Finite Element Method in 1D

02623 - The Finite Element Method for Partial Differential Equations

Authors: Allan Peter Engsig-Karup

Scientific Computing Section

DTU Compute

Department of Applied Mathematics and Computer Science

Problem definition for model problem

Consider the following Boundary Value Problem (BVP).

Seek an unknown solution $u(x)$ to an Ordinary Differential Equation (ODE) in one space-dimension of the form

$$u'' + f(x)u' + g(x)u = q(x,u), \quad 0 \leq x \leq L$$

where $f(x), g(x)$ and $q(x,u)$ are known functions. Impose boundary conditions to be able to uniquely determine the solution

$$au(0) + bu'(0) = 0$$
$$cu(L) + du'(L) = 0$$

where a, b, c, d are defined for specific problems.

Defining a numerical model

Assume we have a mathematical problem which describes some physical system in terms of a BVP.

To create a numerical model we need to make two fundamental choices

- How to represent the solution?
- How to satisfy the differential equation of interest?

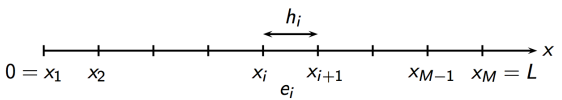
By making suitable choices the outcome is a numerical model formulated in terms of a discrete set of coupled algebraic equations that will need to be solved to constitute an approximate solution to the BVP.

Discretization

We want to solve an ODE on an interval $\bar{\Omega} = 0 \leq x \leq L$.

Discretization of the domain is done by introducing a set of M nodes (or vertices)

$$x_1 = 0, \quad x_{i+1} = x_i + h_i, \quad i = 1, ..., M-1$$



The length of each subinterval $I_i = [x_i, x_{i+1}]$ is defined as

$$h_i = x_{i+1} - x_i$$

We call each subinterval an element $e_i, i = 1, 2, ..., M-1$ and these form a partition of the closed interval $\bar{\Omega}$

$$\bar{\Omega} = \bigcup_{i=0}^{M-1} I_i$$

Finite element basis functions

A way to represent a function on $\bar{\Omega}$ is needed.

Consider the vector space of continuous piecewise linear functions

$$P_h^1 = \{v_h \in \mathcal{C}^0(\bar{\Omega}) | \forall i \in \{1, ..., M\}, v_{h|I_i} \in \mathbb{P}_1\}$$

defined in terms of the chosen partition.

We seek to represent a function $u(x) \in P_h^1$ by a piecewise linear continuous polynomial function of the form

$$\hat{u}(x) = \sum_{i=1}^{M} \hat{u}_i N_i(x), \quad 0 \leq x \leq L$$

where the coefficients $\hat{u}_i$ are weights to each of the global basis functions $N_i(x) \in P_h^1$, $i = 1, 2, ..., M$ defined on the entire domain $\bar{\Omega}$ of interest.

Finite element basis functions

For every node x_i we define a global finite element basis function $N_i(x)$ which is linear on each element and assumed to be continuous on the entire interval $0 \leq x \leq L$.

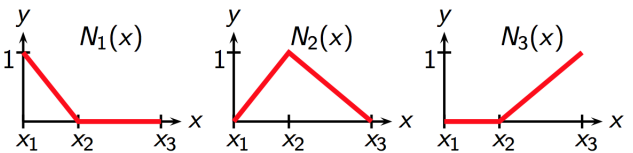
For each node in the mesh we require

$$N_i(x_j) = \begin{cases} 1 & , i = j \\ 0 & , i \neq j \end{cases} \quad , \quad i, j = 1, ..., M$$

The set $\{N_1, N_2, ..., N_M\}$ forms a basis for P_h^1.

Finite element basis functions

Assume $M = 3$. The three resulting global basis functions that fulfil these requirements can be illustrated

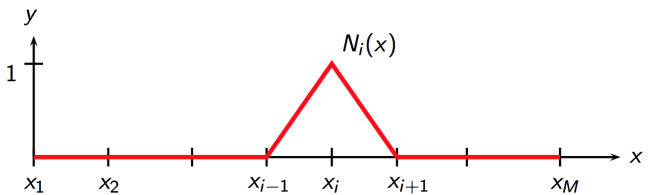


Important properties for every basis function $N_i(x)$, $i = 1, ..., M$;

- each one is nonzero on at most two elements.
- each one is a global function, i.e. defined on the entire domain $0 \leq x \leq L$.

Finite element basis functions

The typical finite element basis function for a mesh with M points can be sketched as



These functions are often referred to as a "hat functions".

The support of a finite element basis function is the region where it is nonzero. Since the functions is to be non-zero across only a few elements, it is said that it has compact (bounded) support across these elements.

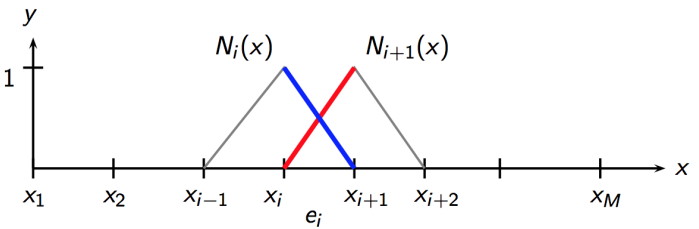
Finite element basis functions

The global finite element basis functions are defined analytically as

$$
\begin{aligned}
N_1(x) &= \begin{cases} 1 - \frac{x - x_1}{h_1} & , x_1 \leq x \leq x_2 \\ 0 & , \text{otherwise} \end{cases} \\
N_i(x) &= \begin{cases} \frac{x - x_{i-1}}{h_{i-1}} & , x_{i-1} \leq x \leq x_i \\ 1 - \frac{x - x_i}{h_i} & , x_i \leq x \leq x_{i+1} \\ 0 & , \text{otherwise} \end{cases} \qquad , i = 2, 3, ..., M-1 \\
N_M(x) &= \begin{cases} \frac{x - x_{M-1}}{h_{M-1}} & , x_{M-1} \leq x \leq x_M \\ 0 & , \text{otherwise} \end{cases}
\end{aligned}
$$

Note how every global basis function by construction are intimately connected with the finite element mesh nodes.

Finite element basis functions



On each element e_i the only two nonzero global finite element functions are $N_i(x)$ and $N_{i+1}(x)$.

Finite element basis functions

Since $N_i(x)$ is nonzero on at most two elements, it follows that we can define two local finite element basis functions for each element $e_i, i = 1, 2, ..., M-1$ as follows

$$
\begin{aligned}
N_1^{(i)}(x) &= N_i(x) = 1 - \frac{x - x_i}{h_i}, \quad x_i \leq x \leq x_{i+1} \\
N_2^{(i)}(x) &= N_{i+1}(x) = \frac{x - x_i}{h_i}, \quad x_i \leq x \leq x_{i+1}
\end{aligned}
$$

The local basis functions are only defined within a single element.

The local basis functions are of central importance, since these functions makes it possible to reduce calculations to individual elements.

Interpolation

The finite element basis functions can be used for interpolation.

Let there be given a function $u(x) \in \mathcal{C}^0(\bar{\Omega})$ defined on the interval $0 \leq x \leq L$.

An interpolating function $u_I(x) \in P_h^1$ can then be defined as

$$u_I(x) = \sum_{i=1}^{M} u(x_i) N_i(x), \quad 0 \leq x \leq L$$

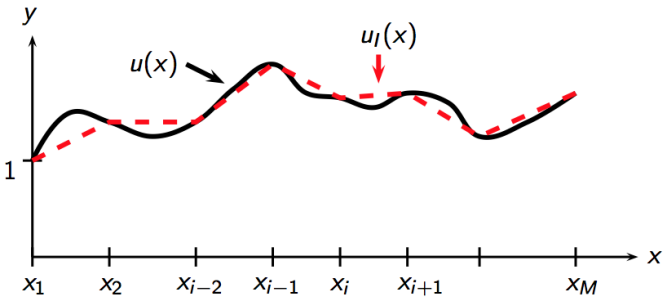
where $N_i(x)$ is a global finite element basis function.
It follows from the property of the basis functions that

$$u_I(x_i) = u(x_i), \quad i = 1, 2, .., M$$

Thus, $u_I(x)$ interpolates $u(x)$ at the nodes.

Since each $N_i(x)$ is a continuous, piecewise linear function, so is this interpolating function.

Interpolation



Function: $u(x)$ (solid line)

Finite element interpolant: $u_I(x)$ (dashed line)

Interpolation

The interpolation error is defined as

$$\epsilon(x) = u_I(x) - u(x).$$

It is possible to show that if $u(x) \in \mathcal{C}^2(\bar{\Omega})$, i.e. the function is at least twice differentiable and derivatives are continuous, then for $i = 1, 2, ..., M-1$ the local errors are bounded asymptotically

$$\max_{x \in e_i} |\epsilon(x)| \leq \frac{1}{8} h_i^2 \max_{x \in e_i} |u''(x)|.$$

From these local bounds a global error bound can be determined as

$$\max_{0 \leq x \leq L} |\epsilon(x)| \leq \frac{1}{8} h^2 \max_{0 \leq x \leq L} |u''(x)|,$$

where the largest element length h is determined as

$$h = \max_{i=0,1,...,M-1} h_i$$

Remark: These results highlight the need for smaller elements where solution i) change rapidly and if ii) higher accuracy is needed.

Interpolation

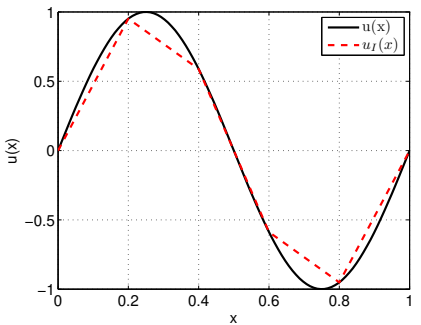
Given a set of value $w_1, w_2, ..., w_M$ we can construct a function $w(x)$ on the interval $0 \leq x \leq L$ which interpolates the values exactly, i.e.

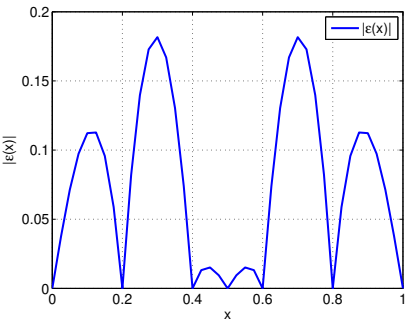
$$w(x_i) = w_i, \quad i = 1, 2, ..., M.$$

This can be achieved by taking $w(x) \in P_h^1(\bar{\Omega})$ as the continuous piecewise linear function

$$w(x) = \sum_{i=1}^{M} w_i N_i(x), \quad 0 \leq x \leq L.$$

Interpolation





Remark: It is always good practice to check if the expected asymptotic convergence rate is achieved.

A simple model problem

Consider the Boundary Value Problem (BVP) defined in terms of the second order differential equation

$$u'' - u = 0, \quad 0 \leq x \leq L$$

with Boundary Conditions (BC)

$$u(0) = c, \quad u(L) = d$$

The exact solution to this problem is

$$u(x) = c_1 e^x + c_2 e^{-x}, \quad 0 \leq x \leq L$$

with the coefficients c_1 and c_2 determined by the specified BCs.

Obviously in the classical sense, we must have $u \in \mathcal{C}^2(\bar{\Omega})$.

The Finite Element Method

To illustrate the FEM methods, we seek an approximation $\hat{u}(x) \approx u(x)$ which is a continuous piece-wise linear function.

$$\hat{u}(x) = \sum_{i=1}^{M} \hat{u}_i N_i(x), \quad 0 \leq x \leq L$$

Such a function in general do not have a continuous slope. Thus, we need to reformulate the BVP to reduce the continuity requirements by removing the second order derivative from the BVP. To formulate a FEM this can be done using Galerkin's method.

Multiply both sides of BVP equation by a sufficiently smooth function $v(x)$ that satisfies the homogenous BCs

$$v(0) = v(L) = 0,$$

and apply integration by parts over the interval $0 \leq x \leq L$.

The Finite Element Method

Reformulate the BVP by applying integration by parts over the interval $0 \leq x \leq L$

$$\int_0^L (u'' - u)v dx = 0$$

$$\Leftrightarrow \quad [u'v]_0^L - \int_0^L u'v' dx - \int_0^L uv dx = 0$$

and using that $v(0) = v(L) = 0$ we find

$$\Rightarrow \quad \int_0^L (u'v' + uv)\, dx = 0$$

This reformulated problem is the weak formulation of the BVP and can be used to generate necessary conditions that makes it possible to determine unknown coefficients in the finite series expansion of $\hat{u}$.

The Finite Element Method

This establishes that the solution of the BVP with BCs is also a solution of the problem of finding a function $u(x)$ that

- satisfies the weak formulation for all sufficiently smooth functions $v(x)$ with property $v(0) = v(L) = 0$.
- satisfies the BCs.

The Finite Element Method

Now, let's apply Galerkin's Method to formulate a FEM based on the weak formulation.

Seek a function $\hat{u}(x) \in P_h^1(\bar{\Omega})$ that approximates (i.e. represents) the exact solution $u(x) \in \mathcal{C}^2(\bar{\Omega})$ of the BVP in the form (M=3)

$$\hat{u}(x) = \hat{u}_1 N_1(x) + \hat{u}_2 N_2(x) + \hat{u}_3 N_3(x), \quad 0 \leq x \leq L.$$

Determine the unknown coefficients $\hat{u}_1$, $\hat{u}_2$ and $\hat{u}_3$. Require

$$\hat{u}(0) = c, \quad \hat{u}(L) = d.$$

Thus, if these BCs are satisfied then

$$\hat{u}(0) = \hat{u}_1 N_1(0) + \hat{u}_2 N_2(0) + \hat{u}_3 N_3(0) = \hat{u}_1,$$
$$\hat{u}(L) = \hat{u}_1 N_1(L) + \hat{u}_2 N_2(L) + \hat{u}_3 N_3(L) = \hat{u}_3.$$

Consequently

$$\hat{u}_1 = c, \quad \hat{u}_3 = d.$$

To determine the last coefficient $\hat{u}_2$ we substitute our approximation $\hat{u}(x)$ for $u(x)$ in the weak formulation of the BVP and put $v(x) = N_2(x)$

$$\int_0^L (\hat{u}'v' + \hat{u}v)\, dx = 0$$

Note that $N_2(x)$ vanishes at the endpoints of the interval as required.

The Finite Element Method

Using the definition for $\hat{u}(x)$, having changed the order of the factors of the integrands and by collecting terms we find

$$\hat{u}_1 \int_0^L \left(v' N_1' + v N_1\right) dx + \hat{u}_2 \int_0^L \left(v' N_2' + v N_2\right) dx + \hat{u}_3 \int_0^L \left(v' N_3' + v N_3\right) dx = 0,$$

and, by substituting $v(x) = N_2(x)$ into the factors of the ingrands

$$\hat{u}_1 \int_0^L \left(N_2' N_1' + N_2 N_1\right) dx + \hat{u}_2 \int_0^L \left(N_2' N_2' + N_2 N_2\right) dx + \hat{u}_3 \int_0^L \left(N_2' N_3' + N_2 N_3\right) dx = 0,$$

Remark: Selecting the test functions $v(x)$ that fulfils the assumptions made in deriving the weak formulation from the set of (linearly independent) global finite element basis functions $\{N_i(x)\}_{i=1}^M$ is what defines Galerkin's method.

The Finite Element Method

The last equation can be written in compact form as

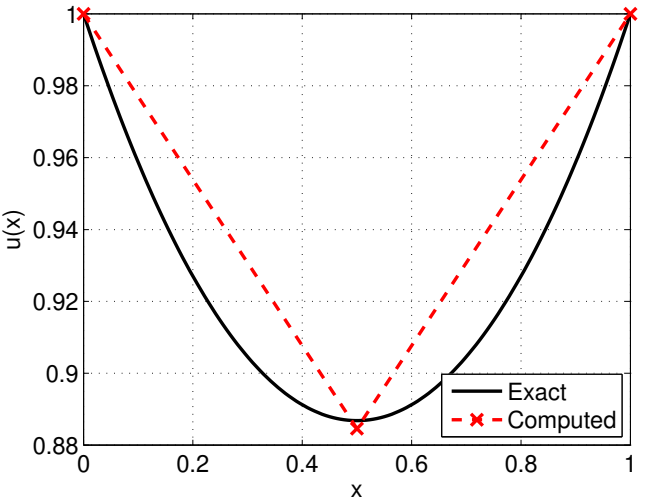
$$a_{2,1}\hat{u}_1 + a_{2,2}\hat{u}_2 + a_{2,3}\hat{u}_3 = 0,$$

where

$$\begin{aligned} a_{2,1} &= \int_0^L (N_2' N_1' + N_2 N_1)\, dx \\ a_{2,2} &= \int_0^L (N_2' N_2' + N_2 N_2)\, dx \\ a_{2,3} &= \int_0^L (N_2' N_3' + N_2 N_3)\, dx \end{aligned}$$

By computing these coefficients it is possible to solve for $\hat{u}_2$.

The Finite Element Method



M=3

The Finite Element Method

Now, let's instead consider a general mesh on $x \in [0, L]$ with M mesh points. Then, the approximate solution $\hat{u}(x)$ has the form

$$\hat{u}(x) = \sum_{j=1}^{M} \hat{u}_j N_j(x)$$

Again, we impose the BCs and immediately find

$$\hat{u}(0) = \hat{u}_1 = c, \quad \hat{u}(L) = \hat{u}_M = d.$$

The Finite Element Method

To find the remaining $(M-2)$ coefficients, we substitute $\hat{u}(x)$ for $u(x)$ i the weak formulation and put $v(x)$ equal to $N_i(x)$, $i = 2, 3, ..., M-1$.

Note: Each $N_i(x)$ satisfy the requirements of $v(x)$.

From this process $(M-2)$ equations are generated

$$\sum_{j=1}^{M} a_{i,j}\hat{u}_j = 0, \quad i = 2, 3, ..., M-1$$

where

$$a_{i,j} = \int_0^L \left(N_i' N_j' + N_i N_j\right) dx$$

Note: $a_{i,j}$ is symmetric.

The Finite Element Method

Since $N_i(x)$ only have overlap with $N_j(x)$, $j = i-1, i, i+1$ the system of equations we have just found for the coefficients reduces to

$$a_{i,i-1}\hat{u}_{i-1} + a_{i,i}\hat{u}_i + a_{i,i+1}\hat{u}_{i+1} = 0, \quad i = 2, 3, ..., M-1,$$

This is a system of $(M-2)$ linear algebraic equations for $(M-2)$ unknowns where

$$\begin{aligned} a_{i,i-1} &= \int_{x_{i-1}}^{x_i} \left(N_i' N_{i-1}' + N_i N_{i-1} \right) dx \\ a_{i,i} &= \int_{x_{i-1}}^{x_i} \left(N_i'^2 + N_i^2 \right) dx + \int_{x_i}^{x_{i+1}} \left(N_i'^2 + N_i^2 \right) dx \\ a_{i,i+1} &= \int_{x_i}^{x_{i+1}} \left(N_i' N_{i+1}' + N_i N_{i+1} \right) dx \end{aligned}$$

Element matrices

The expression for $a_{i,i-1}$, $a_{i,i}$ and $a_{i,i+1}$ can be written in terms of the local finite element basis functions as

$$
\begin{aligned}
a_{i,i-1} &= \int_{x_{i-1}}^{x_i} \left[\left(N_2^{(i-1)} \right)' \left(N_1^{(i-1)} \right)' + N_2^{(i-1)} N_1^{(i-1)} \right] dx \\
a_{i,i} &= \int_{x_{i-1}}^{x_i} \left[\left(N_2^{(i-1)} \right)'^2 + \left(N_2^{(i-1)} \right)^2 \right] dx + \int_{x_i}^{x_{i+1}} \left[\left(N_1^{(i)} \right)'^2 + \left(N_1^{(i)} \right)^2 \right] dx \\
a_{i,i+1} &= \int_{x_i}^{x_{i+1}} \left[\left(N_1^{(i)} \right)' \left(N_2^{(i)} \right)' + N_1^{(i)} N_2^{(i)} \right] dx
\end{aligned}
$$

Associate with each element e_i an element matrix K_i defined by

$$
K_i = \begin{bmatrix} k_{1,1}^{(i)} & k_{1,2}^{(i)} \\ k_{2,1}^{(i)} & k_{2,2}^{(i)} \end{bmatrix},
$$

where

$$
k_{r,s}^{(i)} = \int_{x_i}^{x_{i+1}} \left[\left(N_r^{(i)} \right)' \left(N_s^{(i)} \right)' + N_r^{(i)} N_s^{(i)} \right] dx, \quad r, s = 1, 2.
$$

Element matrices

Using the analytical expressions for $N_r^{(i)}$ and $N_s^{(i)}$ we find

$$K_i = \begin{bmatrix} \left(\frac{1}{h_i} + \frac{h_i}{3}\right) & \left(-\frac{1}{h_i} + \frac{h_i}{6}\right) \\ \left(-\frac{1}{h_i} + \frac{h_i}{6}\right) & \left(\frac{1}{h_i} + \frac{h_i}{3}\right) \end{bmatrix},$$

Further, we observe that

$$a_{i,i-1} = k_{2,1}^{(i-1)}, \quad a_{i,i} = k_{2,2}^{(i-1)} + k_{1,1}^{(i)}, \quad a_{i,i+1} = k_{1,2}^{(i)}$$

Thus, for $M = 3$ we obtain

$$k_{2,1}^{(1)}\hat{u}_1 + \left(k_{2,2}^{(1)} + k_{1,1}^{(2)}\right)\hat{u}_2 + k_{1,2}^{(2)}\hat{u}_3 = 0$$

Element matrices

In the general case, the linear system can be expressed in matrix-vector form as illustrated for $M = 6$ by

$$\begin{bmatrix} k_{2,1}^{(1)} & \left(k_{2,2}^{(1)} + k_{1,1}^{(2)}\right) & k_{1,2}^{(2)} & & & \\ & k_{2,1}^{(2)} & \left(k_{2,2}^{(2)} + k_{1,1}^{(3)}\right) & k_{1,2}^{(3)} & & \\ & & k_{2,1}^{(3)} & \left(k_{2,2}^{(3)} + k_{1,1}^{(4)}\right) & k_{1,2}^{(4)} & \\ & & & k_{2,1}^{(4)} & \left(k_{2,2}^{(4)} + k_{1,1}^{(5)}\right) & k_{1,2}^{(5)} \end{bmatrix} \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \\ \hat{u}_4 \\ \hat{u}_5 \\ \hat{u}_6 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Remark, how this linear system holding four equations for $i = 2, .., M - 1$ can be constructed using the element matrices K_i using a global assembly procedure.

Element matrices

Since $\hat{u}_1 = c$ and $\hat{u}_6 = d$ some of the terms are known and can be moved to the right hand side

$$\begin{bmatrix} \left(k^{(1)}_{2,2} + k^{(2)}_{1,1}\right) & k^{(2)}_{1,2} & & & \\ k^{(2)}_{2,1} & \left(k^{(2)}_{2,2} + k^{(3)}_{1,1}\right) & k^{(3)}_{1,2} & & \\ & k^{(3)}_{2,1} & \left(k^{(3)}_{2,2} + k^{(4)}_{1,1}\right) & k^{(4)}_{1,2} & \\ & & k^{(4)}_{2,1} & \left(k^{(4)}_{2,2} + k^{(5)}_{1,1}\right) & \end{bmatrix} \begin{bmatrix} \hat{u}_2 \\ \hat{u}_3 \\ \hat{u}_4 \\ \hat{u}_5 \end{bmatrix} = \begin{bmatrix} -k^{(1)}_{2,1} c \\ 0 \\ 0 \\ -k^{(5)}_{1,2} d \end{bmatrix},$$

or include all equations in the system as

$$\begin{bmatrix} 1 & & & & & \\ & \left(k^{(1)}_{2,2} + k^{(2)}_{1,1}\right) & k^{(2)}_{1,2} & & & \\ & k^{(2)}_{2,1} & \left(k^{(2)}_{2,2} + k^{(3)}_{1,1}\right) & k^{(3)}_{1,2} & & \\ & & k^{(3)}_{2,1} & \left(k^{(3)}_{2,2} + k^{(4)}_{1,1}\right) & k^{(4)}_{1,2} & \\ & & & k^{(4)}_{2,1} & \left(k^{(4)}_{2,2} + k^{(5)}_{1,1}\right) & \\ & & & & & 1 \end{bmatrix} \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \\ \hat{u}_4 \\ \hat{u}_5 \\ \hat{u}_6 \end{bmatrix} = \begin{bmatrix} c \\ -k^{(1)}_{2,1} c \\ 0 \\ 0 \\ -k^{(5)}_{1,2} d \\ d \end{bmatrix},$$

In compact notation, the M'th-order system is given in the form

$$A\hat{\boldsymbol{u}} = \boldsymbol{b},$$

with the matrix A tridiagonal, symmetric and positive definite.

The computation

Tasks:

- Compute A and $\boldsymbol{b}$
- Solve $A\boldsymbol{u} = \boldsymbol{b}$
  `>> u = A \ b;`
- However, A is symmetric positive definite and tridiagonal.
- Can use sparse storage scheme and exploit symmetry property of A, e.g. by using Cholesky factorization (`help chol`).
  `>> [U]=chol(A);`
  `>> u=U'\ (U'\ b);`
- The computational work should be $\mathcal{O}(M)$ for the sparse Gaussian elimination and Cholesky methods.
  (Plot CPU time for solve step vs. M to show)

The computation

Algorithm 1 (Pseudo-Matlab)

```
% Allocate SPARSE storage for A and FULL for b
A = spalloc(M,M,M*3); b = zeros(M,1);

% Exploit symmetry, construct upper A only
for i = 1 : M-1
   % Compute k(r,s,i)'s from (1.26)
   A(i,i)     = A(i,i) + k(1,1,i);
   A(i,i+1)   = k(1,2,i);
   A(i+1,i+1) = k(2,2,i);
end
```

Algorithm 1 (Pseudo-Python)

```
# import numpy as np
# from scipy.sparse import lil_matrix

# Allocate SPARSE storage for A and FULL for b
A = lil_matrix((M, M))
b = np.zeros(M)

# Exploit symmetry, construct upper A only
for i in range(M - 1):  # Python is 0-indexed
   # Compute k(r, s, i)'s from (1.26)
   A[i, i]     = A[i, i] + k(1, 1, i)
   A[i, i + 1] = k(1, 2, i)
   A[i + 1, i + 1] = k(2, 2, i)
```

The computation

Algorithm 2 (Pseudo-Matlab)

```
% Corrections to A and b takes into account BCs (non-eliminated)
b(1)   = c;
b(2)   = - A(1,2)*c;
A(1,1) = 1;
A(1,2) = 0;
b(M)   = d;
b(M-1) = b(M-1) - A(M-1,M)*d;
A(M,M) = 1;
A(M-1,M) = 0;
```

Algorithm 2 (Pseudo-Python)

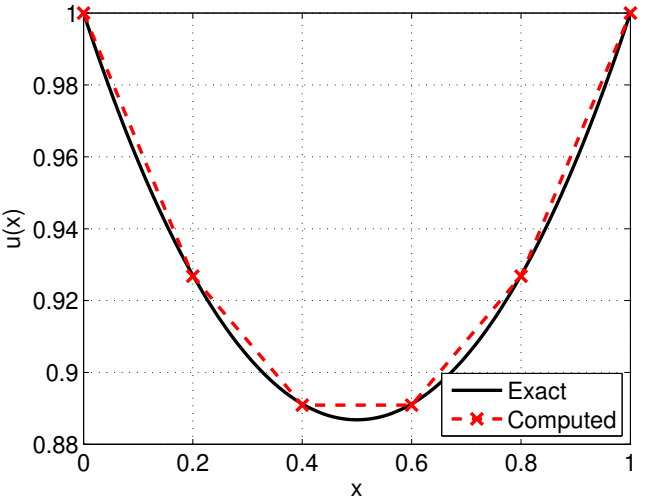
```
# Corrections to A and b takes into account BCs (non-eliminated)
b[0] = c
b[1] = -A[0, 1] * c
A[0, 0] = 1
A[0, 1] = 0
b[M - 1] = d
b[M - 2] = b[M - 2] - A[M - 2, M - 1] * d
A[M - 1, M - 1] = 1
A[M - 2, M - 1] = 0
```

Note that Algorithm 2 exploits symmetry in the element matrix in computing `b(2)`;

$$b_2 = -a_{2,1}c = -k^{(1)}_{2,1}c = -k^{(1)}_{1,2}c = -a_{1,2}c$$

The computation

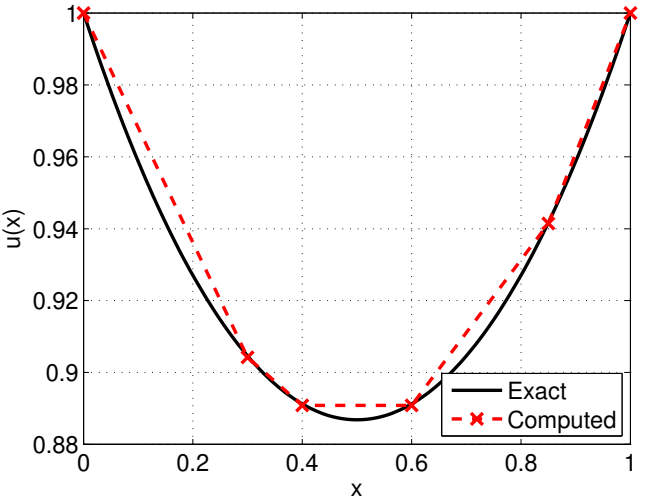
Uniform mesh:



M=6

The computation

Non-uniform mesh:



M=6

Error bounds

Let $u(x)$ be the exact solution to

$$u'' - u = 0, \quad 0 \leq x \leq L$$
$$u(0) = c, \quad u(L) = d$$

Let $\hat{u}(x)$ be a Finite Element approximation to the solution

$$\hat{u}(x) = \sum_{i=1}^{M} \hat{u}_i N_i(x)$$

Remark: $u_I(x) \neq \hat{u}(x)$!

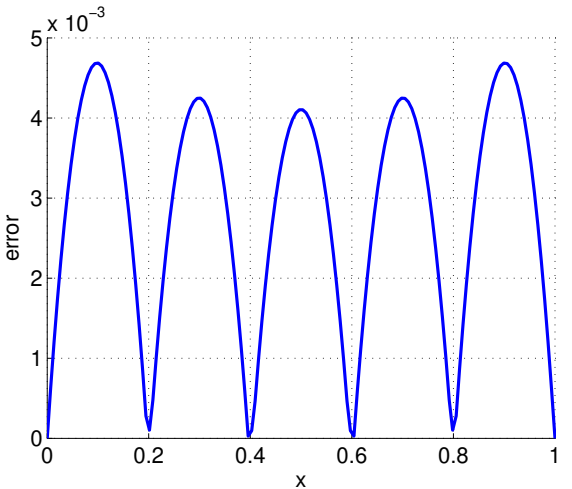
Then it is possible to show that for h sufficiently small

$$\max_{0 \leq x \leq L} |\hat{u}(x) - u(x)| \leq Ch^2$$

where h is a measure of the largest element in the mesh.

Error bounds

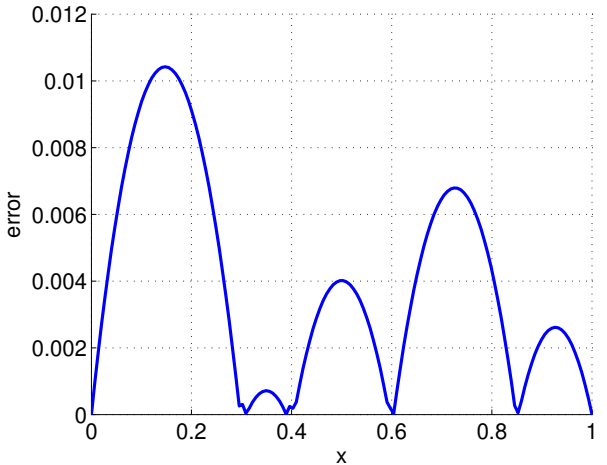
Uniform mesh: M=6



Error is bounded everywhere and at boundaries the solution is exact as expected.

Error bounds

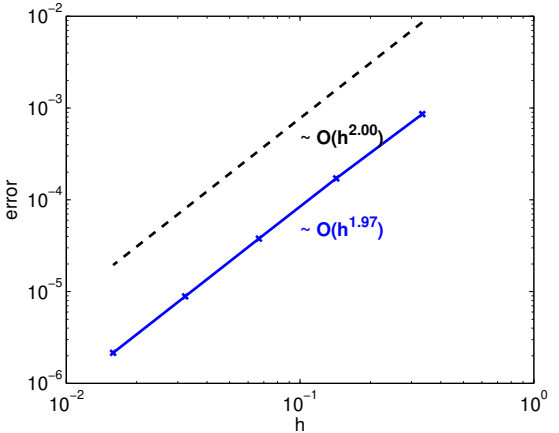
Non-uniform mesh: M=6



Notice differences in magnitude of error in regions where points are either close or far apart.

Error bounds

Uniform mesh: h-type refinement convergence test



Remark: This test is a mandatory verification of theory as well as the actual implementation. A mismatch means you are in troubles.

Error bounds

The function $\hat{u}(x)$ is in general different from the interpolating function $u_I(x)$. However, the given error bounds show that both functions are second-order approximations to the exact solution $u(x)$. In general, the interpolating property

$$|u_I(x_i) - u(x_i)| = 0, \quad i = 1, 2, ..., M$$

does not imply (!)

$$\max_{0 \le x \le L} |u_I(x) - u(x)| \le \max_{0 \le x \le L} |\hat{u}(x) - u(x)|$$

since the largest point-wise error is not guaranteed to be found on a node point. It might as well be in between nodes (cf. previous figures).

Error bounds

There are a number of ways of measuring the error function $(\hat{u}(x) - u(x))$. One is to consider the so-called infinity norm

$$||\hat{u} - u||_\infty \equiv \max_{0 \leq x \leq L} |\hat{u}(x) - u(x)|$$

which is what has been done in the derivation of the given global bounds. Another way is to look at the L_2-norm

$$||\hat{u} - u||_{L_2} \equiv \sqrt{\int_0^L \left[(\hat{u}(x) - u(x))^2 \right] dx}$$

or the energy norm (which is specific to the problem considered, here $u'' - u = 0$)

$$||\hat{u} - u||_E \equiv \sqrt{\int_0^L \left[(\hat{u}(x) - u(x))^2 + (\hat{u}'(x) - u'(x))^2 \right] dx}$$

Error bounds

It can be shown that the function $\hat{u}(x)$ produced by the Finite Element Method minimizes the energy norm of the error. More precisely, if $w(x)$ is a function of the form

$$w(x) = cN_1(x) + dN_M(x) + \sum_{i=2}^{M-1} w_i N_i(x)$$

where w_i, $i = 2, ..., M-1$ are arbitrary, then

$$||\hat{u} - u||_E \leq ||w - u||_E$$

which implies

$$||\hat{u} - u||_E \leq ||u_I - u||_E$$

For further discussions of error, e.g. see literature references in lecture notes.

Error bounds

Consider the earlier error bound given as

$$\max_{0 \leq x \leq L} |\hat{u}(x) - u(x)| \leq Ch^2$$

Some reasons to be interested in theoretical error bounds

- Establishes convergence of a numerical method to a given problem.
- Can be useful for error estimation for assessing the quality of the solution (for h sufficient small).
  (e.g. to guide adaptive mesh refinement algorithms)
- Can be used for comparing convergence rate of methods.
  (e.g. how rapid can we decrease errors in solution?)
- Practical estimates can be compared to theoretical error bounds for verification/debugging purposes! (DO THIS!)

Adaptive Mesh Refinement (AMR)

In Adaptive Mesh Refinement (AMR) the goal is to develop algorithms that can be used for automatically adapting the mesh to the (unknown) solution. The key components are

- Error estimation (or error indication)
- A criteria for selecting elements for refinement
- A refinement algorithm

Remark: In this course, you(!) are to derive the correct formula and implement these on your own for the code to be correct. Hence, the default rule is to not use pre-built functions in your choice of programming language, except for those that are prepared for linear algebra operations (fx. sparse matrices).

Adaptive Mesh Refinement (AMR)

In Exercise 1.6 it is proposed that the error estimation is based on the L_2 (mean-square) norm of the change in an approximate solution across elements under refinement. The change can be measured for the i'th element by the measure

$$\Delta \mathrm{err}_i = ||\Pi_{h/2,i} u - \Pi_{h,i} u||_{L_2(e_i)}, \quad ||f||_{L_2(\Omega)} \equiv \left(\int_\Omega |f|^2 dx \right)^{1/2}$$

In this context, the approximate solution based on interpolation $\Pi_{h,i} u$ of the true $u(x)$ on a given element of size h corresponds to approximating the solution using the interpolating functions as defined by (1.7) in the notes on, respectively, a coarse mesh (h) and a refined mesh $(h/2)$.

In Exercise 1.7 b) it is proposed to consider the change in the L_2 norm of the solution between a coarse and a fine grid when it is computed by FEM (rather than by interpolation).

Adaptive Mesh Refinement (AMR)

A criteria for selecting elements for refinement can be defined as

$$\Delta \text{err}_i > \alpha \cdot \texttt{tol}, \quad \forall i = 1, ..., M-1$$

where $\alpha \in \mathbb{R}$ controls how aggressive a refinement strategy we employ, e.g. with $\alpha = 1$ we only refine those elements which does not meet the acceptable tolerance level `tol`.

Another opportunity is to refine based on the idea of equi-distributing the errors by selecting some kind of fraction of the elements with the largest errors

$$\Delta \text{err}_i > \alpha \cdot \max_i \Delta \text{err}_i$$

where $\alpha < 1$. Feel free to experiment with the marking criteria to find ways to optimize the solution effort and results.

Adaptive Mesh Refinement (AMR)

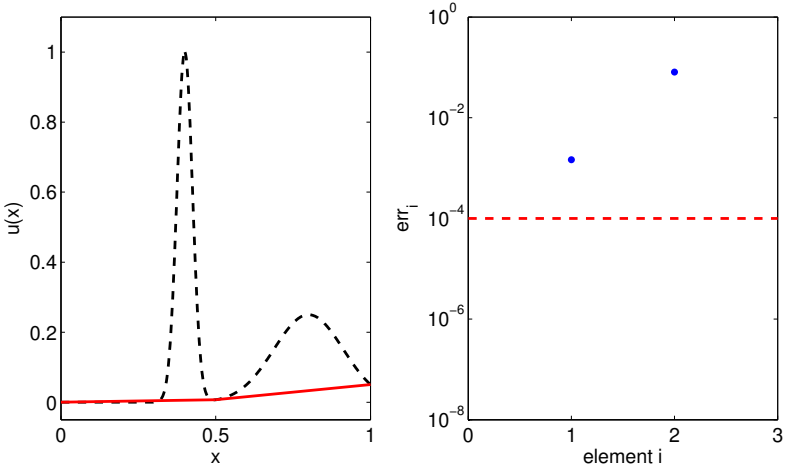
A refinement algorithm can be based on modifying the coarse grid tables `EToV` and `VX` by taking into account which elements have been marked for refinement.

The refinement can then be done using some subdivision algorithm, e.g. simple bisection where the element is in split in two new element of equal size and half of the size of the marked element.

Advice: In Exercises 1.6 and Exercise 1.7 focus on getting the solver to work in the simplest possible way (i.e. without fancy coding tricks). If time permits, try and optimize the solver to be more efficient (perhaps with fancy coding tricks).

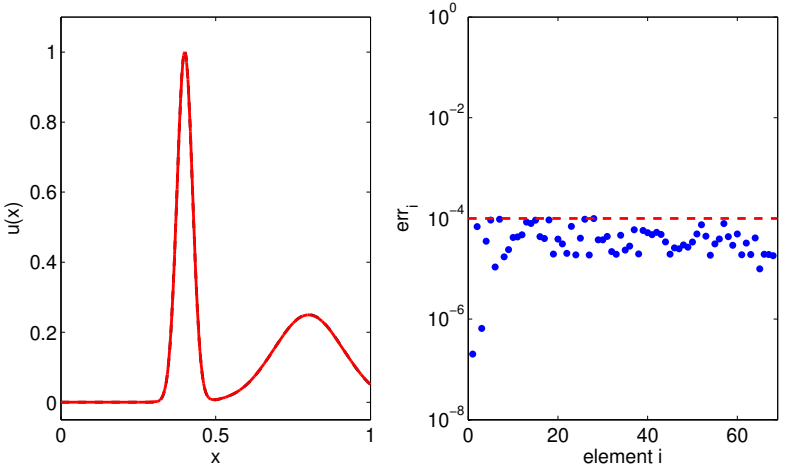
Make sure to compute the discrete L_2 norm efficiently by a suitable approximation derived from the definition of his norm, i.e. consistent with the formal order of the numerical scheme, of the continuous norm. Remark, do not use a function such as **`quad`** in Matlab as this is in general not efficient in this context.

Adaptive Mesh Refinement (AMR)



Initial condition: DOF = 3, Elements=2

Adaptive Mesh Refinement (AMR)



DOF = 69, Iterations = 9, tol=1e-4, Elements=68