Numerical Algorithms

Winter semester 2018/2019
Prof. Dr. Marc Alexander Schweitzer
Denis Duesseldorf

universitätbonn

Exercise sheet 7.

Submission on **Thursday, 2017-12-04, after lecture**.

Exercise 22. (Integrated Legendre-polynomials)

Recall that the Legendre-polynomials L_n, which are orthogonal with respect to $\omega \equiv 1$ on $(-1,1)$ can be given by

$$L_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}\left((x^2-1)^n\right) \ , \quad (n+2)L_{n+2}(x) - (2n+3)xL_{n+1}(x) + (n+1)L_n(x) = 0 \ .$$

The integrated Legendre-polynomials are given by

$$\hat{L}_n(x) = \int_{-1}^{x} L_{n-1}(s)\mathrm{d}s \ , \qquad n \geq 1 \ ,$$

the scaled integrated Legendre-polynomials by

$$\hat{K}_n(x) = (-1)^n \gamma_n \int_{-1}^{x} L_{n-1}(s)\mathrm{d}s \ , \qquad n \geq 2$$

with $\gamma_i = \sqrt{\frac{(2i-3)(2i-1)(2i+1)}{4}}$ and

$$\hat{K}_0(x) = \frac{1-x}{2} \ , \qquad \hat{K}_1(x) = \frac{1+x}{2} \ .$$

a) Show the relation

$$\hat{L}_n(x) = \frac{1}{2n-1}\left(L_n(x) - L_{n-2}(x)\right) \ , \qquad \forall n \geq 2 \ .$$

b) Show

$$\hat{L}_n(\pm 1) = 0 \ , \qquad \forall n \geq 2 \ .$$

c) Show the Orthogonality:

$$\int_{-1}^{1} \hat{L}_n(x)\hat{L}_m(x)\mathrm{d}x = 0 \ , \qquad |n-m| \notin \{0,2\} \ .$$

d) Sketch/Plot the first five integrated Legendre-Polynomials $\hat{L}_n(x)$ and the first five scaled integrated Legendre-Polynomials $\hat{K}_n(x)$ on the interval $[-1,1]$. For an efficient evaluation of the polynomials use their recursion formulas.

(4 Points)

Exercise 23. (1D assembly of integrated Legendre-polynomials)

Recall the scaled, integrated Legendre-polynomials $\hat{K}_n$ from the previous exercise.

a) Compute the mass matrix entries

$$\int_{-1}^{1} \hat{K}_n(x)\hat{K}_m(x)dx\ .$$

b) Compute the stiffness matrix entries

$$\int_{-1}^{1} \nabla\hat{K}_n(x)\nabla\hat{K}_m(x)dx\ .$$

c) Compute and assemble the mass and stiffness matrices for $p = 5$, i.e. using the $\hat{K}_n$, $n = \{0, \ldots, p\}$ as shape functions.

(3 Points)

Exercise 24. (Gauß-Lobatto quadratue)

In the usual case of piecewise linear FEM, numerical quadrature is done using only evaluations of the shape functions at the mesh-nodes. This type of quadrature - as you know - can yield to comparably large errors when used on higher order integrands. It is better to use e.g. the Gauss quadrature rules of order n, which are exact for integrands $f \in P_{2n+1}$. These quadrature-rules however use integration nodes that are almost all contained in the interior of the mesh-cells. This is very unconvenient, since in this case the values of the problem-dependent coefficients cannot be directly used as function evaluations for integration purposes.

One way of alleviating this are the Gauß-Radau and Gauß-Lobatto rules.

In the following again consider numerical integration

$$\sum_{i=0}^{n} w_i f(x_i) = Q(f) \approx I(f) = \int_{-1}^{1} f(x)\mathrm{d}x$$

with additional constraints of having certain x_i being pre-determined. For each of the following cases, determine, how the orthogonality condition from exercise 3 changes and, prove the given integration order and predetermined weights.

a) The asymmetric Gauß-Radau rules with $x_0 = -1$ yields

$$Q_R^-(f) = \frac{2}{n^2}f(-1) + \sum_{i=1}^{n} w_i^- f(x_i^-)\ ,$$

and taking $x_n = 1$ leads to

$$Q_R^+(f) = \sum_{i=0}^{n-1} w_i^+ f(x_i^+) + \frac{2}{n^2}f(1).$$

Show that both quadrature rules are exact for $f \in P_{2n}$. Moreover show that $x_i^- = -x_{n-i}^+$ and $w_i^- = w_{n-i}^+$.

b) The symmetric Gauß-Lobatto rules with $x_0 = -1, x_n = 1$ read

$$Q_L(f) = \frac{2}{n(n-1)}\Big[f(-1) + f(1)\Big] + \sum_{i=1}^{n-1} w_i f(x_i)\ .$$

Show that this rule is exact for $f \in P_{2n-1}$ and that $x_i = -x_{n-i}, w_i = w_{n-i}$.

(3 Points)

Exercise 25. (The Duffy transform)

As we have already seen, quadrature rules and construction of shape functions are more complicated for triangles than quadrilaterals since no Tensor product approach can be applied (i.e. no splitting of the multidimensional integration into a sequence of one-dimensional integrations is possible). The so called *Duffy* trick/transform is a way to bypass this.
Consider the reference triangle T and the quadrilateral Q

$$T = \{(x,y) \in \mathbb{R}^2 | 0 \leq x, y \leq 1, x + y \leq 1\} = ((0,0),(1,0),(0,1)) \ ,$$
$$Q = \{(\xi,\eta) \in \mathbb{R}^2 | -1 \leq \xi, \eta \leq 1\} = ((-1,-1),(1,-1),(1,1),(-1,1)) \ .$$

The nodal shape functions and barycentric coordinates on T are given as

$$\lambda_0 = 1 - x - y \ , \qquad \lambda_1 = x \ , \qquad \lambda_2 = y \ .$$

Consider the Duffy transform

$$D : Q \to T \ , \qquad (\xi,\eta) \mapsto (\frac{1}{4}(1+\xi)(1-\eta), \frac{1}{2}(1+\eta)) \ .$$

a) Compute the inverse D^{-1} and the Jacobian $|\det(\nabla D^{-1})|$ of the transform.

b) Compute the inverse map D^{-1} in terms of the barycentric coordinates of T.

c) How does a 2D tensor product quadrature rule

$$Q^2(f) = \sum_{i=0}^{n} \sum_{j=0}^{n} w_i w_j f(x_i, x_i)$$

of some 1D quadrature rule $\int_{-1}^{1} f(x) \approx \sum_{i=0}^{n} w_i f(x_i)$ translate to T via D?

(2 Points)

Programming exercise 8. (Red-Green refinement)

For many problems the initial mesh of the domain Ω might need to get partially refined after initial mesh creation. For example, some error estimator might warrant the partial refinement near a singularity of the solution, e.g. near a so called re-entrant corner. However, special care must be taken to ensure that the resulting mesh is still an admissible triangulation, e.g. no hanging nodes etc.
One strategy is so called red-green refinement based on bisection of edges.

- A triangle with vertices (a_0, a_1, a_2) will be red-refined by splitting it into the 4 congruent triangles

$$\begin{aligned}&(a_0, \frac{1}{2}(a_0 + a_1), \frac{1}{2}(a_0 + a_2)) ,(a_1, \frac{1}{2}(a_1 + a_2), \frac{1}{2}(a_0 + a_1))\\&(a_2, \frac{1}{2}(a_0 + a_2), \frac{1}{2}(a_1 + a_2)) ,(\frac{1}{2}(a_0 + a_1), \frac{1}{2}(a_1 + a_2), \frac{1}{2}(a_0 + a_2)) .\end{aligned}$$

  The shape of the triangles does not change.

- A triangle (a_0, a_1, a_2) with a hanging node on a single edge, say $\frac{1}{2}(a_1 + a_2)$ is green-refined by splitting it into the 2 triangles

$$(a_0, a_1, \frac{1}{2}(a_1 + a_2)) , (a_2, a_0, \frac{1}{2}(a_1 + a_2))$$

  The new edge is a so called green edge. The new triangles have a different shape.

These are combined in the following algorithm.

1. If there has already been a red-green refinement done before, remove all green edges from the mesh.
2. Red-refine all triangles marked for refinement, e.g. by some error estimator.
3. Red-refine all triangles with 2 or more hanging nodes existing on their edges. Repeat this step until all triangles have at most 1 hanging node on their edges.
4. Green-refine all remaining triangles with 1 hanging node. Mark the new edge green and save for later removal.

This procedure can be put into code in the following steps.

a) Write a function to red refine a predetermined triangle by appending new vertices to existing ones and replacing the triangle by its 4 congruent children.

b) Similarly, write a function to green refine a predetermined triangle.

c) Combine a) and b) to implement the red-green refinement of a mesh.

d) Create a 2-level refinement of the mesh of the L-shaped domain $(-1, 1)^2 \setminus (0, 1)^2$ from programming exercise 5a). First red-green refine with all triangles in $(-0.5, 0.5)^2$ marked for refinement in a first level and all triangles inside $(-0.25, 0.25)^2$ in a second level.

Hint: It is easier to create a new list and copy over elements not marked for deletion than to delete elements from a list while iterating over it.

(4 Points)

Send to `duesseldorf@ins.uni-bonn.de`