Finite Element Method in 2D

02623 - The Finite Element Method for Partial Differential Equations

Author: Max Bitsch & Allan Peter Engsig-Karup

Scientific Computing Section

DTU Compute

Department of Applied Mathematics and Computer Science

Course content

- Teacher
  - Allan P. Engsig-Karup
    Associate Professor in Scientific Computing (DTU Compute)
    Building 303b, room 108, apek@dtu.dk.
- Teaching Assistant
  - Max Bitsch
    PhD Student (DTU Compute)
    Building 303b, maxbit@dtu.dk.

The following topics are covered in the course (cf. tentative course plan provided)

- Introduction to Finite Element Methods
- FEM in one space dimension
- Introduction to direct and iterative solvers of linear system of equations
- FEM in two space dimensions
- Introduction to Mesh generation

Note: Ask questions as needed!

Who am I?

- Industrial PhD at DTU Compute working with DHI
- Solving shallow water equations using a finite volume method.

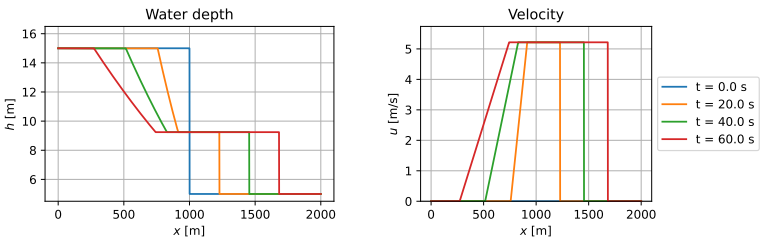


Figure: Multiple snapshots of the water depth and the velocity for a dam break problem.

The finite volume method uses piecewise polynomial basis functions, just like the finite element method, but is based on element centers rather than element vertices.

Real life application – MIKE 21 FM

- MIKE 21 FM is a 2D finite volume solver for solving the shallow water equations.
- Designed to handle real-world geometry with real world forcing.

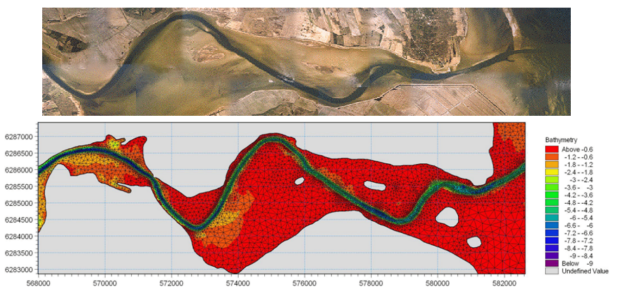


Figure: Picture of Mariager Estuary, Denmark and a computational mesh of the area.

The simulation software is used by engineers worldwide and is based on concepts from this course.

Introduction

Problems in physics and engineering often lead to mathematical problems involving Partial Differential Equations (PDE).

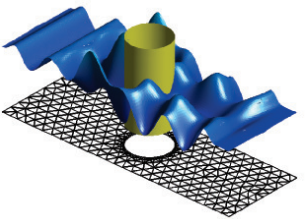


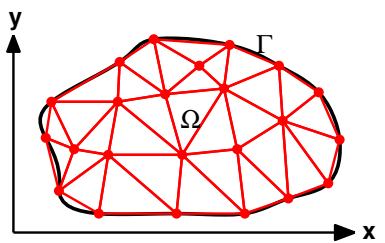
Figure: A snapshot of free water surface interacting with a cylindrical structure in a channel computed using a Discontinuous Galerkin Finite Element Method.

The classical Finite Element Method (FEM) is a numerical method which is well-suited for solving PDE problems with complex geometries in a systematic way.

Discretization

We want to solve a PDE in a domain $\bar{\Omega}$ in two space dimensions $(x, y) \in \mathbb{R}^2$ with interior Ω and boundary Γ.

Discretization of the domain is done by introducing a set of elements e_n, $n = 1, 2, ..., N$ such that $\bar{\Omega} = \cup_{n=1}^{N} e_n$ and nodes (vertices) by (x_i, y_i) with global node numbers $i = 1, 2, ..., M$.



Note that the boundary of the finite element mesh cannot coincide with a curved Γ.

Finite element basis functions

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

Consider the vector space of continuous piecewise linear functions

$$P_h^1 = \{v_h \in \mathcal{C}^0(\bar{\Omega}) | \forall n \in \{1, ..., N\}, v_h|_{e_n} \in \mathbb{P}_1\}$$

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

$$\hat{u}(x) = \sum_{i=1}^{M} \hat{u}_i N_i(x, y), \quad (x, y) \in \bar{\Omega}$$

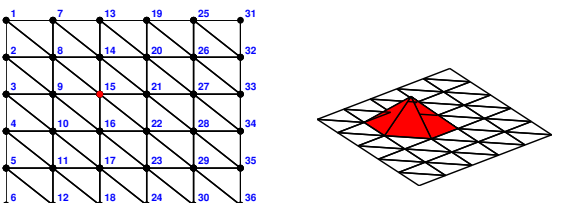
defined on the entire domain of interest.

Finite element basis functions

For each node (x_i, y_i) introduce a global finite element basis function $N_i(x, y)$ which is continuous on the entire domain on $\bar{\Omega}$ and linear on each element e_n and require that at the mesh nodes

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

A typical function of this type is illustrated as



Note $N_i(x, y)$ is only nonzero on elements to which node (x_i, y_i) belongs.

Finite element basis functions

On a given element e_n only three finite element functions $N_i(x, y)$ are nonzero.

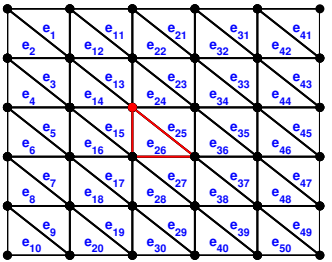
It follows that three local finite element basis functions can be defined for each element e_n, $n = 1, 2, ..., N$ as follows

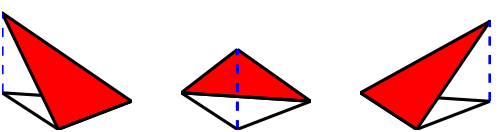
$$\begin{aligned} N_1^{(n)}(x, y) &= N_i(x, y), \quad (x, y) \in e_n \\ N_2^{(n)}(x, y) &= N_j(x, y), \quad (x, y) \in e_n \\ N_3^{(n)}(x, y) &= N_k(x, y), \quad (x, y) \in e_n \end{aligned}$$

For implementation purposes it is important to make use of the local ordering of nodes in each element.

Finite element basis functions

Three linear local finite element basisfunctions are illustrated for element e_{26}





Finite element basis functions

We want to derive explicit formulas for the local finite element basis functions.

Consider the first local finite element basis function $N_1^{(n)}(x, y)$ of element e_n. This function is linear

$$N_1^{(n)}(x, y) = a_1 + b_1 x + c_1 y, \quad (x, y) \in e_n$$

Denote the nodes of e_n (x_1, y_1), (x_2, y_2) and (x_3, y_3), and set

$$\begin{aligned} N_1^{(n)}(x_1, y_1) &= 1, \\ N_1^{(n)}(x_2, y_2) &= 0, \\ N_1^{(n)}(x_3, y_3) &= 0, \end{aligned}$$

These define relations that allow us to define $N_1^{(n)}(x, y)$ uniquely.

Finite element basis functions

We obtain the relations

$$\begin{aligned} a_1 + b_1 x_1 + c_1 y_1 &= 1, \\ a_1 + b_1 x_2 + c_1 y_2 &= 0, \\ a_1 + b_1 x_3 + c_1 y_3 &= 0. \end{aligned}$$

We can solve for the unknown coefficients a_1, b_1 and c_1 and find

$$a_1 = \frac{1}{2\Delta}\tilde{a}_1, \quad b_1 = \frac{1}{2\Delta}\tilde{b}_1, \quad c_1 = \frac{1}{2\Delta}\tilde{c}_1,$$

where

$$\tilde{a}_1 = x_2 y_3 - x_3 y_2, \quad \tilde{b}_1 = y_2 - y_3, \quad \tilde{c}_1 = x_3 - x_2,$$

and

$$\Delta = \tfrac{1}{2}\left(x_2 y_3 - y_2 x_3 - (x_1 y_3 - y_1 x_3) + x_1 y_2 - y_1 x_2\right).$$

Finite element basis functions

Similar results can be found for the two remaining local finite element basis functions. We summarize the results as follows

$$\begin{aligned} N_1^{(n)}(x,y) &= \frac{1}{2\Delta}\left(\tilde{a}_1 + \tilde{b}_1 x + \tilde{c}_1 y\right) \\ N_2^{(n)}(x,y) &= \frac{1}{2\Delta}\left(\tilde{a}_2 + \tilde{b}_2 x + \tilde{c}_2 y\right) \\ N_3^{(n)}(x,y) &= \frac{1}{2\Delta}\left(\tilde{a}_3 + \tilde{b}_3 x + \tilde{c}_3 y\right) \end{aligned}$$

where

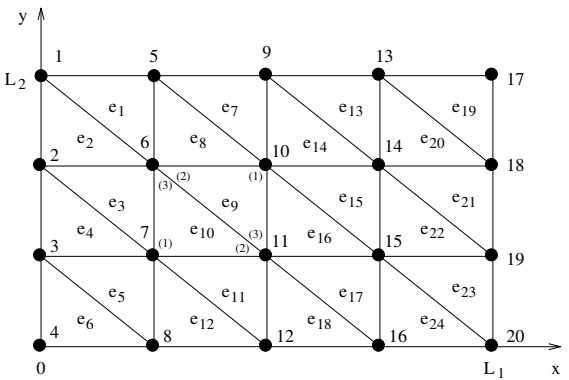
$$\tilde{a}_i = x_j y_k - x_k y_j, \quad \tilde{b}_i = y_j - y_k, \quad \tilde{c}_i = x_k - x_j$$

for the local node numbers $(i,j,k) = (1,2,3), (2,3,1), (3,1,2)$.

The magnitude of the parameter $|\Delta|$ is the area of the element e_n. $\Delta > 0$ for a local ordering counterclockwise, and $\Delta < 0$ for a clockwise ordering.

A mesh on a rectangular domain

To have a reference mesh, consider the following mesh on a rectangular domain



The local ordering of nodes for elements e_9 and e_{10} applies to all elements.

A mesh on a rectangular domain

For the simple mesh on a rectangular domain it is possible to describe the complete mesh in terms of (a) a coordinate table (VX) and (b) a connectivity table/element table (EToV) as follows

	VX	
i	x_i	y_i
1	0	L_2
2	0	$\frac{2}{3}L_2$
3	0	$\frac{1}{3}L_2$
4	0	0
5	$\frac{1}{4}L_1$	L_2
6	$\frac{1}{2}L_1$	$\frac{2}{3}L_2$
⋮	⋮	⋮
20	L_1	0

	EToV		
n \ r	1	2	3
1	5	1	6
2	2	6	1
3	6	2	7
4	3	7	2
⋮	⋮	⋮	⋮
24	16	20	15

Remark, we do not need to store the row numbers in the mesh tables!

Computing geometric information

We seek to determine an outward pointing normal vector (to the element) for an element edge.

Assume that the order of element vertices is counterclockwise, then for an boundary edge defined from (x_1, y_1) to (x_2, y_2) we find

$$\Delta x = x_2 - x_1, \quad \Delta y = y_2 - y_1$$

and thus a tangential vector becomes

$$\boldsymbol{t} = (t_1, t_2)^T = (\Delta x, \Delta y)^T$$

which should be orthogonal to the normal vector. Hence an outward pointing normalized vector is given as

$$\boldsymbol{n} = (n_1, n_2)^T = (t_2, -t_1)^T / \sqrt{t_1^2 + t_2^2}$$

Normal vectors needed for incorporating boundary conditions.

Interpolation

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

An interpolating function $u_I(x,y) \in P_h^1(\bar{\Omega})$ can then be defined as

$$u_I(x,y) = \sum_{i=1}^{M} u(x_i,y_i) N_i(x,y), \quad (x,y) \in \bar{\Omega},$$

where $N_i(x,y)$ is a global finite element basis function.

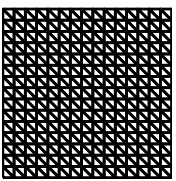
It follows from the property of the basis functions that

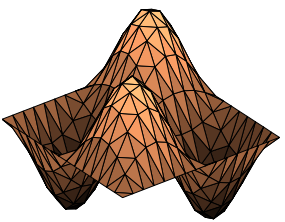
$$u_I(x_i,y_i) = u(x_i,y_i), \quad i = 1,2,...,M.$$

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

Interpolating function

An example of $u_I(x, y)$ interpolating $u(x, y) = \sin(2\pi x)\sin(2\pi y)$. Geometrically, $u_I(x, y)$ describes a surface made up of triangles joined at the edges.





Interpolation

Bounds for the interpolation error defined as

$$\epsilon(x, y) = u_I(x, y) - u(x, y)$$

can be derived under the assumption that all second-order derivatives of $u(x, y)$ are bounded, i.e.

$$|u_{xx}| \leq M_2, \quad |u_{xy}| \leq M_2, \quad |u_{yy}| \leq M_2, \quad (x, y) \in \bar{\Omega}.$$

Furthermore, assume that the mesh boundary coincides with Γ.

Then, there exist a constant C, independent of $u(x, y)$ and the mesh such that for $n = 1, 2, ..., N$

$$|\epsilon(x, y)| \leq C M_2 h_n^2, \quad (x, y) \in e_n,$$

where h_n is the largest edge length in e_n.

Stationary heat conduction

Analysis of stationary heat conduction in a two-dimensional body leads to a partial differential equation for its temperature distribution. The relevant quantities are:

$q_1(x, y)$:	the heat flux in the x direction
$q_2(x, y)$:	the heat flux in the y direction
$\lambda_1(x, y)$:	the heat conductivity in the x direction
$\lambda_2(x, y)$:	the heat conductivity in the y direction
$u(x, y)$:	the spatial temperature distribution
$\tilde{q}(x, y)$:	a heat source (if $\tilde{q}(x, y) > 0$) or heat sink (if $\tilde{q}(x, y) < 0$)

Stationary heat conduction

Figure: Jean Baptiste Joseph Fourier (21 March 1768 - 16 May 1830) was a French mathematician and physicist born in Auxerre and best known for initiating the investigation of Fourier series and their applications to problems of heat transfer and vibrations. Source: Wikipedia.

We will need Fourier's law, which states relationships for the heat fluxes in the x- and $y-directions$, respectively

$$q_1(x,y) = -\lambda_1(x,y)u_x(x,y),$$
$$q_2(x,y) = -\lambda_2(x,y)u_y(x,y),$$

The direction of heat transfer depends on the temperature gradients $\nabla u(x,y) = (u_x, u_y)^T$ in the body. Heat flux is positive in the direction from hot to cold temperatures.

Stationary heat conduction

From the law of conservation of thermal energy we have the heat balance equation

$$\text{(Heat out) - (Heat in)} = \text{(Heat produced)}$$

This law can be stated in the form of a PDE

$$(q_1)_x + (q_2)_y = \tilde{q}, \quad (x, y) \in \bar{\Omega}$$

having assumed that the heat flux is a smooth function which is differentiable (cf. derivation in notes).

Using Fourier's law of heat conduction, this PDE can be stated for the temperature $u(x, y)$ as

$$(\lambda_1(x, y)u_x)_x + (\lambda_2(x, y)u_y)_y = -\tilde{q}, \quad (x, y) \in \bar{\Omega}$$

Stationary heat conduction

Important special cases of the stationary heat equation are given below;

Poisson's equation

$$u_{xx} + u_{yy} = -\frac{\tilde{q}(x,y)}{\lambda},$$

where $\lambda_1(x,y) = \lambda_2(x,y) = \lambda$ (constant).

If, there is no heat sources we can set $\tilde{q} = 0$ and we find Laplace's equation

$$u_{xx} + u_{yy} = 0.$$

A Boundary Value Problem (BVP) for the temperature distribution is formulated by specifying boundary conditions.

Boundary conditions

The solution of PDEs requires a set of boundary conditions for $(x, y) \in \Gamma$ to ensure that the mathematical problem is well-posed, i.e. the problem has only one and only one solution.

The most common types of boundary conditions are (with physical interpretations, cf. the Heat Problem);

Dirichlet boundary condition:

$$u = f(x, y), \quad (x, y) \in \Gamma$$

Physical interpretation: The function $f(x, y)$ is a given temperature distribution on the boundary.

Boundary conditions

Neumann boundary condition:

$$\lambda_1(x,y)u_x n_1(x,y) + \lambda_2(x,y)u_y n_2(x,y) = -q(x,y)$$

or written in vector notation

$$\boldsymbol{n} \cdot \begin{pmatrix} -\lambda_1(x,y)u_x \\ -\lambda_2(x,y)u_y \end{pmatrix} = q(x,y)$$

Physical interpretation: The function $q(x,y)$ is the heat flux on the boundary point (x,y) in the direction of the outer normal vector $\boldsymbol{n} = (n_1, n_2)^T$. If $q > 0$ then heat leaves the body, if $q < 0$ then heat enters the body, and if $q = 0$ the body is insulated at the point.

Boundary conditions

Neumann boundary condition:

Neumann boundary conditions always have to be combined with one of the other types of boundary conditions to ensure that a unique solution exist.

If $u(x, y)$ is a solution that satisfies the PDE on Ω and with specified Neumann boundary conditions on Γ, then so does the function $u(x, y) + C$ with C an arbitrary constant (infinite number of solutions).

Boundary conditions

Neumann boundary condition:

An example of valid specification of the Neumann boundary condition;

Let Γ be decomposed into two pieces Γ_1 and Γ_2. An example of the valid use of the Neumann boundary condition is

$$\lambda_1 u_x n_1 + \lambda_2 u_y n_2 = -q, \quad (x, y) \in \Gamma_1$$

together with

$$u(x, y) = f(x, y), \quad (x, y) \in \Gamma_2$$

or

$$\lambda_1 u_x n_1 + \lambda_2 u_y n_2 = -h(u - u_0), \quad (x, y) \in \Gamma_2$$

Cases, where boundary conditions of different types are imposed on different parts of the boundary, are called mixed boundary conditions.

Boundary conditions

Robin boundary condition:

$$\lambda_1(x,y)u_x n_1(x,y) + \lambda_2(x,)u_y n_2(x,y) = -h(x,y)[u(x,y) - u_0(x,y)]$$

The Robin boundary condition is a linear combination of Dirichlet and Neumann boundary conditions.

Physical interpretation (Newton's law of cooling): The function $h(x,y)$ is the convective heat transfer coefficient at the boundary, and $u_0(x,y)$ is the external ambient temperature. The exchange of energy with surroundings is defined such that the heat flux across the interface of a body is proportional to the temperature difference across the interface.

The finite element method

Consider the Stationary Heat Problem governed by the PDE

$$(q_1)_x + (q_2)_y = \tilde{q}, \quad (x, y) \in \bar{\Omega}$$

combined with Dirichlet Boundary Conditions (BCs) specified on $(x, y) \in \Gamma$.

We will derive a weak formulation of this boundary value problem in order to solve it using the finite element method.

The finite element method

Let $v(x,y)$ be an arbitrary smooth function defined on $\bar{\Omega}$ with property

$$v(x,y) = 0, \quad (x,y) \in \Gamma.$$

Multiply both sides of the PDE by $v(x,y)$ and integrate over $\bar{\Omega}$

$$\iint_\Omega \left[(\lambda_1 u_x)_x + (\lambda_2 u_y)_y\right] v dxdy = -\iint_\Omega \tilde{q} v dxdy$$

Using integration by parts we can find the identities

$$\begin{aligned}
\iint_\Omega \left[(\lambda_1 u_x)_x\right] v dxdy &= \int_\Gamma \lambda_1 u_x v n_1 ds - \iint_\Omega \lambda_1 u_x v_x dxdy \\
\iint_\Omega \left[(\lambda_2 u_y)_y\right] v dxdy &= \int_\Gamma \lambda_2 u_y v n_2 ds - \iint_\Omega \lambda_2 u_y v_y dxdy
\end{aligned}$$

The finite element method

Using the two identities we can rewrite the integral form of the PDE in the form

$$\int_\Gamma (\lambda_1 u_x n_1 + \lambda_2 u_y n_2)\, v ds - \iint_\Omega (\lambda_1 u_x v_x + \lambda_2 u_y v_y)\, dxdy = -\iint_\Omega \tilde{q} v dxdy$$

However, we required that $v = 0$ on Γ and therefore this formulation reduces to the weak formulation of the problem

$$\iint_\Omega (\lambda_1 u_x v_x + \lambda_2 u_y v_y)\, dxdy = \iint_\Omega \tilde{q} v dxdy$$

A function $u(x, y)$ which is a solution to the problem should satisfy

- the weak formulation for all sufficiently smooth functions v with $v = 0$ on Γ.
- the specified Dirichlet boundary conditions on Γ.

The finite element method

Consider a mesh on $\bar{\Omega}$ with elements, e_n, $n = 1, 2, ..., N$ and nodes (x_i, y_i), $i = 1, 2, ..., M$. Seek an approximation $\hat{u}(x, y) \in P_h^1(\bar{\Omega})$ to $u(x, y) \in \mathcal{C}^2(\bar{\Omega})$ given as

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

where $N_i(x_i, y_i) = 1$, $N_j(x_i, y_i) = 0$, $j \neq i$. To satisfy the Dirichlet boundary conditions put

$$\hat{u}_i = f(x_i, y_i), \quad \forall i : \ (x_i, y_i) \in \Gamma$$

This ensures that the BCs are enforced

$$\hat{u}(x_i, y_i) = f(x_i, y_i), \quad \forall i : \ (x_i, y_i) \in \Gamma$$

The finite element method

To determine a finite element approximation $\hat{u}(x, y)$ we need to determine $M = M_\Omega + M_\Gamma$ coefficients.

- From the Dirichlet boundary conditions M_Γ coefficients are known.
- The remaining M_Ω coefficients are unknown and has to be determined.

The unknown coefficients are determined using the weak formulation of the formulated PDE problem.

The finite element method

To find the M_Ω coefficients, substitute $\hat{u}(x,y)$ for $u(x,y)$ and put $v(x)$ equal to $N_i(x,y)$, $\forall i$ where $(x_i, y_i) \in \Omega$.

From the above process we find M_Ω equations of the type

$$\iint_\Omega \left[\lambda_1 \hat{u}_x (N_i)_x + \lambda_2 \hat{u}_y (N_i)_y\right] dxdy = \iint_\Omega \tilde{q} N_i dxdy$$

or

$$\iint_\Omega \left[\lambda_1 \sum_{j=1}^{M_\Omega} \hat{u}_j (N_j)_x (N_i)_x + \lambda_2 \sum_{j=1}^{M_\Omega} \hat{u}_j (N_j)_y (N_i)_y\right] dxdy = \iint_\Omega \tilde{q} N_i dxdy$$

Which we can express in the form of a linear system $A\hat{u} = b$ with M_Ω equations.

The finite element method

Each equation in the linear system $A\hat{u} = b$ is defined from

$$\sum_{j=1}^{M_\Omega} a_{i,j}\hat{u}_j = b_i, \quad \forall i : \ (x_i, y_i) \in \Omega$$

where

$$\begin{aligned} a_{i,j} &= \iint_\Omega \left[\lambda_1 (N_i)_x (N_j)_x + \lambda_2 (N_i)_y (N_j)_y\right] dxdy \\ b_i &= \iint_\Omega \tilde{q} N_i dxdy \end{aligned}$$

To solve the linear system for the unknown $\hat{u}$ we need to compute these coefficients.

The finite element method

To determine the coefficients we first rewrite them into a sum of element contributions as

$$a_{i,j} = \sum_{n=1}^{N} \iint_{e_n} \left[\lambda_1 (N_i)_x (N_j)_x + \lambda_2 (N_i)_y (N_j)_y\right] dxdy$$

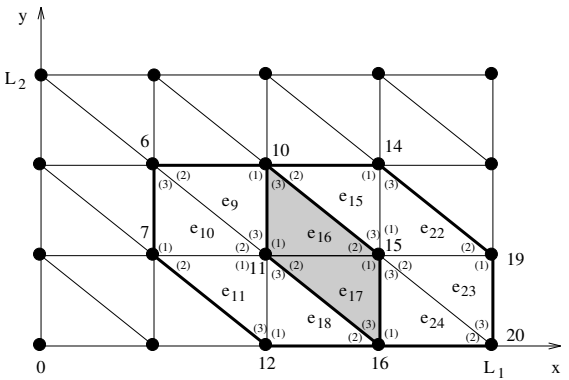
$$b_i = \sum_{n=1}^{N} \iint_{e_n} \tilde{q} N_i dxdy$$

Due to the choice of global basis functions we find

- $a_{i,j} \neq 0$ if the nonzero parts of $N_i(x, y)$ and $N_j(x, y)$ overlap.
- $a_{i,j} = 0$ if the nonzero parts of $N_i(x, y)$ and $N_j(x, y)$ has no overlap.
- b_i is determined from a sum of contributions of those elements where $N_i(x, y)$ is nonzero.

Element matrices

In the reference mesh we see, for example, that nodes 11 and 15 are directly connected.



$N_{11}(x, y)$ is nonzero in elements 9, 10, 11, 16, 17 and 18, while $N_{15}(x, y)$ is nonzero in elements 15, 16, 17, 22, 23, 24.

Element matrices

Thus, we find that only two integral terms has nonzero contributions to $a_{11,15}$, i.e.

$$a_{11,15} = \iint_{e_{16}} \left[\lambda_1 \left(N_{11}\right)_x \left(N_{15}\right)_x + \lambda_2 \left(N_{11}\right)_y \left(N_{15}\right)_y\right] dxdy + \iint_{e_{17}} \left[\lambda_1 \left(N_{11}\right)_x \left(N_{15}\right)_x + \lambda_2 \left(N_{11}\right)_y \left(N_{15}\right)_y\right] dxdy$$

In terms of the local basis functions this becomes

$$a_{11,15} = \iint_{e_{16}} \left[\lambda_1 \left(N_1^{(16)}\right)_x \left(N_2^{(16)}\right)_x + \lambda_2 \left(N_1^{(16)}\right)_y \left(N_2^{(16)}\right)_y\right] dxdy + \iint_{e_{17}} \left[\lambda_1 \left(N_2^{(17)}\right)_x \left(N_1^{(17)}\right)_x + \lambda_2 \left(N_2^{(17)}\right)_y \left(N_1^{(17)}\right)_y\right] dxdy$$

Element matrices

For each element e_n we now introduce the element matrix

$$K_n = \begin{bmatrix} k_{1,1}^{(n)} & k_{1,2}^{(n)} & k_{1,3}^{(n)} \\ k_{2,1}^{(n)} & k_{2,2}^{(n)} & k_{2,3}^{(n)} \\ k_{3,1}^{(n)} & k_{3,2}^{(n)} & k_{3,3}^{(n)} \end{bmatrix}$$

where

$$k_{r,s}^{(n)} = \iint_{e_n} \left[\lambda_1 (N_r^{(n)})_x (N_s^{(n)})_x + \lambda_2 (N_r^{(n)})_y (N_s^{(n)})_y \right] dxdy$$

We observe that we can then express $a_{11,15}$ as

$$a_{11,15} = k_{1,2}^{(16)} + k_{2,1}^{(17)}$$

Element matrices

Each nonzero $a_{i,j}$ is a sum of some number of element matrix entries $k_{r,s}^{(n)}$.

If $i \neq j$ then the sum consists of at most two terms. Two terms if nodes (x_i, y_i) and (x_j, y_j) is directly connected.

If $i = j$ then the sum consists of a number of terms corresponding to the number of elements which the i'th node belongs.

Element matrices

Example:

To determine the right hand side vector elements b_i of the linear system we find that

$$
\begin{aligned}
b_{11} &= \sum_{n=9,10,11,16,17,18} \iint_{e_n} \tilde{q} N_{11} dxdy \\
&= \iint_{e_9} \tilde{q} N_3^{(9)} dxdy + \iint_{e_{10}} \tilde{q} N_2^{(10)} dxdy + \iint_{e_{11}} \tilde{q} N_1^{(11)} dxdy \\
&+ \iint_{e_{16}} \tilde{q} N_1^{(16)} dxdy + \iint_{e_{17}} \tilde{q} N_2^{(17)} dxdy + \iint_{e_{18}} \tilde{q} N_3^{(18)} dxdy
\end{aligned}
$$

Element matrices

For each element e_n we define an element vector with the number of elements equal to the number of local basis functions by

$$\tilde{\boldsymbol{q}}_n = \begin{bmatrix} \tilde{q}_1^{(n)} \\ \tilde{q}_2^{(n)} \\ \tilde{q}_3^{(n)} \end{bmatrix}$$

where

$$\tilde{q}_r^{(n)} = \iint_{e_n} \tilde{q} N_r^{(n)} dxdy$$

Everey b_i can be expressed as a sum of components of element vectors. For example,

$$b_{11} = \tilde{q}_3^{(9)} + \tilde{q}_2^{(10)} + \tilde{q}_1^{(11)} + \tilde{q}_1^{(16)} + \tilde{q}_2^{(17)} + \tilde{q}_3^{(18)}$$

Element matrices

To evaluate $k_{r,s}^{(n)}$ and $\tilde{q}_r^{(n)}$ we insert the expressions for the local basis functions

$$k_{r,s}^{(n)} = \frac{1}{4\Delta^2} \iint_{e_n} \left(\lambda_1 \tilde{b}_r \tilde{b}_s + \lambda_2 \tilde{c}_r \tilde{c}_s\right) dxdy, \quad r,s = 1,2,3$$

$$\tilde{q}_r^{(n)} = \frac{1}{2\Delta} \iint_{e_n} \tilde{q} \left(\tilde{a}_r + \tilde{b}_r x + \tilde{c}_r y\right) dxdy, \quad r = 1,2,3$$

In the case when λ_1, λ_2 and $\tilde{q}$ are constant with respect to x and y, we find that

$$k_{r,s}^{(n)} = \frac{1}{4|\Delta|} \left(\lambda_1 \tilde{b}_r \tilde{b}_s + \lambda_2 \tilde{c}_r \tilde{c}_s\right), \quad r,s = 1,2,3$$

$$\tilde{q}_r^{(n)} = \frac{|\Delta|}{3} \tilde{q}, \quad r = 1,2,3$$

Element matrices

If any of the functions λ_1, λ_2 or $\tilde{q}$ are not constant, they can be approximated by their value at the center (x_c, y_c) of each element e_n defined by

$$(x_c, y_c) = \left(\frac{x_i + x_j + x_k}{3}, \frac{y_i + y_j + y_k}{3} \right)$$

or by the average of the function values at the three nodes of element e_n.

The computation

Tasks:

- Compute A and $\boldsymbol{b}$
  - Part I : Setup matrix and vector while ignoring boundary conditions (most work)
  - Part II : Modify matrix and vector to impose boundary conditions (minor work)
- A is a sparse symmetric positive definite matrix for this problem
- Can use sparse storage scheme and exploit symmetry property of A
- Computational work can be reduced by ensuring a good ordering of equations and unknowns

The computation

Algorithm 4 (Pseudo Matlab-code)

```
% ALLOCATE STORAGE
b = zeros(M,1);
A = zeros(M,M);
% DETERMINE elements of A and b
for n = 1 to N
    Look up the global numbers and
      (x,y)-coordinates of the nodes in e_n.
    for r = 1 to 3
        Compute q̃_r^(n) from (2.28)
        % N.B.: i is the global number of node r
        b(i) = b(i) + q̃_r^(n)
        for s = 1 to 3
            Compute k_{r,s}^(n) from (2.27)
            % N.B.: j is the global number of node s
            a(i,j) = a(i,j) + k_{r,s}^(n)
```

End of algorithm

The computation

Comments to the Algorithm 4;

- No advantages of the symmetry exploited, namely that $a_{i,j} = a_{j,i}$ and $k_{r,s}^{(n)} = k_{s,r}^{(n)}$.
- To exploit symmetry, it is only necessary to compute $a_{i,j}$ for $j \geq i$ and $k_{r,s}^{(n)}$ for $s \geq r$.
- Full array storage allocate for A although it is very sparse since $a_{i,j}$ is nonzero only if nodes (x_i, y_i) and (x_j, y_j) are directly connected in the mesh.

Algorithm 5 eliminates both types of unnessesary work.

The computation

Algorithm 5 (Pseudo Matlab-code)

```
% ALLOCATE STORAGE
b = zeros(M,1); A = spalloc(M,M,nnzmax);
% DETERMINE elements of A and b
for n = 1 to N
    Look up the global numbers and (x,y)-coordinates of the nodes in e_n.
    for r = 1 to 3
        Compute q̃_r^(n) from (2.28)
        % N.B.: i is the global number of node r
        b(i) = b(i) + q̃_r^(n)
        for s = r to 3
            Compute k_{r,s}^(n) from (2.27)
            % N.B.: j is the global number of node s
            if j ≥ i
                a(i,j) = a(i,j) + k_{r,s}^(n)
            else
                a(j,i) = a(j,i) + k_{r,s}^(n)
```

End of algorithm

The computation

Let's consider the handling of Dirichlet boundary conditions.

This is illustrated for a case where $M = 4$ with only a single node on the boundary, namely (x_2, y_2).
(Note that no mesh can in fact have this property!)

After Algorithm 4 the linear system can be expressed as

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \\ \hat{u}_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

The computation

The Dirichlet boundary condition makes $\hat{u}_2 = f_2$. Hence, we modify the system into

$$\begin{bmatrix} a_{1,1} & 0 & a_{1,3} & a_{1,4} \\ 0 & 1 & 0 & 0 \\ a_{3,1} & 0 & a_{3,3} & a_{3,4} \\ a_{4,1} & 0 & a_{4,3} & a_{4,4} \end{bmatrix} \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \\ \hat{u}_4 \end{bmatrix} = \begin{bmatrix} b_1 - a_{1,2} f_2 \\ f_2 \\ b_3 - a_{3,2} f_2 \\ b_4 - a_{4,2} f_2 \end{bmatrix}$$

The purpose of Algorithm 6 is to perform modifications of this type. First, consider a primitive version of this algorithm...

The computation

Algorithm 6 (in pseudo Matlab-code)

```
if (x_i, y_i) ∈ Γ
    a(i,i) = 1; b(i) = f(i);
    for j = 1 : M ( j ≠ i )
        a(i,j) = 0;
        if (x_j, y_j) ∈ Ω
            b(j)= b(j) - a(j,i) * f(i);
            a(j,i) = 0;
```

End of algorithm

This algorithm does

- too much work (because most $a_{i,j}$ and $a_{j,i}$ are zero)
- not exploit symmetry

The computation

Let's define two sets of global node numbers. Let S_Γ denote the set of global node numbers of nodes on $(x_i, y_i) \in \Gamma$.

$$S_\Gamma \in \{i : (x_i, y_i) \in \Gamma\}$$

For every $i \in S_\Gamma$ let $S_{\bar{\Omega}}^{(i)}$ denote the set of global node numbers of those nodes in $\bar{\Omega}$ that are directly connected to (x_i, y_i). For the reference mesh, we find

$$S_\Gamma = \{1, 2, 3, 4, 5, 8, 9, 12, 13, 16, 17, 18, 19, 20\},$$
$$S_{\bar{\Omega}}^{(5)} = \{1, 6, 9, 10\}$$

The computation

Algorithm 7 (in pseudo Matlab-code)

```
if i ∈ S_Γ
    a(i,i) = 1; b(i) = f(i);
    for j ∈ S_Ω^(i)
        if j < i
            b(j) = b(j) - a(j,i) * f(i);
            a(j,i) = 0;
        else
            b(j) = b(j) - a(i,j) * f(i);
            a(i,j) = 0;
```

End of algorithm This algorithm should be executed after Algorithm 4.

The computation

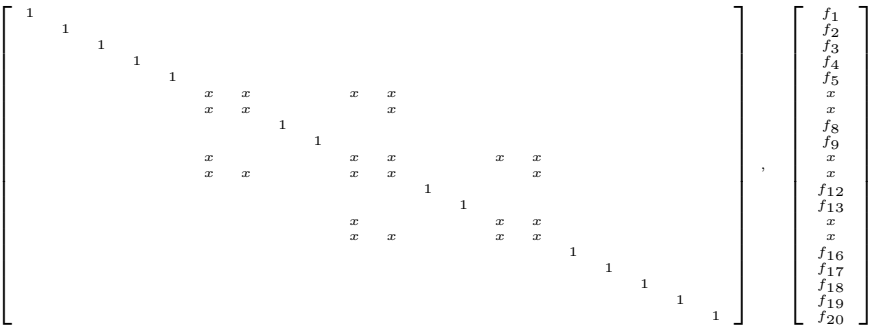
Having executed Algorithm 4 and then Algorithm 7 we have determined an M'th-order linear system of equations of the form

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

Let's consider the appearance of the system of equations (see page 44 in notes) with Dirichlet conditions imposed for the mesh shown in Figure 2.5...

The computation

For the mesh shown in Figure 2.5, A and $\boldsymbol{b}$ have the following appearance:



For every $i \in S_\Gamma$, the i'th row of system is of the form $\hat{u}_i = f_i$ and those rows are uncoupled from the system.

The computation

For the mesh with different node ordering shown in Figure 2.9, A and $\boldsymbol{b}$ have the following appearance:

$$
\begin{bmatrix}
x & x & x & x & & & & & & & & & & & & & & & & \\
x & x & & x & & & & & & & & & & & & & & & & \\
x & & x & x & x & x & & & & & & & & & & & & & & \\
x & x & x & x & & x & & & & & & & & & & & & & & \\
 & & x & & x & x & & & & & & & & & & & & & & \\
 & & x & x & x & x & & & & & & & & & & & & & & \\
 & & & & & & 1 & & & & & & & & & & & & & \\
 & & & & & & & 1 & & & & & & & & & & & & \\
 & & & & & & & & 1 & & & & & & & & & & & \\
 & & & & & & & & & 1 & & & & & & & & & & \\
 & & & & & & & & & & 1 & & & & & & & & & \\
 & & & & & & & & & & & 1 & & & & & & & & \\
 & & & & & & & & & & & & 1 & & & & & & & \\
 & & & & & & & & & & & & & 1 & & & & & & \\
 & & & & & & & & & & & & & & 1 & & & & & \\
 & & & & & & & & & & & & & & & 1 & & & & \\
 & & & & & & & & & & & & & & & & 1 & & & \\
 & & & & & & & & & & & & & & & & & 1 & & \\
 & & & & & & & & & & & & & & & & & & 1 & \\
 & & & & & & & & & & & & & & & & & & & 1
\end{bmatrix}
,
\begin{bmatrix}
x \\ x \\ x \\ x \\ x \\ x \\ f_7 \\ f_8 \\ f_9 \\ f_{10} \\ f_{11} \\ f_{12} \\ f_{13} \\ f_{14} \\ f_{15} \\ f_{16} \\ f_{17} \\ f_{18} \\ f_{19} \\ f_{20}
\end{bmatrix}
$$

Again, for every $i \in S_\Gamma$, the i'th row of system is of the form $\hat{u}_i = f_i$ and those rows are uncoupled from the system.

The computation

The first six unknowns can be found by solving the smaller system

$$\begin{bmatrix} x & x & x & x & & \\ x & x & & x & & \\ x & & x & x & x & x \\ x & x & x & x & & x \\ & & x & & x & x \\ & & x & x & x & x \end{bmatrix} \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \\ \hat{u}_4 \\ \hat{u}_5 \\ \hat{u}_6 \end{bmatrix} = \begin{bmatrix} x \\ x \\ x \\ x \\ x \\ x \end{bmatrix}$$

In general, whenever the boundary condition is of Dirichlet type then we could work with a system of order M_Ω instead of one of order M.

However, the impact on performance between using a reduced and a full system of equations will be less important the larger the system because the boundary nodes is a much smaller fraction of the total number of equations M.

The computation

To solve the linear system of the form $A\hat{\boldsymbol{u}} = \boldsymbol{b}$ we are faced with choice of

- data structure for A (should be practical)
- direct vs. iterative methods (should be efficient)

Direct solution techniques, e.g. Gaussian elimination or Cholesky factorization can suffer in performance due to fill-in[1] or rounding errors. To minimize fill-in and resulting impact on performance, both nodes and equations can be reordered. Direct methods can be made very efficient if both symmetry and sparsity is exploited.

Iterative solution techniques, e.g. Successive OverRelaxation (SOR) or Conjugate Gradient (CG) methods, produce a sequence of vectors that converge to the solution. The methods are attractive for large problems as these do not create fill-in. Techniques for enhancing the convergence rate may be needed.

[1] nonzero entries in matrix in positions originally occupied by zeros.

The computation

An important type of sparse matrix is a band matrix, where the nonzero entries are concentrated close to the main diagonal.

Let $\phi(i)$ denote the column number of the last nonzero in the i'th row of A, and let

$$h(A) = \max_{1 \le i \le M} \{\phi(i) - i + 1\}$$
$$b(A) = 2h(A) - 1$$

$h(A)$ is called the half-bandwidth and $b(A)$ is called the bandwidth of A.

Note: if the bandwidth is small we can be sure that the matrix is sparse, but the converse is not necessarily the case.

The computation

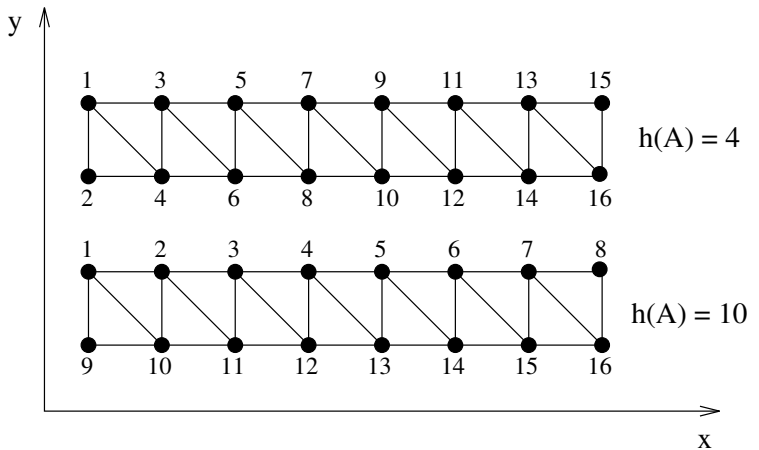


Figure: 2.10 Two orderings of a mesh and the corresponding half-bandwidths. (Boundary conditions ignored)

The computation

Purposes of fill-reducing matrix permutations

- Minimize bandwidth (keep nonzeros close to diagonal).
- Reduce fill-in for factorizations ($\Rightarrow$ less work!).

In Matlab, e.g. try orderings `ymrcm , \verb ymamd , md , \verb colperm , \verb colmd .`

```
>> p = symamd(A);
>> [U] = chol(A(p,p));
>> u(p) = U \ (U' \ b(p));
```

or

```
>> p = symamd(A);
>> u(p) = A(p,p) \ b(p);
```

Count nonzeros in factorization

```
>> p = symamd(A);
>> [U] = chol(A(p,p));
>> nonzeros = nnz(U)
```

Towards more advanced computations...

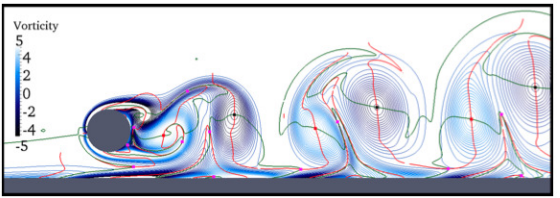


Figure: Analysis of vortex structures in behind a cylinder using a high-order Spectral/hp-FEM solver (M.Sc. Rasmus E. Christensen, DTU Compute 2013).

We have considered fundamental details of how to construct FEM in two space dimensions.

We have seen

- Conceptually, the extension to 2D from 1D is straightforward.
- Main new complications are proper choice of data structures and efficiency issues.

Next lecture, we shall discuss mesh generation issues...