T U T O R I A L

"Numerical Methods for Solving Partial Differential Equations"

to the Lectures on NuPDE

T III Monday, 12 November 2007 (Time: 08:30 – 10:00, Room: T 212)

1.3 FEM for BVPs for second-order ODEs

Let $\Omega = (0,\,1)$, $\Gamma = \partial\Omega = \{0,\,1\} = \Gamma_D \cup \Gamma_R$ with $\Gamma_D \cap \Gamma_R = \emptyset$.
Consider the one-dimensional boundary value problem: Find $u(x)$ such that

$$\begin{aligned} -u''(x) &= f(x) && x \in \Omega\,, \\ u(x) &= g_D(x) && x \in \Gamma_D\,, \\ u'(x) &= \alpha(x)(g_R(x) - u(x)) && x \in \Gamma_R\,. \end{aligned} \tag{1.21}$$

We discretize this problem using the finite element method with Courant elements.
We consider the nodes $0 = x_0 < x_1 < \cdots < x_{N_h - 1} = 1$ which define a mesh (grid) $\mathcal{T}_h$ of Ω with the subintervals $T_k = (x_{k-1},\, x_k)$, $k = 1, \ldots, N_h$. We introduce the finite element space

$$V_h := \{v_h \in \mathcal{C}(\overline{\Omega}) : v_h|_{T_K} \in \mathcal{P}_1 \text{ for all } T_k \in \mathcal{T}_h\}$$

whose basis is given by the nodal (hat) functions φ_i, $i = 0, \ldots, N_h$, with

$$\varphi_i(x_j) = \delta_{ij} \quad \text{for } i,\, j = 0, \ldots, N_h\,.$$

In the following exercises we start to develop a program that will allow us to compute the finite element approximation u_h of the solution u of (1.21).

Recommended programming language: C^{++}
Also allowed: C, Fortran, Java

We denote the input parameter of a function by ↓ and output parameters by ↑.

13 Write a function `ElementStiffnessMatrix(↓xa, ↓xb, ↑element_matrix)` which for given `xa`$=x_{k-1}$ and `xb`$=x_k$ returns the 2×2 local stiffness matrix `element_matrix`$=K_h^{(k)}$ on the element T_k, i. e.,

$$K_h^{(k)} = \begin{bmatrix} \displaystyle\int_{T_k} (\varphi'_{k-1}(x))^2\,dx & \displaystyle\int_{T_k} \varphi'_{k-1}(x)\,\varphi'_k(x)\,dx \\ \displaystyle\int_{T_k} \varphi'_k(x)\,\varphi'_{k-1}(x)\,dx & \displaystyle\int_{T_k} (\varphi'_k(x))^2\,dx \end{bmatrix}.$$

14 Write a function `ElementLoadVector(↓(*f)(x), ↓xa, ↓xb, ↑element_vector)` which for a given function `f` $= f \in \mathcal{C}([0,\,1] \to \mathbb{R})$ and `xa`$=x_{k-1}$ and `xb`$=x_k$ returns the 2-dimensional local load vector `element_vector` $= f_h^{(k)}$ on the element T_k,

i. e.,

$$f_h^{(k)} = \begin{pmatrix} \displaystyle\int_{T_k} f(x)\,\varphi_{k-1}(x)\,dx \\ \displaystyle\int_{T_k} f(x)\,\varphi_k(x)\,dx \end{pmatrix}.$$

Use the trapezoidal rule to approximate above integrals:

$$\int_a^b g(x)\,dx \simeq \frac{b-a}{2}\big[g(a)+g(b)\big].$$

15 Define a data type `Mesh` which contains all the information on the mesh $\mathcal{T}_h$ – see also the lecture notes!

Hint: use `class` *in C++, or* `struct` *in C.*

16 Define an efficient data type `Matrix` for the sparse stiffness matrix K_h exploiting the fact that K_h is tridiagonal.

Hint: use `class` *or* `struct`.

Consider now $\Gamma_D = \emptyset$, $\Gamma_R = \{0, 1\}$ and $\alpha(x) = 0$ (pure homogeneous Neumann boundary conditions).

17 Write a function `AssembleStiffnessMatrix(↓mesh, ↑matrix)` that assembles the global $(N_h + 1) \times (N_h + 1)$ stiffness matrix `matrix` $= K_h$ for a given subdivision `mesh` $= \mathcal{T}_h$ of Ω.

Hint: Set $K_h = 0$, then start with $K_h^{(0)}$ and loop over all elements T_k to update the matrix K_h. On each element T_k, use the function `ElementStiffnessMatrix` *to compute $K_h^{(k)}$ and pay attention to put the entries of $K_h^{(k)}$ at the correct positions in K_h.*

18 Write a function `AssembleLoadVector(↓(*f)(x), ↓mesh, ↑vector)` that assembles the global $(N_h + 1)$-dimensional load vector `vector` $= \underline{f}_h$ for a given mesh $\mathcal{T}_h$ of Ω.

Hint: Set $\underline{f}_h = 0$, then start with $\underline{f}_h^{(0)}$ and loop over all elements T_k to update the vector $\underline{f}_h$. On each element T_k, use the function `ElementLoadVector` *to compute $\underline{f}_h^{(k)}$ and pay attention to add the entries in the right place.*

Test the implemented data types and functions using some simple examples, e. g., consider equidistant nodes x_i for different values of N_h, and simple functions $f(x) = 1$, $f(x) = x$, etc.