Method of Finite Elements I

PhD Candidate - Charilaos Mylonas
HIL H33.1

Quadrature and Boundary Conditions, 26 March, 2018
Outline

1 Quadrature
   - Integration in 1D
   - Integration in 2D and 3D

2 Boundary conditions
   - Penalty method
   - Lagrange multiplier method
Outline

1 Quadrature
   - Integration in 1D
   - Integration in 2D and 3D

2 Boundary conditions
   - Penalty method
   - Lagrange multiplier method
Integration

Basic numerical integration

- Riemann sum
- Trapezoidal rule

Comments:
Both simple but inefficient!
- useful techniques in other contexts
- we use many more points than needed in order to achieve good accuracy
example: Euler-Bernoulli beam

We are going to apply the Galerkin method to the Euler Bernoulli beam, but with the basis

\[ N_1(x) = x^2 \quad N_2(x) = x^3 \quad N_3(x) = x^4 \quad \ldots \]

Instead of the Hermite basis functions, and discretize only displacements! Strong form:

\[ \frac{d^2}{dx^2} \left( EI(x) \frac{d^2 w}{dx^2} \right) = f(x) \]

multiply with test function \( u(x) \) and integrate by parts twice \(^a\)

\[ \int_0^1 EI(x) \frac{d^2 w(x)}{dx^2} \frac{d^2 u(x)}{dx^2} dx = \int_0^1 f(x)u(x)dx \]

\(^a\)boundary terms \([\cdot]_0^1\) zero by basis selection
The full commented code of the following demo will be made available to you in case you would like to play around. This code is a comparison between analytical and numerical evaluation of integrals needed in the implementation of the Galerkin procedure. In the following slides the most important parts of the code are highlighted. Namely:

- The definition of the load
- The assembly of the system with analytical integration (symbolic integration)
- The assembly of the system with numerical integration (Gauss-Legendre)
syms x % Coordinate axis
syms q(x)
% The loading of the beam:
q(x) = x.^2 - 0.3*x.^4
NBASIS = 4;
for i = 1:NBASIS
    Phi(i) = x.^(i+1);
end
% Compute the integrals needed for the stiffness matrix:
for i=1:NBASIS
    for j = 1:NBASIS
        K(i,j)= ...
        int(EI*diff(Phi(i),2)*diff(Phi(j),2),0,L);
    end
% Analytical computation of the load vector:
    f(i) = int(q(x) * Phi(i),0,L);
end
Numerical integration for LHS

```matlab
quadr_order = QUAD_ORD;

% Loading the Gauss–Legendre quadrature nodes
% and weights for a file:
nodes_weights = W{quadr_order - 1};
nodes = (nodes_weights.nodes +1)*L/2;
weights = W{quadr_order - 1}.weights;

for i=1:N_basis
    for j=1:N_basis
        V = eval(subs(EI * diff(Phi(i),2) * diff(Phi(j),2),nodes));
        K_approx(i,j) = V' * weights;
        if note that we don’t integrate analytically!
            % we can get EXACT results without doing so
    end
end

f_approx(i) = eval(subs(Phi(i)*q(x),nodes))' * weights;
end
```
Numerical Results
Notice the errors in the approximation ($1.7e^{-13}$ for quadrature of order 4!). This is close to machine precision (the "eps" command in matlab).

1D Quadrature order "N" means we evaluate the function inside the integral "N" times, and we sum the results according to some weight (see next slide).

Instead of integrating analytically a function, we can get EXACT results by carefully selecting points of evaluation (instead of regular intervals as in trapezoidal and Riemann sums).

The accuracy of the approximation depends on the Gaussian quadrature order
Optimal points and weights integration for polynomials:

Quadrature - Optimal weights given the nodes

For a polynomial \( f(x) = a_0 + a_1 x + \cdots + a_n x^n \), the integral is approximated as

\[
\int_{-1}^{1} f(x) dx \simeq \sum_{i=1}^{N} w_i f(x_i)
\]

Assume \( x_i \) are given. Also, for simplicity, \( f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \). Then,

\[
a_0 \int_{-1}^{1} dx + a_1 \int_{-1}^{1} x dx + a_2 \int_{-1}^{1} x^2 dx + a_3 \int_{-1}^{1} x^3 dx =
\]

\[
\sum_{i=1}^{4} w_i (a_0 + a_1 x_i + a_2 x_i^2 + a_3 x_i^3)
\]

We equate all the terms, for every \( a_i \) term. we arrive at the following linear system:

\[
\begin{bmatrix}
1 & 1 & 1 & 1 \\
x_1 & x_2 & x_3 & x_4 \\
x_1^2 & x_2^2 & x_3^2 & x_4^2 \\
x_1^3 & x_2^3 & x_3^3 & x_4^3
\end{bmatrix}
\begin{bmatrix}
w_1 \\
w_2 \\
w_3 \\
w_4
\end{bmatrix}
=
\begin{bmatrix}
\int_{-1}^{1} dx \\
\int_{-1}^{1} x dx \\
\int_{-1}^{1} x^2 dx \\
\int_{-1}^{1} x^3 dx
\end{bmatrix}
\]
Non-Gaussian Quadrature

- N-point quadrature, *exact* for polynomials up to degree N-1
- Weights can be negative, in practice they have values of very different scales and solution of the system for high order quadrature is numerically unstable!
- The technique will approximate well integrals of functions that are approximated well with the corresponding polynomials.
- Lower order quadrature works, but as we will see it is far from the best we can do! (half as good).
In the following, we will see how the Legendre polynomials are important for quadrature in 1D. We are to use $x_i$ such that this integral is exact for every polynomial $f(x)$ with degree $n < 2N - 1$. For a given $N$, solution $x_i$ are roots of the Legendre orthogonal polynomial of degree $N$. 

![Legendre polynomials graph](image)

**roots for N=5**
Orthogonal polynomials

Legendre orthogonal polynomials \( L_n \) satisfy (by construction)\(^a\):

\[
\langle L_k, L_m \rangle = \int_{-1}^{1} L_k(x) L_m(x) \, dx = \frac{2}{2m + 1} \delta_{mk}
\]

From now on this integral is going to be referred to as \( L^2 \) inner product. Formulas for the first few polynomials are\(^b\),

\[
\begin{align*}
L_0(x) &= 1 \\
L_1(x) &= x \\
L_2(x) &= \frac{1}{2} (3x^2 - 1) \\
L_3(x) &= \frac{1}{8} (5x^3 - 3x) \\
L_4(x) &= \frac{1}{8} (35x^4 - 30x^2 + 3)
\end{align*}
\]

\(^a\)There are more general definitions of orthogonality and Gaussian quadrature where \( \langle f, g \rangle_w = \int_\Omega f(x) g(x) w(x) \, dx \). The resulting polynomials are useful in other important contexts (stochastic finite elements!)

\(^b\)They are computed with a process known as Gram-Schmidt orthogonalization
Legendre orthogonal polynomials and quadrature

The roots of Legendre polynomials of degree $N$, used as integration nodes, yield the exact integral for $f$ polynomial of maximum degree $2N - 1$.

Outline of proof:

1. consider a polynomial $f_{2N-1}(x) = a_0 + a_1 x + \cdots + a_{2N-1} x^{2N-1}$

2. Apply polynomial division with Legendre polynomial of degree $N$:

$$f_{2N-1}(x) = \underbrace{g(x)}_{\text{degree } \leq N-1} L_N(x) + \underbrace{r(x)}_{\text{degree } \leq N-1}$$

3. $\{x_1^{(N)}, \cdots, x_N^{(N)}\}$ are the $N$ roots of the Legendre polynomial of degree $N$. For brevity, we omit the superscript $x_i^{(N)} = x_i$. Chose the quadrature nodes to be these exact points:

$$\int_{-1}^{1} g(x) L_n(x) \, dx + \int_{-1}^{1} r(x) \, dx = \sum_{i=1}^{N} w_i g(x_i) L_n(x_i) + \sum_{i=1}^{N} w_i r(x_i)$$

4. Since $L_n(x_i) = 0$, the first term of the RHS vanishes. Moreover, since the degree of $g(x)$ is lower than $N$, the first integral term on the left is also zero! That is, through orthogonality and the ability to express polynomials of order $N-1$ with a polynomial basis that contains all the monomials (such as the Legendre basis order $N$)!
Considering again that we are evaluating the quadrature rule on the roots of Legendre polynomial of degree $N$, the term $L_N$ becomes zero

$$f_{2N-1}(x_i) = g(x_i)L_N(x_i) + r(x_i) = r(x_i)$$

Finally we are left with

$$\int_{-1}^{1} f_{2N-1}(x)dx = \int_{-1}^{1} r(x) = \sum_{i=1}^{N} w_i f_{2N-1}(x_i)$$

And we are left with the task of determining the weights as in the non-gaussian case. Therefore, surprizingly, a $N$-point gauss quadrature rule is exact for polynomials up to degree $2N - 1$.

In practice, the computational determination of weights $w_i$ and nodes $x_i$ of 1D Gaussian quadrature rules happens through the so-called Golub-Welsch algorithm.
**Tensor-product Quadrature**

Take tensor product of the 1D rules. Example: Quadrature in $x$:

$$w_i^{(\xi)} = \{w_1^{(\xi)}, w_2^{(\xi)}, w_3^{(\xi)}, w_4^{(\xi)}\}$$

$$\xi_i = \{\xi_1, \xi_2, \xi_3, \xi_4\}$$

Quadrature in $y$

$$w_i^{(\eta)} = \{w_1^{(\eta)}, w_2^{(\eta)}, w_3^{(\eta)}, x_4^{(\eta)}\}$$

$$\eta_i = \{\eta_1, \eta_2, \eta_3, \eta_4\}$$

Quadrature rule for quadrilateral:

$$\int_{-1}^{1} \int_{-1}^{1} f(\xi, \eta) \, dx \, dy = \sum_{i=1}^{4} \sum_{j=1}^{4} w_i^{(\eta)} w_j^{(\xi)} f(\xi_i, \eta_j)$$
Outline

1 Quadrature
   - Integration in 1D
   - Integration in 2D and 3D

2 Boundary conditions
   - Penalty method
   - Lagrange multiplier method
Linear system from FE discretization

Consider a linear system from a FE discretization:

\[
[K]\{u\} = \{f\}
\]

\[
\begin{bmatrix}
    K_{11} & \cdots & K_{1N} \\
    \vdots & \ddots & \vdots \\
    K_{N1} & \cdots & K_{NN}
\end{bmatrix}
\begin{bmatrix}
    u_1 \\
    \vdots \\
    u_N
\end{bmatrix}
= 
\begin{bmatrix}
    f_1 \\
    \vdots \\
    f_N
\end{bmatrix}
\]

Homogeneous constraints

the constraint \( u_n = 0 \), in a matrix form consistent with our FE linear system reads,

\[
\sum_{i=0}^{N} v_i u_i = 0
\]

\[
\begin{bmatrix}
    0 & \cdots & 1 & \cdots & 0
\end{bmatrix}
\begin{bmatrix}
    u_1 \\
    \vdots \\
    u_n \\
    \vdots \\
    u_N
\end{bmatrix}
= 0
\]

Institute of Structural Engineering
Method of Finite Elements I
Imposing homogeneous constr. with the Penalty method

Imposing the constraint

The constraint vector should be in a form consistent with the matrix (a bilinear form that corresponds to a matrix).

In order to impose the constraint, we simply multiply with a large number $p >> \text{tr}([K])$ and add that to our linear system.

\[
[V] = V_{ij} = v_i v_j \\
[V] = \{v\}^T \{v\} \\
[V] = \begin{bmatrix}
0 & \ldots & 0 \\
\vdots & \ddots & \vdots \\
0 & \ldots & 0
\end{bmatrix}
\]

This forces the degrees of freedom in the system to satisfy the constraint numerically (not explicitly).
Adding to the linear system

Adding the constraint

Simply set,

\[ [K] \leftarrow [K]_{pen} \]

with

\[ [K]_{pen} = [K] + [V] \cdot p \]

with \( p >> tr([K]) \) and solve

\[ \{u\} = [K]^{-1}_{pen} \{f\} \]

Inhomogeneous constraints and MFC:

What if, we need to set a displacement to a given value (inhomogeneous constraint), or a displacement the same as another displacement (multi-DOF constraints)?
Consider we have a constraint of the form $2u_5 + 3u_7 - 2u_{10} = 0$. Again, written as an equation,

$$\begin{bmatrix}
0 & \ldots & 2 & 0 & 3 & 0 & 0 & -2 & \ldots & 0
\end{bmatrix}
\begin{bmatrix}
u_1 \\
\vdots \\
u_5 \\
u_6 \\
u_7 \\
u_8 \\
u_9 \\
u_{10} \\
\vdots \\
u_N
\end{bmatrix} = 0$$
Penalty, multi-DOF constraints

Application of the constraints

Not much change when being systematic about the application!

$$[\mathbf{V}] = \{v\}^T \{v\}$$

$$[\mathbf{V}] = \begin{bmatrix}
0 & \cdots & 0 \\
\vdots & & \vdots \\
4 & 0 & 6 & 0 & 0 & -4 \\
0 & 0 & 0 & 0 & 0 & 0 \\
6 & 0 & 9 & 0 & 0 & -6 \\
\vdots & & \vdots & & & \\
0 & 0 & 0 & 0 & 0 & 0 \\
-4 & 0 & -6 & 0 & 0 & 4 \\
0 & \cdots & 0
\end{bmatrix}$$

The rest are the same! (multiply by a large number and add to $[\mathbf{K}]$ before solution).
Non-homogeneous constraints with the Penalty method and multiple constraints

Non-zero RHS

Say, we want to constrain \( u_3 - 2u_5 = 3 \) and at the same time \( u_2 - u_3 = 0 \). In matrix form, the constraints read:

\[
\begin{bmatrix}
0 & 0 & 1 & 0 & -2 & \ldots \\
\end{bmatrix}
\begin{bmatrix}
v_1 \\
v_2 \\
\end{bmatrix}
= 3
\]

\[
\begin{bmatrix}
0 & 1 & -1 & 0 & 0 & \ldots \\
\end{bmatrix}
\begin{bmatrix}
v_1 \\
v_2 \\
\end{bmatrix}
= 0
\]
Application of BC on system

In order to impose the constraint on the system, we proceed as such: Multiplying both equations with the corresponding $\{v_i\}^T$ and summing up the constraints for the left-hand side we get,

$$[V] = \sum_{i=1}^{2} \{v_i\}^T \{v_i\}$$

and for the right-hand-side:

$$\{A\} = \sum_{i=1}^{2} \{v_i\}^T a_i$$

$$[K_{\text{pen}}] = [K] + [V] \cdot p$$

$$\{f_{\text{pen}}\} = \{f\} + \{A\} \cdot p$$

What changes, is that we also have a right-hand-side due to the non-homogeneous constraints!

Solution

Simply compute

$$\{u\} = [K_{\text{pen}}]^{-1} \{f_{\text{pen}}\}$$
Another method, just as general and simple to implement, is the Lagrange multiplier method: given the constraint matrix $[v]$ from the previous slide, set

$$[K_{Lagr}] = \begin{bmatrix} [K] & [v]^T \\ [v] & 0 \end{bmatrix}$$

$$[K_{Lagr}] = \begin{bmatrix} K_{11} & \ldots & K_{1N} & v_{11} & v_{21} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ K_{N1} & \ldots & K_{NN} & v_{1N} & v_{2N} \\ v_{11} & \ldots & v_{1N} & 0 & 0 \\ v_{21} & \ldots & v_{2N} & 0 & 0 \end{bmatrix}$$

Constraints at RHS

Instead of adding, we are again appending.

$$\{f_{Lagr}\} = \left\{ \begin{array}{c} \{f\} \\ \{P\} \end{array} \right\}$$

Where, $\{P\}$ is a vector containing the values of the non-homogeneous constraints. From the previous example, $\{P\} = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$. 
Comparisson

- The clear advantage of Lagr. Multiplier method over penalty is accuracy.
- Penalty has the advantage that does not introduce degrees of freedom to the linear system. However, this rarely becomes a real problem with modern hardware.
- Some linear system solvers rely on the matrix being positive definite for fast convergence - Lagrange multipliers may not work with some linear system solvers.
- Lagrange multiplier method provides directly the constraint forces (as the values corresponding to the rows and columns of the constraints).
- When constraints are linearly dependent (a constraint vector can be expressed as linear combination of other constraints in the same system) iterative solvers (i.e. GMRES) have to be used. Linear independence of constraints is hard to judge in some cases (however, there are linear algebra techniques one can perform to remedy that).
- The multi-freedom homogeneous case for the Master-slave elimination method, is described partially in the project description. This method is also exact (as Lagr. Multipliers).