the second derivatives with respect to the first and second global coordinates, and $k = 6$ corresponds to the mixed derivative.
Unlike the Lagrangian interpolation functions, the Bell shape functions are derived by interpolating nodal derivatives as well as nodal displacements. Also, they have to satisfy the properties that $\psi_{i1} = 1$ at node i and $\psi_{i1} = 0$ at other nodes. Furthermore, all of the first and the second derivatives of $\psi_{i1} = 0$ at all nodes. Also, $\partial_x\psi_{i2} = 1$ at node i and $\partial_x\psi_{i2} = 0$ at other nodes. Similarly for the first derivative with respect to y, we have that $\partial_y\psi_{i3} = 1$ at node i and $\partial_y\psi_{i3} = 0$ at other nodes. These shape functions associated with the first derivatives are zero at all nodes for ψ_{im}, $\partial_{xx}\psi_{im}$, $\partial_{yy}\psi_{im}$, $\partial_{xy}\psi_{im}, \forall m = 2, 3$. Moreover, $\partial_{xx}\psi_{i4}(\mathbf{x}_k) = \delta_{ik}$, $\partial_{yy}\psi_{i5}(\mathbf{x}_k) = \delta_{ik}$, and $\partial_{xy}\psi_{i6}(\mathbf{x}_k) = \delta_{ik}$. Furthermore, these shape functions associated with the second derivatives are zero at all nodes for ψ_{im}, $\partial_x\psi_{im}$, and $\partial_y\psi_{im}, \forall m = 4, 5, 6$.
The Bell shape functions are mathematically expressed as

$$\psi_{11}(\lambda_1, \lambda_2, \lambda_3) = \lambda_1^5 + 5\lambda_1^4(\lambda_2 + \lambda_3) + 10\lambda_1^3(\lambda_2 + \lambda_3)^2 + 30\lambda_1^2\lambda_2\lambda_3(l_3\lambda_2 + l_2'\lambda_3),$$

$$\psi_{12}(\lambda_1, \lambda_2, \lambda_3) = 3b_1\lambda_1^2\lambda_2\lambda_3(\lambda_2 - \lambda_3) + \lambda_1^3(b_3\lambda_2 - b_2\lambda_3)(\lambda_1 + 4\lambda_2 + 4\lambda_3)$$
$$+15\lambda_1^2\lambda_2\lambda_3(b_3l_3\lambda_2 - b_2l_2'\lambda_3),$$

$$\psi_{13}(\lambda_1,\lambda_2,\lambda_3) = -3c_1\lambda_1^2\lambda_2\lambda_3\left(\lambda_2-\lambda_3\right) - \lambda_1^3\left(c_3\lambda_2 - c_2\lambda_3\right)\left(\lambda_1+4\lambda_2+4\lambda_3\right)$$

$$-15\lambda_1^2\lambda_2\lambda_3\left(c_3 l_3\lambda_2 - c_2 l_2'\lambda_3\right),$$

$$\psi_{14}(\lambda_1,\lambda_2,\lambda_3) = 1/2\lambda_1^3\left(b_3^2\lambda_2^2 + b_2^2\lambda_3^2\right) + \lambda_1^2\lambda_2\lambda_3\left(-b_2b_3\lambda_1 + b_3b_1\lambda_2 + b_1b_2\lambda_3\right)$$

$$+5/2\lambda_1^2\lambda_2\lambda_3\left(b_3^2 l_3\lambda_2 + b_2^2 l_2'\lambda_3\right),$$

$$\psi_{15}(\lambda_1,\lambda_2,\lambda_3) = 1/2\lambda_1^3\left(c_3^2\lambda_2^2 + c_2^2\lambda_3^2\right) + \lambda_1^2\lambda_2\lambda_3\left(-c_2c_3\lambda_1 + c_3c_1\lambda_2 + c_1c_2\lambda_3\right)$$

$$+5/2\lambda_1^2\lambda_2\lambda_3\left(c_3^2 l_3\lambda_2 + c_2^2 l_2'\lambda_3\right),$$

$$\psi_{16}(\lambda_1,\lambda_2,\lambda_3) = b_1c_1\lambda_1^2\lambda_2\lambda_3\left(\lambda_1+\lambda_2+\lambda_3\right) + b_2c_2\lambda_1^2\lambda_3\left(\lambda_2\lambda_3 - \lambda_3\lambda_1 - \lambda_1\lambda_2 - \lambda_2^2\right)$$

$$+b_3c_3\lambda_1^2\lambda_2\left(\lambda_2\lambda_3 - \lambda_3\lambda_1 - \lambda_1\lambda_2 - \lambda_3^2\right) - 5\lambda_1^2\lambda_2\lambda_3\left(b_2c_2l_2'\lambda_3 + b_3c_3l_3\lambda_2\right). \quad (13)$$

These are shape functions associated with the first node with its 6 degrees of freedom. The shape functions at another two nodes can be obtained by performing a cyclic permutation of the Barycentric coordinates in (13). The parameters $b_i, c_i \forall i = 1,2,3$, appearing in the equations can be obtained from

$$b_1 = x_3 - x_2, \quad b_2 = x_1 - x_3, \quad b_3 = x_2 - x_1,$$

$$c_1 = y_2 - y_3, \quad c_2 = y_3 - y_1, \quad c_3 = y_1 - y_2,$$

where $x_i, y_i; i = 1,2,3$, are components of a vertex $\mathbf{x}_i$.
Note that all the first and second derivatives can be obtained by the help of the chain rule as

$$\frac{\partial\psi_{ij}}{\partial x_\alpha} = \frac{\partial\psi_{ij}}{\partial\lambda_k}\frac{\partial\lambda_k}{\partial x_\alpha},$$

$$\frac{\partial^2\psi_{ij}}{\partial x_\alpha\partial x_\beta} = \frac{\partial^2\psi_{ij}}{\partial\lambda_m\partial\lambda_n}\frac{\partial\lambda_m}{\partial x_\alpha}\frac{\partial\lambda_n}{\partial x_\beta} + \frac{\partial^2\lambda_k}{\partial x_\alpha\partial x_\beta}\frac{\partial\psi_{ij}}{\partial\lambda_k},$$

and the derivatives of area coordinates with respect to global coordinates can be obtained from considering (12) with the substitutions of $b_i, c_i; i = 1,2,3$. They are expressed as follows:

$$\frac{\partial\lambda_1}{\partial x} = \frac{c_1}{2A}, \quad \frac{\partial\lambda_2}{\partial x} = \frac{c_2}{2A}, \quad \frac{\partial\lambda_3}{\partial x} = \frac{c_3}{2A},$$

$$\frac{\partial\lambda_1}{\partial y} = \frac{b_1}{2A}, \quad \frac{\partial\lambda_2}{\partial y} = \frac{b_2}{2A}, \quad \frac{\partial\lambda_3}{\partial y} = \frac{b_3}{2A}.$$

Here are graphical Bell shape functions at the first node, $\psi_{1j}, j = 1,...,6$, evaluated at $\mathbf{x}_1$. The figures show that the Bell shape function at the first node have to satisfy the properties that $\psi_{11} = 1$ at node 1 and $\psi_{11} = 0$ at other nodes. Furthermore, all of the Bell shape functions corresponded to all first and second derivatives $\psi_{1j} = 0, j = 2,...,6$ at all nodes.

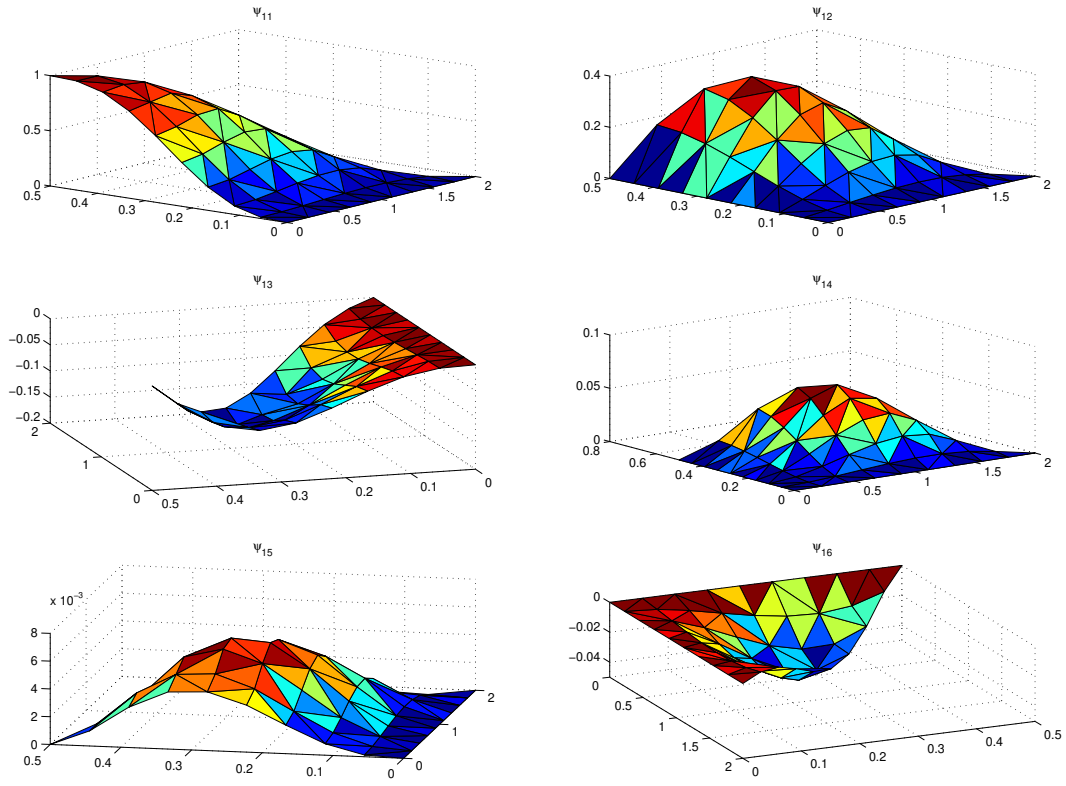

Figure 1.3 The Bell shape functions that correspond to the 6 degrees of freedom defined previously at the first node. Here the shape functions are evaluated at x1 which show the appreciation of the crucial properties in the Bell shape functions.

Note that the Bell shape functions and also the first and second derivatives are differentiated with respect to global coordinates. Therefore, the Jacobian of mapping between global to local coordinates is no longer needed to derive the derivatives of the shape functions for this type of element.

1.1.3 The interface of accessible functions to the shape functions in the BellElement

Since the `BellElement` is constructed with the subparametric idea, there are two sets of approximating functions defined in the element. The linear Lagrange polynomials defined in (10) will be inherited from `TElement<2,2>` in order to approximate the geometry. These functions can be accessible by

```
BellElement<>::shape(s,psi).
```

In order to approximate variables in the `BellElement`, the Bell shape functions defined in (13) will be overloaded from `BellElementShape::Bshape(...)`. This function will compute the basis functions at local coordinate s and it provides C^1-continuity.
As the global coordinates are required in the derivation of the shape functions of the `BellElementShape<>`, the physical coordinates of vertices have to be passed as an argument when shape functions are overloaded. The interface of the Bell shape functions obtained from `BellElementShape<>` at local coordinate s is `BellElementShape<>::Bshape(s,psi,position)`. Similarly, the first and the second-order derivatives can be obtained at local coordinate s as `BellElementShape<>::dBshape(s,psi,dpsi,position)` and `BellElementShape<>::d2Bshape(s,psi,dpsi,d2psi,position)`, respectively.

Note that the vector `position` is the collection of coordinates on vertices of Bell triangles. The order of nodal positions defined in the vector `position` is correspond with the order of those on physical elements. Also, the vertex nodes 0,1, and 2 situate anticlockwise.
Now, to obtained the basis functions that employed to approximate variables in `BellElement`, they can be accessible via

```
BellElement<>::basis(s,psi),
```

and, the first and second-order derivatives of the Bell shape functions can be accessible via

```
BellElement<>::dbasis(s,psi,dpsi) BellElement<>::d2basis(s,psi,dpsi,d2psi),
```

respectively.

1.2 Results

Before the numerical results obtained from solving the 2D Biharmonic equation with the Bell finite element will be illustrated, the boundary conditions have to be specified. Since the Bell finite element is employed, the boundary specifications have to correspond with degrees of freedom defined for the `BellElement` in the previous section. Since we consider the Biharmonic equation with the Dirichlet boundary conditions, we have that the physical conditions that allow to be pinned on boundaries are the value and the normal derivatives. However, there is no normal derivative defined as a degree of freedom to be specified for the Bell shape function. Therefore, the first-order derivatives have to be imposed instead and the boundary specification can be worked out from the normal derivative on that boundary.
Furthermore, the values of the unknown and the first-order derivatives that have to be specified on the boundaries of the domain for the Bell element can be obtained from

$$u(x_1,x_2)=\cos(x_1)e^{x_2},\quad u_{x_1}(x_1,x_2)=-\sin(x_1)e^{x_2},\quad u_{x_2}(x_1,x_2)=\cos(x_1)e^{x_2}. \quad (14)$$

For the rest of degrees of freedom associated with the second-order derivatives, they are derived conditions that can be taken care by the natural boundary conditions. In this study, we set them to be pinned in order to reduce the number of degrees of freedom in the problem. The values of the second-order derivatives that have to be specified on the boundaries of the domain for the Bell element can be worked out from (14) and are expressed as

$$u_{x_1x_1}(x_1,x_2)=-\cos(x_1)e^{x_2},\quad u_{x_1x_2}(x_1,x_2)=-\sin(x_1)e^{x_2},\quad u_{x_2x_2}(x_1,x_2)=\cos(x_1)e^{x_2}.$$

The following is a table illustrating the performance with the associated computational time of the Bell triangular elements. The accuracy of the solutions will base on the L^2-norm error which is mathematically described as

$$|u_{exact}-u_{FE}|=\left(\int_\Omega |u_{exact}-u_{FE}|^2 d\Omega\right)^{1/2},$$

where u_{exact}, u_{FE} denote the exact and the finite element solutions, respectively.
Table 1: L^2-norm error of the solution obtained from the Biharmonic implementation using the Bell element with various numbers of elements.

Element size (h)	Number of elements	Number of dofs	Error	Time (sec)
0.5	16	18	3.45428×10^{-5}	0.18
0.2	100	216	7.78578×10^{-7}	1.19
0.1429	196	468	1.89314×10^{-7}	3.74
0.1	400	1026	4.29805×10^{-8}	4.77
0.005	1600	4446	2.52419×10^{-9}	19.67
0.01	40000	118206	3.86145×10^{-12}	1197.06

The figures below illustrate, respectively, the finite element solution and absolute error obtained from solving the Biharmonic equation with the `BellElement` on structured meshes. This implementation employs 16 triangular elements in the mesh with the number of degrees of freedom equals to 18.

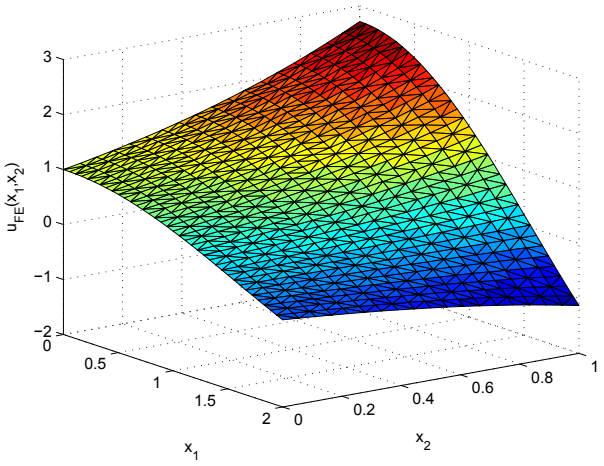

Figure 1.4 The finite element solution of the Biharmonic equation obtained from the BellElement.

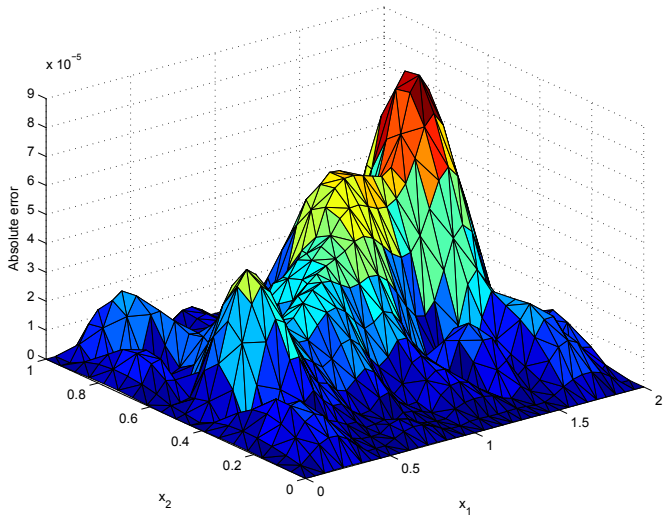

Figure 1.5 Absolute error between the exact and the finite element solutions of the Biharmonic equation obtained from the BellElement.

Next, plotting the logarithm of L^2-error with that of the element size shows that the obtained rate of convergence of the Bell element is quartic (see figure 1.4).

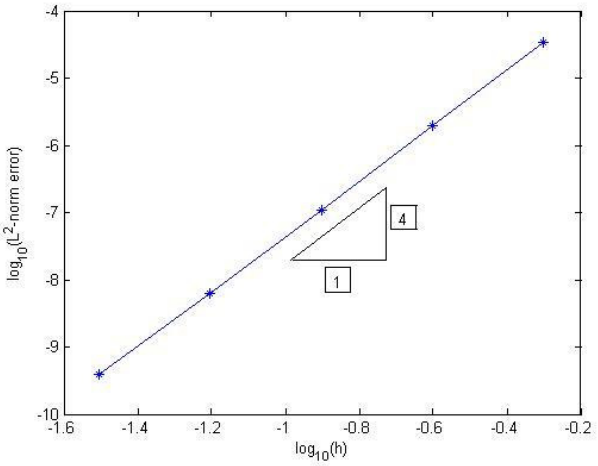

Figure 1.6 A log-plot between error and the element size from solving the Biharmonic equation with the BellElement in a rectangular domain.

1.3 Implementation in oomph-lib

1.3.1 Global parameters and functions

The namespace `Physical_Variables` is where the source function and the exact solution are defined. The source function can be specified via `Physical_Variables::source_function()` while the exact solutions are defined via `Physical_Variables::get_exact_u()`. Note that the six exact solutions correspond to the six degrees of freedom defined on each node.

```
//==start_of_namespace=================================================
/// Namespace for the solution of 2D Biharmonic equation
//=====================================================================
namespace Physical_Variables
{

 /// number of elements in the x direction
 unsigned n_x = 4;

 /// number of elements in the y direction
 unsigned n_y = 2;

 /// length in the x direction
 double l_x = 2.0;

 /// length in the y direction
 double l_y = 1.0;

 /// Exact solution as a Vector
 void get_exact_u(const Vector<double>& x, Vector<double>& u)
 {
  u[0] = cos(x[0])*exp(x[1]);
  u[1] = -sin(x[0])*exp(x[1]);
  u[2] = cos(x[0])*exp(x[1]);
  u[3] = -cos(x[0])*exp(x[1]);
  u[4] = cos(x[0])*exp(x[1]);
  u[5] = -sin(x[0])*exp(x[1]);
 }

 /// Source function required to make the solution above an exact solution
 void source_function(const Vector<double>& x, double& source)
 {
  source = 0.0;
 }

} // end of namespace
```

1.3.2 The driver code

The driver code is very simple and short. It is where the problem is defined. In this study, the problem is constructed using the structured mesh with triangular elements in 2D. A number of nodes in an element has to be specific as

a template parameter in the problem set up. This is crucial in order to take care of element nodes which we will describe in another tutorial. Following the usual self-test, we call the function `BiharmonicProblem::newton_solve()` to compute the solution of the problem. `BiharmonicProblem::doc_solution()` will output the solution afterwards.

```
//======start_of_main=====================================================
/// Driver for 2D Biharmonic problem
//========================================================================
int main()
{
 // Set up the problem:
 // Solve a 2D Biharmonic problem
 MyBiharmonicBellProblem<BiharmonicBellElement<2,2>,2,2> //Element type as template parameter
  problem(Physical_Variables::source_function);

 // Check whether the problem can be solved
 cout « "\n\n\nProblem self-test ";
 if (problem.self_test()==0)
  {
   cout « "passed: Problem can be solved." « std::endl;
  }
 else
  {
   throw OomphLibError("failed!","main()",OOMPH_EXCEPTION_LOCATION);
  }

 // Set up doc info
 DocInfo doc_info;
 doc_info.set_directory("RESLT");
 doc_info.number()=0;
 // Solve the problem
 problem.newton_solve();

 //Output the solution
 problem.doc_solution(doc_info);
} // end of main
```

1.3.3 The problem class

The problem class has four member functions, illustrated as follows:

- The problem constructor
- `action_before_newton_solve()` : Update the problem specifications before solve. Boundary conditions maybe set here.
- `action_after_newton_solve()` : Update the problem specifications after solve.
- `doc_solution()` : Pass the number of the case considered, so that output files can be distinguished.

From the above mentioned functions, only the problem constructor is non-trivial.
In the present problem, the function `Problem::actions_after_newton_solve()` is not required, so it remains empty. Also, the class includes a private data member which store a pointer to a source function.

```
//==start_of_problem_class============================================
/// 2D Biharmonic problem.
//====================================================================
template<class ELEMENT, unsigned DIM, unsigned NNODE_1D>
class MyBiharmonicBellProblem : public Problem
{
public:

 /// Constructor: Pass number of elements and pointer to source function
 MyBiharmonicBellProblem(typename MyBiharmonicEquations<DIM,NNODE_1D>::SourceFctPt source_fct_pt);

 /// Destructor (empty)
 ~MyBiharmonicBellProblem()
  {
   delete mesh_pt();
  }

 /// Update the problem specs before solve: (Re)set boundary conditions
 void actions_before_newton_solve();

 /// Update the problem specs after solve (empty)
 void actions_after_newton_solve(){}

 /// \short Doc the solution, pass the number of the case considered,
 /// so that output files can be distinguished.
 void doc_solution(DocInfo& doc_info);

private:
```

```
 /// Pointer to source function
 typename MyBiharmonicEquations<DIM,NNODE_1D>::SourceFctPt Source_fct_pt;

}; // end of problem class
```

1.3.4 The Problem constructor

The problem constructor starts by overloading the function `Problem::mesh_pt()` and set to the specific mesh used in this problem. In this tutorial, we implement the problem with 2D triangular structured mesh which is internally created by `SimpleRectangularTriMesh<>`. The generated output will be used to build `oomph-lib` mesh.

```
//=====start_of_constructor==============================================
/// \short Constructor for 2D Biharmonic problem.
/// Discretise the 2D domain with n_element elements of type ELEMENT.
/// Specify function pointer to source function.
//=======================================================================
template<class ELEMENT, unsigned DIM, unsigned NNODE_1D>
MyBiharmonicBellProblem<ELEMENT,DIM,NNODE_1D>::MyBiharmonicBellProblem
(typename MyBiharmonicEquations<DIM,NNODE_1D>::SourceFctPt source_fct_pt) :
 Source_fct_pt(source_fct_pt)
{
 // Build mesh and store pointer in Problem
 Problem::mesh_pt() = new SimpleRectangularTriMesh<ELEMENT>
(Physical_Variables::n_x,Physical_Variables::n_y,Physical_Variables::l_x,Physical_Variables::l_y);
```

Next, the boundary conditions of the problem will be taken care. We pin the nodal values on all boundaries and apply the Dirichlet boundary conditions. Note that the boundary identities are labelled anticlockwise where the first boundary is associated with the side $(x, y) = (0, y)$.

```
 // start_of_boundary_conditions
 unsigned n_bound = mesh_pt()->nboundary();
 for(unsigned i=0;i<n_bound;i++)
  {
   unsigned n_node = mesh_pt()->nboundary_node(i);
   for (unsigned n=0;n<n_node;n++)
    {
     mesh_pt()->boundary_node_pt(i,n)->pin(0);
     mesh_pt()->boundary_node_pt(i,n)->pin(1);
     mesh_pt()->boundary_node_pt(i,n)->pin(2);
     mesh_pt()->boundary_node_pt(i,n)->pin(3);
     mesh_pt()->boundary_node_pt(i,n)->pin(4);
     mesh_pt()->boundary_node_pt(i,n)->pin(5);
    }
  }
 // end of boundary conditionss
```

We then loop over the elements and set the pointer to the physical parameters, the function pointer to the source function which is the function on the right-hand side of the Biharmonic equation.

```
 // Loop over elements and set pointers to Physical parameters
 for(unsigned i=0;i<n_element;i++)
  {
   // Upcast from GeneralisedElement to the present element
   ELEMENT *elem_pt = dynamic_cast<ELEMENT*>(mesh_pt()->element_pt(i));

   //Set the source function pointer and all physical variables
   elem_pt->source_fct_pt() = Source_fct_pt;
  }
 // end of pointers set up
```

We finish the constructor by assigning the equation numbering scheme.

```
 // Setup equation numbering scheme
 assign_eqn_numbers();
} // end of constructor
```

1.3.5 Action before newton solve

In the `action_before_newton_solve()`, the problem specifications will be updated before performing the newton solve. The boundary values will be (re)set from the exact solutions.

```
//===start_of_actions_before_newton_solve=================================
/// \short Update the problem specs before solve: (Re)set boundary values
/// from the exact solution.
//========================================================================
template<class ELEMENT, unsigned DIM, unsigned NNODE_1D>
void MyBiharmonicBellProblem<ELEMENT,DIM,NNODE_1D>::actions_before_newton_solve()
{
 // How many boundaries are there?
 unsigned n_bound = mesh_pt()->nboundary();

 //Loop over the boundaries
 for(unsigned i=0;i<n_bound;i++)
  {
   // How many nodes are there on this boundary?
   unsigned n_node = mesh_pt()->nboundary_node(i);
   // Loop over the nodes on boundary
```

```
   for (unsigned n=0;n<n_node;n++)
    {
     // Get pointer to node
     Node* nod_pt=mesh_pt()->boundary_node_pt(i,n);
     // Extract nodal coordinates from node:
     Vector<double> x(2);
     x[0]=nod_pt->x(0);
     x[1]=nod_pt->x(1);
     // Compute the value of the exact solution at the nodal point
     Vector<double> u(6);
     for(unsigned i=0;i<6;i++)
      {
       Physical_Variables::get_exact_u(x,u);

       // Assign the value to the one (and only) nodal value at this node
       nod_pt->set_value(i,u[i]);
      }
    }
  }
} // end of actions before solve
```

1.3.6 Post-processing

The post-processing routine writes the computed results to an output file.

```
//===start_of_doc=====================================================
/// Doc the solution in tecplot format. Label files with label.
//====================================================================
template<class ELEMENT, unsigned DIM, unsigned NNODE_1D>
void MyBiharmonicBellProblem<ELEMENT,DIM,NNODE_1D>::doc_solution(DocInfo& doc_info)
{
 ofstream some_file;
 char filename[100];
 // Number of plot points
 unsigned npts;
 npts=10;
 // Output solution with specified number of plot points per element
 sprintf(filename,"%s/soln%i.dat",doc_info.directory().c_str(),
         doc_info.number());
 some_file.open(filename);
 some_file.precision(20);
 mesh_pt()->output(some_file,npts);
 some_file.close();
 // Output exact solution
 sprintf(filename,"%s/exact_soln%i.dat",doc_info.directory().c_str(),
         doc_info.number());some_file.open(filename);
 some_file.open(filename);
 mesh_pt()->output_fct(some_file,npts,Physical_Variables::get_exact_u);
 some_file.close();
 // Doc pointwise error and compute norm of error and of the solution
 double error,norm;
 sprintf(filename,"%s/error%i.dat",doc_info.directory().c_str(),
         doc_info.number());
 some_file.open(filename);
 mesh_pt()->compute_error(some_file,Physical_Variables::get_exact_u,
                          error,norm);
 some_file.close();
 // Doc error norm:
 cout << "\nNorm of error    : " << sqrt(error) << std::endl;
 cout << "Norm of solution : " << sqrt(norm) << std::endl << std::endl;
 cout << std::endl;
} // end of doc
```

1.4 Source files for this tutorial

- The source files for this tutorial are located in the directory:

  demo_drivers/biharmonic/structured_2d_biharmonic/

- The driver code is:

demo_drivers/biharmonic/structured_2d_biharmonic/structured_2d_↩
biharmonic_bellelement.cc

1.5 PDF file

A pdf version of this document is available.