Integration with Gaussian Quadrature

A convenient method for performing Gaussian integration in MATLAB is to construct a quadrature rule table. The quadrature rule table has `nsd` + 1 rows and `n` columns, where `nsd` is the number of spatial dimensions (1, 2, or 3) and `n` is the number of quadrature points. In one dimension, the quadrature table is populated as

$$\texttt{quad\_pts} = \begin{bmatrix} \xi_1 & \xi_2 & \dots & \xi_n \\ w_1 & w_2 & \dots & w_n \end{bmatrix}, \tag{1}$$

where ξ_i are the parent coordinates of the quadrature point within the element, and w_i are the quadrature weights. An example function that generates the one-dimensional quadrature rules for one to four points is given in Code 1. Expressions for higher order quadrature rules can be found at https://en.wikipedia.org/wiki/Gaussian_quadrature.

```
function [quad_pts] = quadrature(n)
  if n == 1
    quad_pts = [0; 2];
  elseif n == 2
    quad_pts = [[-1, 1]/sqrt(3); 1, 1];
  elseif n == 3
    quad_pts = [[1,0,-1]*sqrt(3/5); [5,8,5]/9];
  elseif n == 4
    x1 = sqrt(3/7 - 2/7*sqrt(6/5));
    x2 = sqrt(3/7 + 2/7*sqrt(6/5));
    w1 = (18+sqrt(30)) / 36;
    w2 = (18-sqrt(30)) / 36;
    quad_pts = [x1,-x1, x2, -x2; w1, w1, w2, w2];
  else
    error('Quadrature rule n must be 1, 2, 3, or 4');
  end
end
```

Code 1 – Function to return quadrature rule given the number of quadrature points.

Integration by Gaussian quadrature approximates the integral as the sum of weighted function values

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} f(x_i)w_i. \tag{2}$$

In the case that the integrand is a polynomial then integration by Gaussian quadrature will compute the integral exactly provided that the number of integration points is sufficient for the order of the polynomial. The one point integration rule can exactly up to linear polynomials. For each additional quadrature point, the highest degree of a polynomial that can be exactly integrated increases by two, e.g. up to quintic polynomials are exactly integrated with three integration points.

As an example, we consider the quartic function plotted in Fig. 1. Drawn below the function is a five element mesh. If each element is integrated with two quadrature points, then the integral will be numerically evaluated as a weighted sum of the function value evaluated at 10 points. As shown, the integration error is about 0.004%. With two integration points, the error of the integral is proportional to h^4, and thus by doubling the number of elements, the error decreases by a factor of 16.

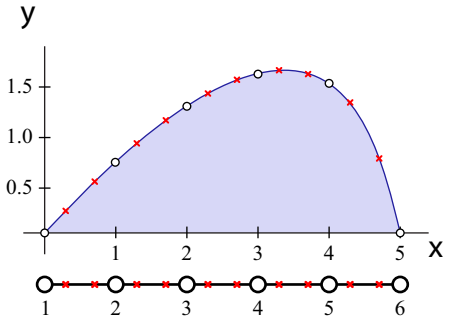

Figure 1 – Quadrature point for two point Gaussian quadrature of the function $f(x) = x - \frac{1}{5}x^2 + \frac{1}{20}x^3 - \frac{1}{100}x^4$.

Code 2 demonstrates how Gaussian quadrature is implemented in MATLAB. In the code example, the variables `ne` and `L` represent the number of elements and domain length, respectively. After defining a uniform one-dimensional mesh along the domain, the integral is computed element-by-element. Following the pattern of iterating over the columns of the connectivity matrix, the nodal coordinates of each element are gathered into `xe`. Next, we iterate over the columns of the quadrature table. In this loop, the variable $\mathsf{q} = [\xi_i \; w_i]^\mathsf{T}$ contains the quadrature point location and weight. The quadrature point location is represented in the element's parent coordinates, $-1 < \xi < 1$. We evaluate the shape functions at the location of the quadrature point and compute the global coordinate of the quadrature point by $x = \sum x_I^e N_I$. From the global point, we evaluate the function we want to integrate, and add its weighted value into the integral sum.

```
mesh.x = linspace(0, L, ne+1);
mesh.conn = [1:ne; 2:ne+1];
result = 0;
for c = mesh.conn
  xe = mesh.x(:,e);             % Nodal coordinates of element.
  le = xe(:,2) - xe(:,1);       % Element length.
  for q = quadrature(2)         % Call to function to generate quadrature table.
    N = 0.5*[1-q(1); 1+q(1)];   % Evaluates shape functions at the quadrature point.
    x = xe*N;
    f = x - 0.2*x^2 + 0.05*x^3 - 0.01*x^4;
    result = result + f*(le/2)*q(2);
  end
end
```

Code 2 – Code fragment to numerically integrate the quartic function.