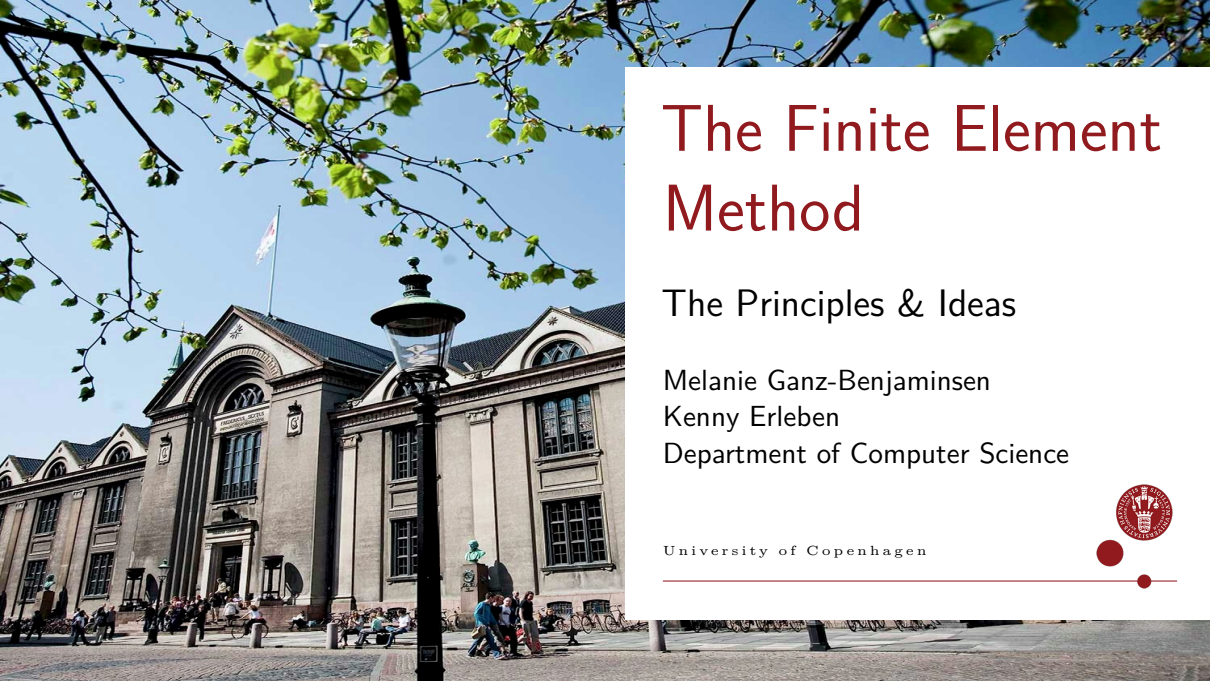


The Finite Element Method

The Principles & Ideas

Melanie Ganz-Benjaminsen
Kenny Erleben
Department of Computer Science

University of Copenhagen

Background

- The Finite Element Method (FEM) has a long history. It has been used and studied in many fields, including mathematics, engineering, computer science, etc. Its origins go back to before we even had computers.
- It is a method for discretizing and solving a partial differential equation. FEM is based on strong mathematical concepts that allow one to solve pretty much anything as long as one follows the "five" step recipe shown on the following pages. One is given a guarantee that things will work, and one will be able to compute a solution.
- FEM has many more strengths, but one thing that stands out with previous methods we have looked at is that it can be applied to unstructured meshes. That is for instance triangle or tetrahedral meshes.

The Basic Steps

The five basic steps in deriving a FEM for a given problem

- STEP 1: Rewrite the problem as a volume integral
- STEP 2: Do integration by parts
- STEP 3: Make an approximation
- STEP 4: Choose a test function
- STEP 5: Compute a solution

Now let us try it out!

First we need a Problem

We wish to find the unknown function $y(x) : \mathbb{R} \mapsto \mathbb{R}$ on some domain $x \in [x_1..x_n]$. All we know is some differential equation

$$\frac{\partial^2 y(x)}{\partial x^2} = c(x)$$

where $c(x) : \mathbb{R} \mapsto \mathbb{R}$ is some known function; also we are given some boundary conditions

$$y(x_1) = a\,,$$
$$y(x_n) = b\,.$$

We will find $y(x)$ using the finite element method (FEM).

STEP 1 – Rewrite into a Volume Integral

First, we multiply by an arbitrary $v(x)$ defined so we have $v(x_1) = v(x_n) = 0$ and then we integrate over our domain

$$\int_{x_1}^{x_n} v(x) \left(\frac{\partial^2 y(x)}{\partial x^2} - c(x) \right) dx = 0$$

The $v(x)$ function is called a test function or a weight function. We will go with the name test function from here on.
To remember the name, think of this function as a "random" valued function that is testing if the original differential problem holds at a specifically chosen point x.

Test Functions and Boundary Conditions

The boundary conditions are a bit trivial for this 1D example since they are point based.

- Technically for more complicated boundary conditions, one would apply the same step to the boundary conditions as these too are just some partial differential equations.
- Afterwards one just adds the transformed governing equation term with the transformed boundary condition term and ends up with a single scalar integral equation.

Let us try and write this out

$$\int_{x_1}^{x_n} v(x) \left(\frac{\partial^2 y(x)}{\partial x^2} - c(x) \right) dx + \int_{x_1}^{x_n} v_a(x) \left(y(x) - a \right) dx + \int_{x_1}^{x_n} v_b(x) \left(y(x) - b \right) dx = 0$$

We introduce two new test functions, one for each of our boundary equations. Here $v_a(x) = 0$ for all $x \neq x_1$ and $v_b(x) = 0$ for all $x \neq x_n$.

Test Functions and Boundary Conditions

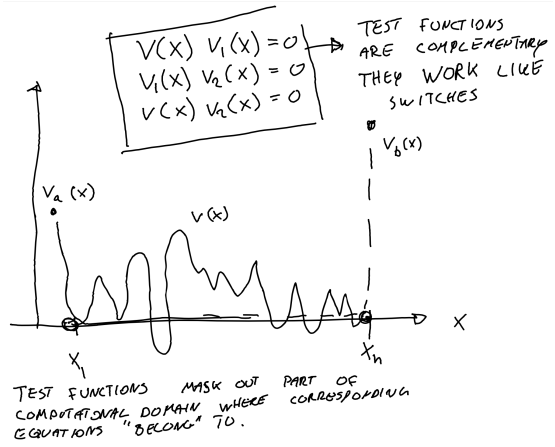
Because $v_a(x)$ and $v_b(x)$ in our example only are non-zero at the points x_1 and x_n the integrals collapse and we have,

$$\int_{x_1}^{x_n} v(x) \left(\frac{\partial^2 y(x)}{\partial x^2} - c(x) \right) dx + v_a(x_1) \left(y(x_1) - a \right) + v_b(x_n) \left(y(x_n) - b \right) = 0 \,.$$

To keep the math in this introduction lean and mean we will "ignore" the boundary terms in the rest of the introduction of the FEM machinery.

- Obviously the point-wise nature makes the boundary conditions in our example somewhat trivial to deal with.
- The idea we presented is general and can be applied to any kind of boundary condition equation. In the general case then one could need to perform the same further steps as we will show in the first term coming from the governing equation.

The Masking Property of Test Functions



STEP 2 – Integration by Parts

So we have

$$\int_{x_1}^{x_n} v(x)\left(\frac{\partial^2 y(x)}{\partial x^2} - c(x)\right)dx = \int_{x_1}^{x_n} v(x)\frac{\partial^2 y(x)}{\partial x^2}dx - \int_{x_1}^{x_n} v(x)c(x)dx = 0$$

Integration by parts of the first term on the right-hand-side (rhs) gives us

$$\left[v(x)\frac{\partial y(x)}{\partial x}\right]_{x_1}^{x_n} - \int_{x_1}^{x_n} \frac{\partial v(x)}{\partial x}\frac{\partial y(x)}{\partial x}dx - \int_{x_1}^{x_n} v(x)c(x)dx = 0$$

Using $v(x_1) = v(x_n) = 0$ and cleaning up yields

$$\int_{x_1}^{x_n} \frac{\partial v(x)}{\partial x}\frac{\partial y(x)}{\partial x}dx + \int_{x_1}^{x_n} v(x)c(x)dx = 0$$

What Happened?

Originally we had the term

$$\int v(x) \frac{\partial^2 y(x)}{\partial x^2} dx$$

Here $y(x)$ needs to be twice continuously differentiable. Now we have

$$\int \frac{\partial v(x)}{\partial x} \frac{\partial y(x)}{\partial x} dx$$

Now $y(x)$ only needs to be continuously differentiable.

$$\text{Strong Form} \Rightarrow \text{Weak Form}$$

The downside is that $v(x)$ must be differentiable.

STEP 3 – Make an Approximation

Rather than finding y we wish to find a good approximation $\tilde{y}$ which we write abstractly as

$$y(x) \approx \tilde{y}(x) = \sum_{i=1}^{n} N_i(x)\hat{y}_i$$

That is we have broken up our continuous function into a set of n discrete values $(\hat{y}_i, x_i)$ that we somehow combine using some weighting/interpolation scheme given by the global shape functions N_i. The wording "shape function " requires a bit of attention.

- A more catch-all term would be a basis function. We like the word shape function as it indicates the connection to the geometry of the elements. The N_i function tells us how things vary inside an element.
- In FEM textbooks and on web pages the word trial functions are often used. It really refers to the approximation function $\tilde{y}(x)$. We just call this an "approximation" and try to avoid the word "trail" as it semantically gets mixed with the word "test".

Introducing Matrix Notation

More elegantly we write the approximation compactly using matrix notation,

$$\tilde{y}(x) = \sum_{i=1}^{n} N_i(x)\hat{y}_i = \underbrace{\begin{bmatrix} N_1(x) & \cdots & N_n(x) \end{bmatrix}}_{\mathbf{N}(x)} \underbrace{\begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_n \end{bmatrix}}_{\hat{y}} = \mathbf{N}\hat{y}$$

Observe that $\mathbf{N}$ is a matrix and not a row vector. Later in a higher-dimensional domain, we will see that $\mathbf{N}$ will change dimensions.

A Few Notes on the Shape Functions

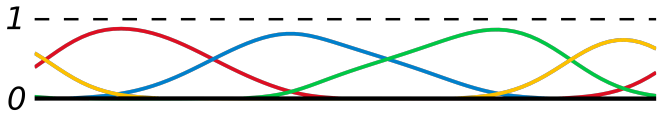
The shape function should interpolate the points

$$N_i(x) = \begin{cases} 1 & x = x_i \\ 0 & x = x_j \wedge j \neq i \end{cases}$$

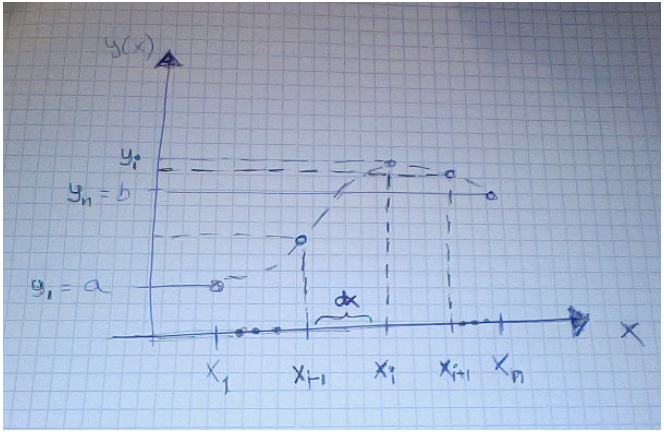
and the sum of all the function values at a given point x is 1

$$\sum_i N_i(x) = 1$$

The last condition is known as the partition of unity.



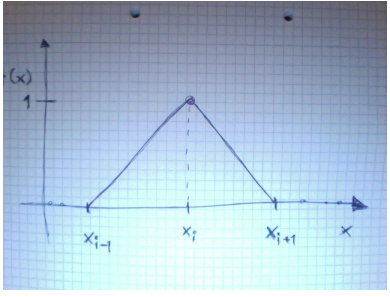
An Illustration of the Computational Mesh



Let us use the Hat-Function

Using linear interpolation yields the so-called hat functions

$$N_i(x) = \max\left(0, 1 - \left|\frac{x_i - x}{\Delta x}\right|\right) = \begin{cases} 1 - \frac{x_i - x}{\Delta x} & x_{i-1} \leq x \leq x_i \\ 1 - \frac{x - x_i}{\Delta x} & x_i < x \leq x_{i+1} \\ 0 & \text{otherwise} \end{cases}$$



Using the Approximation

So we have

$$\int_{x_1}^{x_n} \frac{\partial v(x)}{\partial x} \frac{\partial y(x)}{\partial x} dx + \int_{x_1}^{x_n} v(x)c(x)dx = 0$$

Let us insert the approximation into our integrals

$$\int_{x_1}^{x_n} \frac{\partial v(x)}{\partial x} \frac{\partial \mathbf{N}(x)}{\partial x} \hat{y} dx + \int_{x_1}^{x_n} v(x)c(x)dx = 0$$

This is the main trick of the finite element method (FEM)

FEM is a method of Approximation

STEP 4 – Choose a Test Function

We will make the specific choice of a test function

$$v(x) \equiv \mathbf{N}(x)\delta\mathbf{y}$$

where $\delta\mathbf{y} \equiv \begin{bmatrix}\delta y_1 & \cdots & \delta y_n\end{bmatrix}^T$ are understood to be a vector of completely arbitrary values at the discrete points, the values are like stochastic variables. There is still the matter of the boundaries. Technically, we should have δy_k be random for all $1 < k < n$ but $\delta y_1 = \delta y_n = 0$. A more precise statement would be,

$$\delta\mathbf{y} \equiv \begin{bmatrix}0 & \delta y_2 & \cdots & \delta y_{n-1} & 0\end{bmatrix}^T .$$

For the test functions $v_a(x)$ and $v_b(x)$ we see that we would have $\delta\mathbf{y}_a \equiv \begin{bmatrix}\delta y_1 & 0 & \cdots 0\end{bmatrix}^T$ and $\delta\mathbf{y}_b \equiv \begin{bmatrix}0 & \cdots 0 & \delta y_n\end{bmatrix}^T$ and then define $v_a(x) \equiv \mathbf{N}(x)\delta\mathbf{y}_a$ and $v_b(x) \equiv \mathbf{N}(x)\delta\mathbf{y}_b$.

Using the Test Function

Inserting our choice of test function yields

$$\int_{x_1}^{x_n} \frac{\partial \left(\mathbf{N}(x)\delta\mathbf{y}\right)}{\partial x} \frac{\partial \mathbf{N}(x)}{\partial x} \hat{y} dx + \int_{x_1}^{x_n} \mathbf{N}(x)\delta\mathbf{y} c(x) dx = 0$$

Since $\mathbf{N}(x)\delta\mathbf{y}$ is a scalar, it holds that $\mathbf{N}(x)\delta\mathbf{y} = \left(\mathbf{N}(x)\delta\mathbf{y}\right)^T$

$$\int_{x_1}^{x_n} \left(\frac{\partial \left(\mathbf{N}(x)\delta\mathbf{y}\right)^T}{\partial x} \right) \frac{\partial \mathbf{N}(x)}{\partial x} \hat{y} dx + \int_{x_1}^{x_n} \left(\mathbf{N}(x)\delta\mathbf{y}\right)^T c(x) dx = 0$$

Cleaning Up

Now

$$\delta \mathbf{y}^T \left(\left(\int_{x_1}^{x_n} \frac{\partial \mathbf{N}^T(x)}{\partial x} \frac{\partial \mathbf{N}(x)}{\partial x} dx \right) \hat{y} + \int_{x_1}^{x_n} \mathbf{N}^T(x) c(x) dx \right) = 0$$

must hold for all arbitrary values of $\delta \mathbf{y}$ so

$$\underbrace{\left(\int_{x_1}^{x_n} \frac{\partial \mathbf{N}^T(x)}{\partial x} \frac{\partial \mathbf{N}(x)}{\partial x} dx \right)}_{\mathbf{K}} \hat{y} + \underbrace{\int_{x_1}^{x_n} \mathbf{N}^T(x) c(x) dx}_{-\mathbf{f}} = \mathbf{0}$$

Thus we now have a simple linear system

$$\mathbf{K}\hat{y} = \mathbf{f}$$

STEP 5 – Compute a Solution

Compute (numerically or analytically)

$$\mathbf{K} = \int_{x_1}^{x_n} \frac{\partial \mathbf{N}^T(x)}{\partial x} \frac{\partial \mathbf{N}(x)}{\partial x} dx$$

$$\mathbf{f} = -\int_{x_1}^{x_n} \mathbf{N}^T(x) c(x) dx$$

then we have

$$\mathbf{K}\hat{y} = \mathbf{f}$$

We are almost done, we just need to add the boundary conditions before we can solve the linear system for our solution $\hat{y}$.

Remember to Add Boundary Conditions

We have the system

$$\begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{12} & \cdots & \mathbf{K}_{1,n-1} & \mathbf{K}_{1n} \\ \mathbf{K}_{21} & \mathbf{K}_{22} & \cdots & \mathbf{K}_{2,n-1} & \mathbf{K}_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{K}_{n-1,1} & \mathbf{K}_{n-1,2} & \cdots & \mathbf{K}_{n-1,n-1} & \mathbf{K}_{n-1,n} \\ \mathbf{K}_{n1} & \mathbf{K}_{n2} & \cdots & \mathbf{K}_{n,n-1} & \mathbf{K}_{nn} \end{bmatrix} \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_{n-1} \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{n-1} \\ \mathbf{f}_n \end{bmatrix}$$

Inserting the boundary conditions $\hat{y}_1 = a$ and $\hat{y}_n = b$

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ \mathbf{K}_{21} & \mathbf{K}_{22} & \cdots & \mathbf{K}_{2,n-1} & \mathbf{K}_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{K}_{n-1,1} & \mathbf{K}_{n-1,2} & \cdots & \mathbf{K}_{n-1,n-1} & \mathbf{K}_{n-1,n} \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_{n-1} \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} a \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{n-1} \\ b \end{bmatrix}$$

What happened with the boundary integrals?

Recall how we defined $\delta\mathbf{y}$, $\delta\mathbf{y}_a$ and $\delta\mathbf{y}_b$. The substitution we just made,

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ \mathbf{K}_{21} & \mathbf{K}_{22} & \cdots & \mathbf{K}_{2,n-1} & \mathbf{K}_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{K}_{n-1,1} & \mathbf{K}_{n-1,2} & \cdots & \mathbf{K}_{n-1,n-1} & \mathbf{K}_{n-1,n} \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_{n-1} \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} a \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{n-1} \\ b \end{bmatrix},$$

is a consequence of how we set up our test functions. Remember,

$$\begin{aligned} \delta\mathbf{y} &\equiv \begin{bmatrix} 0 & \delta y_2 & \cdots & \delta y_{n-1} & 0 \end{bmatrix} \\ \delta\mathbf{y}_a &\equiv \begin{bmatrix} \delta y_1 & 0 & \cdots & 0 & 0 \end{bmatrix} \\ \delta\mathbf{y}_b &\equiv \begin{bmatrix} 0 & 0 & \cdots & 0 & \delta y_n \end{bmatrix} \end{aligned}$$

The trick is to ignore the zeros for the governing equation during assembly and add them in again after the assembly.

Remember to Add Boundary Conditions

The rest is basically, just linear algebra one-on-one, we move "known" parts to the right-hand side of the equation. This yields,

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & \mathbf{K}_{22} & \cdots & \mathbf{K}_{2,n-1} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \mathbf{K}_{n-1,2} & \cdots & \mathbf{K}_{n-1,n-1} & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_{n-1} \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} a \\ \mathbf{f}_2 - \mathbf{K}_{21}\, a - \mathbf{K}_{2n}\, b \\ \vdots \\ \mathbf{f}_{n-1} - \mathbf{K}_{n-1,1}\, a - \mathbf{K}_{n-1,n}\, b \\ b \end{bmatrix}$$

We are now ready to use our favorite linear system solver on this system.

Remember to Add Boundary Conditions

To summarize the recipe: The boundary conditions are

$$\hat{y}_1 = a$$
$$\hat{y}_n = b$$

We make the modifications to $\mathbf{f}$ and $\mathbf{K}$ as follows

$$f_i \leftarrow f_i - \mathbf{K}_{i1} a - \mathbf{K}_{in} b \quad \forall i \in 1, \ldots, n$$

and

$$\mathbf{K}_{1i} \leftarrow \begin{cases} 1 & \text{if } i = 1 \\ 0 & \text{otherwise} \end{cases}$$

similar for $\mathbf{K}_{ni}$. This can be done more elegantly in matrix notation.

Boundary Conditions in Matrix Form

Assume a discrete model

$$\mathbf{K}\hat{y} = \mathbf{f}$$

where $\hat{y}, \mathbf{f} \in \mathbb{R}^n$ and $\mathbf{K} \in \mathbb{R}^{n \times n}$. Now we wish to apply Dirichlet boundary conditions

$$\mathcal{D} \equiv \{i | \hat{y}_i = d_i\}$$

then

$$\mathcal{F} \equiv \{i | i \notin \mathcal{D}\}$$

Make partitioning of original system

$$\begin{bmatrix} \mathbf{K}_{\mathcal{F}\mathcal{F}} & \mathbf{K}_{\mathcal{F}\mathcal{D}} \\ \mathbf{K}_{\mathcal{D}\mathcal{F}} & \mathbf{K}_{\mathcal{D}\mathcal{D}} \end{bmatrix} \begin{bmatrix} \hat{y}_{\mathcal{F}} \\ \hat{y}_{\mathcal{D}} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_{\mathcal{F}} \\ \mathbf{f}_{\mathcal{D}} \end{bmatrix}$$

Inserting the Boundary Condition

Insert Dirichlet boundary conditions

$$\mathbf{I}\hat{y}_{\mathcal{D}} = \mathbf{d}$$

Into partitioned system

$$\begin{bmatrix} \mathbf{K}_{\mathcal{F}\mathcal{F}} & \mathbf{K}_{\mathcal{F}\mathcal{D}} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \hat{y}_{\mathcal{F}} \\ \hat{y}_{\mathcal{D}} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_{\mathcal{F}} \\ \mathbf{d} \end{bmatrix}$$

Resulting in the reduced system

$$\mathbf{K}_{\mathcal{F}\mathcal{F}}\hat{y}_{\mathcal{F}} = \mathbf{f}_{\mathcal{F}} - \mathbf{K}_{\mathcal{F}\mathcal{D}}\mathbf{d}$$

Notice that boundary conditions end up as extra term on right hand side (extra source term)

Where did the Elements go?

Its called the finite element method but we have derived an approximate solution in a global setting.

$$\int_{x_1}^{x_n} \frac{\partial \mathbf{N}^T}{\partial x} \frac{\partial \mathbf{N}}{\partial x} \hat{y} dx + \int_{x_1}^{x_n} \mathbf{N}^T c dx = \sum_e \left(\int_{x_i}^{x_j} \frac{\partial (\mathbf{N}^e)^T}{\partial x} \frac{\partial \mathbf{N}^e}{\partial x} \hat{y}^e dx + \int_{x_i}^{x_j} (\mathbf{N}^e)^T c dx \right) = 0$$

where e is the element between two consecutive nodes x_i and x_j.

The Local Shape Functions

We have

$$\mathbf{N}^e(x) = \begin{bmatrix} N_i^e(x) & N_j^e(x) \end{bmatrix}$$
$$\hat{y}^e = \begin{bmatrix} \hat{y}_i \\ \hat{y}_j \end{bmatrix}$$

where the local shape functions are

$$N_i^e = \frac{x_j - x}{x_j - x_i} \quad \text{and} \quad N_j^e = \frac{x - x_i}{x_j - x_i}$$

The Spatial Derivatives

So assuming equidistant x_i's we have $\Delta x = x_j - x_i$ and

$$N_i^e = \frac{x_j - x}{\Delta x} \quad \text{and} \quad N_j^e = \frac{x - x_i}{\Delta x}$$

Further

$$\frac{\partial N_i^e}{\partial x} = \frac{-1}{\Delta x} \quad \text{and} \quad \frac{\partial N_j^e}{\partial x} = \frac{1}{\Delta x}$$

which are constants independent of x

The Finite Elements

So

$$\int_{x_i}^{x_j} \frac{\partial(\mathbf{N}^e)^T}{\partial x} \frac{\partial \mathbf{N}^e}{\partial x} \hat{y}^e dx = \frac{1}{\Delta x^2} \left(\begin{bmatrix} -1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \end{bmatrix} \right) \hat{y}^e \int_{x_i}^{x_j} dx$$

$$= \underbrace{\frac{1}{\Delta x} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}}_{\mathbf{K}^e} \hat{y}^e$$

and

$$\mathbf{f}^e = -\int_{x_i}^{x_j} (\mathbf{N}^e)^T c dx$$

Now for this educational example we simply set $c(x) = 0$ so $\mathbf{f}^e = \mathbf{0}$

Assembling local Elements into a Global System

So we finally have

$$\mathbf{K}\hat{y} = \sum_e \mathbf{K}^e \hat{y}^e = \sum_e \mathbf{f}^e = \mathbf{f}$$

Or more algorithmically we have ...

The Computer Scientist at Work

The Final Assembly Algorithm

for all e
  $\mathbf{K}^e \leftarrow$ **compute element matrix**
  $\mathbf{f}^e \leftarrow$ **compute element vector**
next e
$\mathbf{K} \leftarrow$ **zeroes**(n, n)
$\mathbf{f} \leftarrow$ **zeroes**$(n, 1)$
for all e
  $(i, j) \leftarrow$ **local to global**$(1, 2)$
  $\mathbf{K}_{ii} \leftarrow \mathbf{K}_{ii} + \mathbf{K}^e_{11}$
  $\mathbf{K}_{ij} \leftarrow \mathbf{K}_{ij} + \mathbf{K}^e_{12}$
  $\mathbf{K}_{ji} \leftarrow \mathbf{K}_{ji} + \mathbf{K}^e_{21}$
  $\mathbf{K}_{jj} \leftarrow \mathbf{K}_{jj} + \mathbf{K}^e_{22}$
  $\mathbf{f}_i \leftarrow \mathbf{f}_i + \mathbf{f}^e_1$
  $\mathbf{f}_j \leftarrow \mathbf{f}_j + \mathbf{f}^e_2$
next e

The Boundary Condition Algorithm

Given the boundary conditions

$$\mathcal{D} = \{i | \hat{y}_i = \mathbf{d}_i\}$$

Make the modifications

for all $i \in \mathcal{D}$
  $\mathbf{f}_i \leftarrow \mathbf{d}_i$
  for all $j \notin \mathcal{D}$
    $\mathbf{f}_j \leftarrow \mathbf{f}_j - \mathbf{K}_{ji}\mathbf{d}_i$
  next j
  $\mathbf{K}_{i*} \leftarrow 0$
  $\mathbf{K}_{*i} \leftarrow 0$
  $\mathbf{K}_{ii} \leftarrow 1$
next i

The Final Computation

Since $\hat{y}_{\mathcal{D}}$ is given we only solve the reduced system

$$\mathbf{K}_{\mathcal{F}\mathcal{F}}\hat{y}_{\mathcal{F}} = \mathbf{f}_{\mathcal{F}}$$

where $\mathcal{F} \equiv \{j | j \notin \mathcal{D}\}$.

Let us go to 2D

Lets do the fast track this time

- STEP 1: Define Elements
- STEP 2: Choose Test and Shape Functions
- STEP 3: Solve Element wise Integrals
- STEP 4: Do Assembly of Elements
- STEP 5: Apply Boundary Conditions
- STEP 6: Compute Solution

The Problem

We wish to find the unknown function $u(\mathbf{p}) : \mathbb{R}^2 \mapsto \mathbb{R}$ on some domain $\mathbf{p} = \begin{bmatrix} x & y \end{bmatrix}^T \in \Omega \in \mathbb{R}^2$. All we know is some differential equation

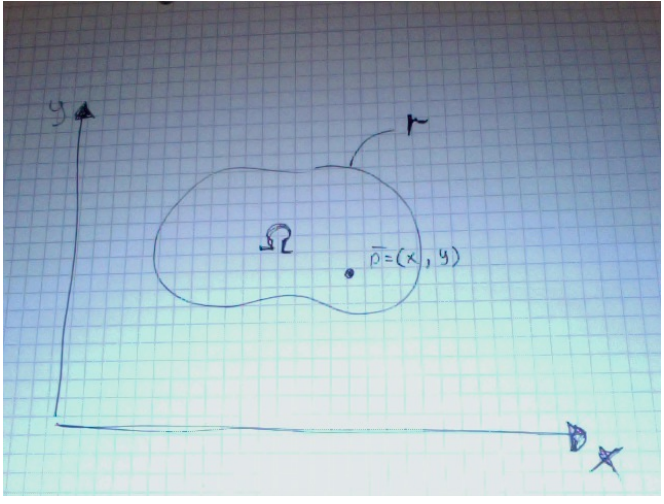
$$\frac{\partial^2 u(\mathbf{p})}{\partial x^2} + \frac{\partial^2 u(\mathbf{p})}{\partial y^2} = c(\mathbf{p})$$

where $c(\mathbf{p}) : \mathbb{R}^2 \mapsto \mathbb{R}$ is some known function. Additionally, we are given some boundary conditions

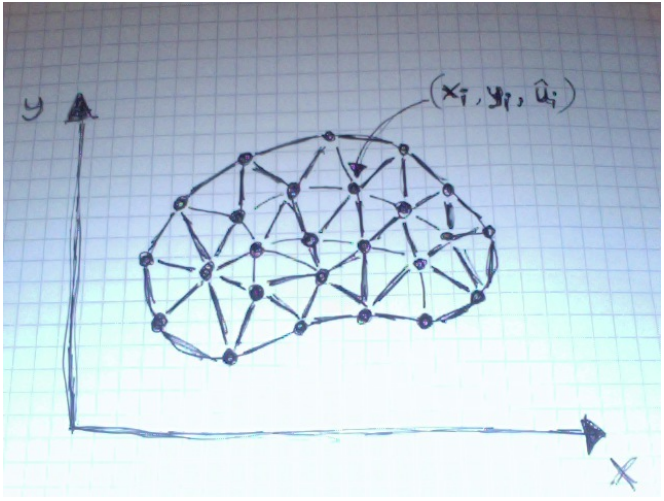
$$u(\mathbf{p}) = a(\mathbf{p})$$

For all $\mathbf{p} \in \Gamma \equiv \partial\Omega$. Now we will do this using the finite element method (FEM).

The 2D Domain

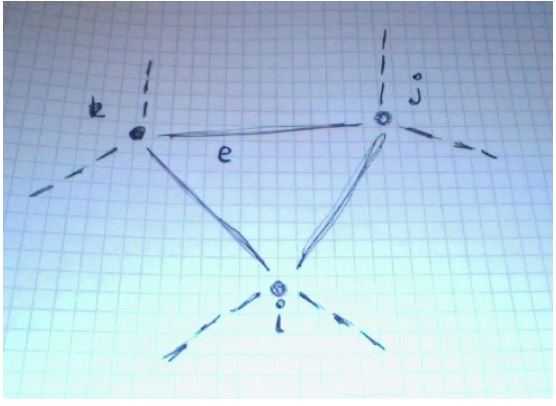


The 2D Computational Mesh



STEP 1: Define Elements

We have triangle elements



Element e consists of the nodes i, j, and k in counterclockwise order.

STEP 2: Choose Test and Shape Functions

Let $\mathbf{p}(x, y) \in \Omega^e$ then we want to find the approximation

$$u(\mathbf{p}) \approx \hat{u}(\mathbf{p}) = \begin{bmatrix} N_i^e(\mathbf{p}) & N_j^e(\mathbf{p}) & N_k^e(\mathbf{p}) \end{bmatrix} \begin{bmatrix} \hat{u}_i^e \\ \hat{u}_j^e \\ \hat{u}_k^e \end{bmatrix} = \mathbf{N}^e \hat{u}^e$$

and we choose test functions to be $v^e(\mathbf{p}) = w^e(\mathbf{p}) = \mathbf{N}^e \delta u^e$.

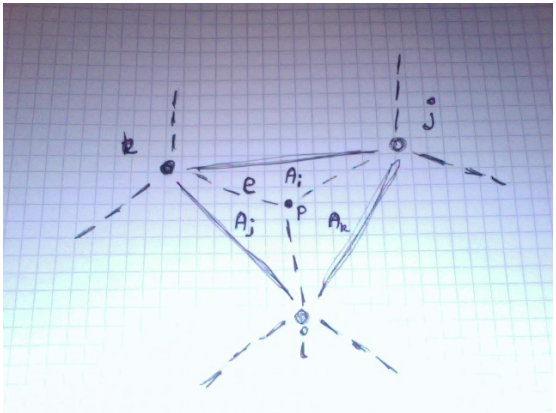
For a triangle, we will use barycentric coordinates also known as area-weighted coordinates.

$$N_i^e = \frac{A_i^e}{A^e}, \; N_j^e = \frac{A_j^e}{A^e}, \text{ and } N_k^e = \frac{A_k^e}{A^e}$$

Here A's are triangle areas and $A^e = A_i^e + A_j^e + A_k^e$ total area.

Barycentric Coords ≡ Area Weighted Coords

Geometrically that looks like this



The Local Shape Functions

We can use cross-products to quickly compute triangle areas

$$N_i^e = \frac{(\mathbf{p}_k - \mathbf{p}_j) \times (\mathbf{p} - \mathbf{p}_j)}{2A^e}$$
$$N_j^e = \frac{(\mathbf{p}_i - \mathbf{p}_k) \times (\mathbf{p} - \mathbf{p}_k)}{2A^e}$$
$$N_k^e = \frac{(\mathbf{p}_j - \mathbf{p}_i) \times (\mathbf{p} - \mathbf{p}_i)}{2A^e}$$

where $A^e = \frac{1}{2}\ (\mathbf{p}_j - \mathbf{p}_i) \times (\mathbf{p}_k - \mathbf{p}_i)$. Observe these are linear functions in x and y.

Remembering how the Cross-Product Works

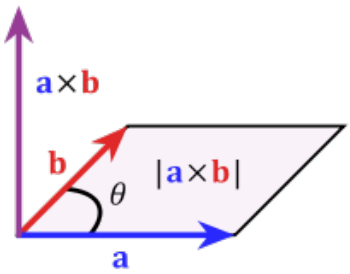
- In 2D the cross-product gives a scalar. Given $\mathbf{a}, \mathbf{b} \in \mathbb{R}^2$ then

$$\mathbf{a} \times \mathbf{b} \equiv \det \begin{bmatrix} \mathbf{a}^T \\ \mathbf{b}^T \end{bmatrix}$$

- In 3D the cross-product gives a vector. Given $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$ then

$$\mathbf{a} \times \mathbf{b} \equiv \det \begin{bmatrix} \hat{i} & \hat{j} & \hat{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{bmatrix}$$

  In 3D one needs to compute the length of the vector to get to the area measure.



The Spatial Derivatives

So we find by differentiation of the cross-product definition that,

$$\frac{\partial N_i^e}{\partial x} = -\frac{(\mathbf{p}_k - \mathbf{p}_j)_y}{2A^e}$$
$$\frac{\partial N_i^e}{\partial y} = \frac{(\mathbf{p}_k - \mathbf{p}_j)_x}{2A^e}$$

The subscripts x and y refers to the component of the vectors $(\mathbf{p}_k - \mathbf{p}_j)$ and $(\mathbf{p}_k - \mathbf{p}_j)$. We can derive similar expressions for N_j^e and N_k^e, we leave this for a home exercise. Observe the computed derivatives above are constants independent of x and y.

A Geometric Approach for The Spatial Derivatives

Let us try this differently, let us go back to the definition of the local shape function

$$N_i^e(\mathbf{p}) \equiv \frac{A_i^e(\mathbf{p})}{A^e}$$

The gradient would then be

$$\nabla N_i^e \equiv \frac{\nabla A_i^e(\mathbf{p})}{A^e}$$

where $\mathbf{p} \equiv (x, y)^T$ is some point inside the triangle.
Observation: we only need to understand what the gradient of the triangle area A_i^e means. That is $\nabla A_i^e(\mathbf{p})$.

The gradient of a Triangle Area using Geometry

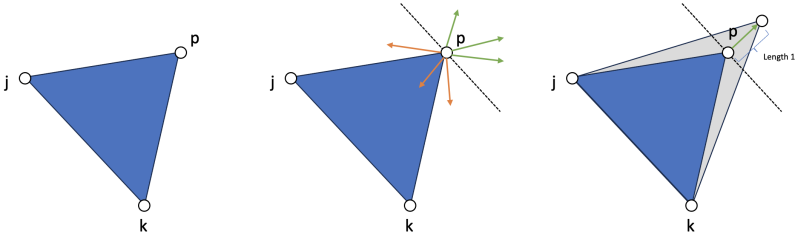
The gradient tells us two things

- What direction to move the corner point $\mathbf{p}$ of the triangle A_i^e in to make the area value grow quickest
- Further, it tells us how large the growth will be if we move one unit in the direction of the gradient.

Let us try to just "draw" that up!

The gradient of a Triangle Area using Geometry

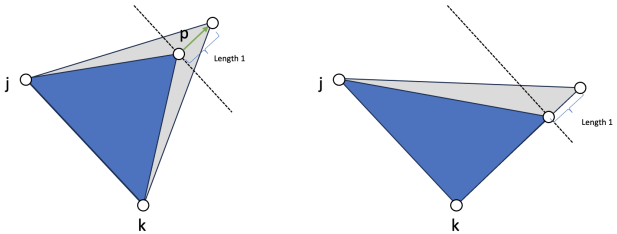
Any green direction on the upper side of the dashed line will grow the blue area.



The green arrow that is orthogonal to the dashed line will grow the area quickest. The change in area, when we take one unit step in that direction, gives us the magnitude of the gradient. That is the grey area.

The gradient of a Triangle Area using Geometry

To compute the area of the grey area we just imaginary push the p-vertex along the dashed line to make a right triangle. This will not change the area of the blue or grey triangles.



From the drawing, we immediately see that the grey area is half of the length of $i - k$ edge.

The gradient of a Triangle Area using Geometry

Let us recap in math. The unit direction, $\mathbf{d}$, that is orthogonal to the dashed line, we get from hatting the vector going from $\mathbf{p}_j$ to $\mathbf{p}_k$.

$$\mathbf{d} \equiv -\frac{\widehat{\mathbf{p}_j - \mathbf{p}_k}}{\| \mathbf{p}_j - \mathbf{p}_k \|}$$

To get the gradient we must multiply this direction with the value of the grey area,

$$\begin{aligned}\nabla N_i^e &\equiv \frac{1}{2} \| \mathbf{p}_j - \mathbf{p}_k \| \mathbf{d} \\ &= -\frac{1}{2} \| \mathbf{p}_j - \mathbf{p}_k \| \frac{\widehat{\mathbf{p}_j - \mathbf{p}_k}}{\| \mathbf{p}_j - \mathbf{p}_k \|} = -\frac{1}{2} \mathbf{p}_j \hat{-} \mathbf{p}_k\end{aligned}$$

Observe we used the hat operator to rotate a vector 90 degrees CCW.

STEP 3: Solve the Element-wise Integrals

So we have

$$\int_{\Omega^e} v^e \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - c \right) d\Omega + \int_{\Gamma^e} w^e \left(u - a \right) d\Gamma = 0$$

Remember that by design the test functions v and w are defined such that $v(\mathbf{p})$ is arbitrary for all $\mathbf{p} \in \Omega$ and zero elsewhere and $w(\mathbf{p})$ is arbitrary for all $\mathbf{p} \in \Gamma$ and zero elsewhere. That means v and w are complementary so we will ignore the boundary term for now and concentrate on the governing term. That is,

$$\int_{\Omega^e} v^e \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - c \right) d\Omega = 0$$

Getting the Local Element Matrices

Substitution of our choices and integration by parts etc give us

$$\mathbf{K}^e \hat{u}^e = \mathbf{f}^e$$

where

$$\mathbf{K}^e = \left(\left(\frac{\partial \mathbf{N}^e}{\partial x} \right)^T \frac{\partial \mathbf{N}^e}{\partial x} + \left(\frac{\partial \mathbf{N}^e}{\partial y} \right)^T \frac{\partial \mathbf{N}^e}{\partial y} \right) A^e$$

$$\mathbf{f}^e = -\int_{\Omega^e} (\mathbf{N}^e)^T c d\Omega$$

Now for our example we assume $c = 0$ and we will have closed form formulas.

STEP 4: Do Assembly of Elements

Same as in 1D

STEP 5: Apply Boundary Conditions

Same as in 1D

(If conditions are point wise)

STEP 6: Compute Solution

Same as in 1D

The Boundary Condition Term

So if boundary conditions are not pointwise we have to look at

$$\int_{\Gamma^e} w^e \left(u - a\right) d\Gamma = 0$$

Doing the same algebraic manipulations as before we end up with

$$\int_{\Gamma^e} \left(\mathbf{N}^e\right)^T \left(\mathbf{N}^e \hat{u}^e - a\right) d\Gamma = 0$$

Why is this different from the 1D example?

- The boundary condition is applied along the whole boundary

Observe that the equation will give a result like

$$\mathbf{u}^e = \mathbf{d}^e$$

Its Really Just Like Pointwise Conditions

Knowing that we end up with

$$\mathbf{u}^e = \mathbf{d}^e$$

for the element boundary

- Compute boundary terms for all elements with a boundary
- Assemble all boundary terms into one boundary vector
- Apply the assembled boundary vector as point-wise boundary conditions to assembled system.

2D Elasticity

STEPS 1-2 and 4-6

We use the exact same setup as for the previous 2D example

- 2D domain
- Triangle elements
- Area weighted coordinates for shape functions
- Point-wise boundary conditions

Now all that is missing is the problem definition and how to phrase it as an element integral.

The Problem and Element Integral

The weak form integral of Cauchy's equation (the equivalent of Newton 2. Law for continuum) results in

$$\int_{\Omega^e} (\delta\epsilon)^T \sigma d\Omega = (\delta \mathbf{u}^e)^T \mathbf{f}^e$$

where

- σ is the Cauchy stress tensor written as a 3-dimensional vector
- $\delta\epsilon$ is the virtual Cauchy strain tensor written as a 3-dimensional vector
- $\delta\mathbf{u}^e$ is the virtual displacement vector of dimension 6
- $\mathbf{f}^e$ is the external body force vector of dimension 6

As usual, we treat boundary conditions as point-wise and can ignore them for now. Observe we have "cheated" with the right-hand-side and ignore a proper FEM derivation for the sake of brevity.

The Virtual Displacements

Given nodal virtual displacements $\delta\mathbf{u}^e$ we can use our shape functions

$$\delta\mathbf{u} = \mathbf{N}^e \delta\mathbf{u}^e$$

where $\mathbf{N}$ is a 2-by-6 dimensional matrix given by

$$\mathbf{N}^e = \begin{bmatrix} N_i^e \mathbf{I}_{2\times 2} & N_j^e \mathbf{I}_{2\times 2} & N_k^e \mathbf{I}_{2\times 2} \end{bmatrix}$$

and the N's are the barycentric coordinates introduced previously with one small change. Observe that $\mathbf{N}$ changed dimensions from 1D to 2D. This is why $\mathbf{I}_{2\times 2}$ terms pop up.

The Virtual Strain

Strain, ϵ, is a function of the displacement field thus by definition

$$\delta\epsilon = \mathbf{S}\delta\mathbf{u} = \underbrace{\mathbf{S}\mathbf{N}^e}_{\mathbf{B}}\delta\mathbf{u}^e = \mathbf{B}\delta\mathbf{u}^e$$

where $\mathbf{S}$ is the differential operator of dimension 3-by-2

$$\mathbf{S} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}$$

and so $\mathbf{B}$ is of dimension 3-by-6

Linear Elasticity

If one imposes the condition that σ is symmetric then the stress tensor can be saved in a 3-dimensional vector

$$\sigma = \begin{bmatrix} \sigma_{11} & \sigma_{22} & \sigma_{12} \end{bmatrix}^T$$

A similar trick can be used for a symmetric strain tensor

$$\epsilon = \begin{bmatrix} \epsilon_{11} & \epsilon_{22} & \gamma_{12} \end{bmatrix}^T$$

The stress tensor may be related to the Cauchy strain tensor by the linear relation

$$\sigma = \mathbf{D}\epsilon = \mathbf{DBu}^e$$

where $\mathbf{D}$ is a 3-by-3 elasticity matrix.

Substitution of the Math

If we substitute into our element integrals we have

$$\int_{\Omega^e} (\delta \mathbf{u}^e)^T \mathbf{B}^T \mathbf{D} \mathbf{B} \mathbf{u}^e d\Omega = (\delta \mathbf{u}^e)^T \mathbf{f}^e$$

This must hold for all virtual displacements so

$$\int_{\Omega^e} \mathbf{B}^T \mathbf{D} \mathbf{B} \mathbf{u}^e d\Omega = \mathbf{f}^e$$

which we can solve

$$\underbrace{\mathbf{B}^T \mathbf{D} \mathbf{B} A^e}_{\mathbf{K}^e} \mathbf{u}^e = \mathbf{f}^e$$

That is it!

A Few Implementation Tips

For aluminium Young modulus is $E = 69 \cdot 10^9$ Pa and Poisson ratio is $\nu = 0.35$ and the Elasticity matrix would be given by

$$\mathbf{D} = \frac{E}{(1-\nu^2)} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{(1-\nu)}{2} \end{bmatrix}$$

and

$$\mathbf{B} = \begin{bmatrix} \begin{bmatrix} \frac{\partial N_i^e}{\partial x} & 0 \\ 0 & \frac{\partial N_i^e}{\partial y} \\ \frac{\partial N_i^e}{\partial y} & \frac{\partial N_i^e}{\partial x} \end{bmatrix} & \begin{bmatrix} \frac{\partial N_j^e}{\partial x} & 0 \\ 0 & \frac{\partial N_j^e}{\partial y} \\ \frac{\partial N_j^e}{\partial y} & \frac{\partial N_j^e}{\partial x} \end{bmatrix} & \begin{bmatrix} \frac{\partial N_k^e}{\partial x} & 0 \\ 0 & \frac{\partial N_k^e}{\partial y} \\ \frac{\partial N_k^e}{\partial y} & \frac{\partial N_k^e}{\partial x} \end{bmatrix} \end{bmatrix}$$

For steel, one would have $E = 210 \cdot 10^9$ Pa and $\nu = 0.29$.

Further Reading

- O. C. Zienkiewicz and R. L. Taylor: The Finite Element Method, volume 1, The Basis, Fifth Edition, Butterworth and Heinemann, 2002
- R. D. Cook, D. S. Malkus, M. E. Plesha and R. J. Witt: Concepts and applications of finite element analysis, fourth edition, John Wiley and Sons, 2002.
- Müller and M. Gross: Interactive virtual materials, Proceedings of Graphics Interface 2004.
- R. Schmedding and M. Teschner: Inversion Handling for Stable Deformable Modeling, The Visual Computer, vol. 24, no. 7-9 (CGI 2008 Special Issue), pp. 625-633, 2008.
- A. W. Bargteil, C. Wojtan, J. K. Hodgins, and G. Turk: A finite element method for animating large viscoplastic flow, ACM Trans. Graph. vol 26. no. 3. 2007.

Assignment

- Make an implementation of the 1D problem on page 4. Use the values

$$x_1 = 1, x_n = 2, a = 1, b = 2, c = 0$$
$$r = \sqrt[n-1]{2}, x_k = r^k \text{ for } 0 < k < n$$

  Start by sketching on paper showing how you expect $y(x)$ to look like.
- Discuss which parts of the algorithm are embarrassingly parallel. If you have time consider rewriting the parts that have race conditions to become embarrassingly parallel.
- Consider using an iterative method (like Conjugate Gradient) for solving the reduced system, can you suggest any optimizations? (Hint: do not assemble the global matrix).
- Implement your pseudo-code algorithms for your optimizations.

Assignment

- Prove the formula for the spatial cross products for all shape functions N_i, N_j and N_k on Page 46 are correct by deriving the equations.

Assignment

- Derive the formula for $\mathbf{K}^e$ and $\mathbf{f}^e$ on Page 53 in full detail for the 2D example on Page 38.
- Implement the 2D example on a rectangular domain that is 6-by-2 in size.
  - Set $a = 1$ on the left edge of the rectangle and $a = 2$ on the right size.
  - Use $c = 0$
  - Treat boundary conditions as point-wise conditions
- Discuss from a computer scientist's viewpoint which steps are the same and which steps are different compared to the 1D example.

Assignment

- Make an implementation of the 2D elastic assembly process using the closed form solution from Page 65 and 66.
- Discuss from a computer science viewpoint which steps are the same and which steps are different in the 2D elastic case from the previous 1D and 2D Poisson problems.
- Speculate how to write a general FEM computer program that could simulate any partial differential equation with point-wise boundary conditions.

Assignment

- For the 2D elasticity problem,

$$\int_{\Omega^e} (\delta\epsilon)^T \sigma d\Omega = \int_{\Gamma^e} (\delta \mathbf{u})^T \mathbf{f} dS$$

  where $\mathbf{f}$ is a given constant load (force per meter) use the FEM method to derive the $\mathbf{K}^e$ and $\mathbf{f}^e$ terms such that

$$\mathbf{K}^e \mathbf{u}^e = \mathbf{f}^e$$

- Based on your FEM derivation implement a rectangular solid 2D bar of dimension 6-by-2 meters and made of steel like material. Attach the left side of the bar to a wall (use boundary condition $u = 0$) and apply a downward nodal load on the right edge of the bar.

Assignment

- Examine the eigenvalue spectrum of the global stiffness matrix of your implementation from Page 72 before applying boundary conditions and after (Hint: write down how many zero eigenvalues you expect before and after applying boundary conditions and explain how the minimum eigenvalue is related to the boundary conditions after having applied these).
- Examine the fill pattern of the stiffness matrix (Hint: reflect on the shape of the fill-in and how the non-zero values in $\mathbf{K}$ are related to the mesh).

Assignment

Use your implementation from Page 72.

- Investigate what happens to the deformations when you vary the resolution and load on the bar. Try to find the best suitable mesh resolution. (Hint: it is critical that you have the right formula for $\mathbf{f}^e$ and do not use a "constant" load per node value, can you explain why?)
- From everyday life steel material appears to be noticeably incompressible. Examine whether your implementation has a volume (area in 2D) conservation (Hint: Explain the causes for your observations, and consider carefully what could have "gone" wrong).