Study Guide: Introduction to Finite Element Methods

Hans Petter Langtangen[1,2]

[1]Center for Biomedical Computing, Simula Research Laboratory
[2]Department of Informatics, University of Oslo

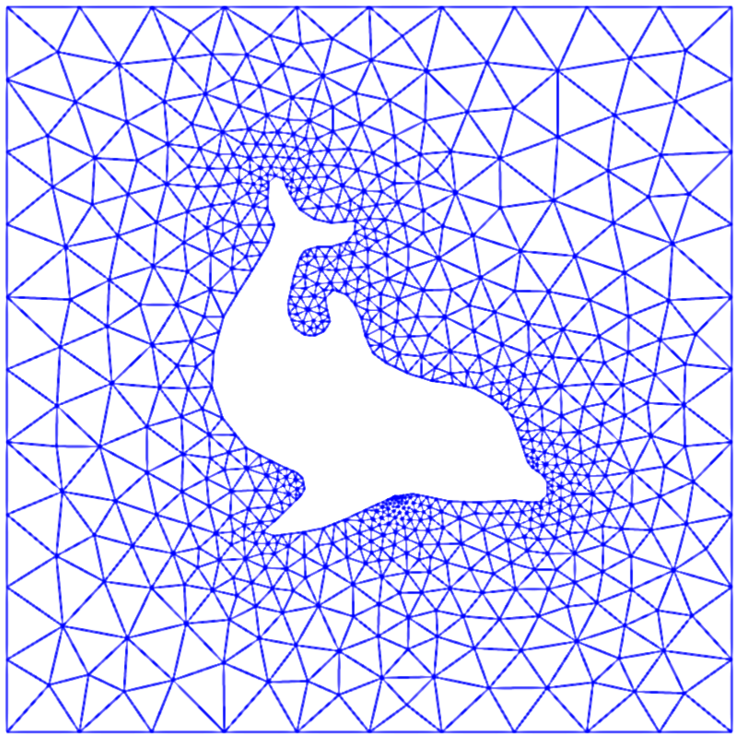
Sep 26, 2013

Contents

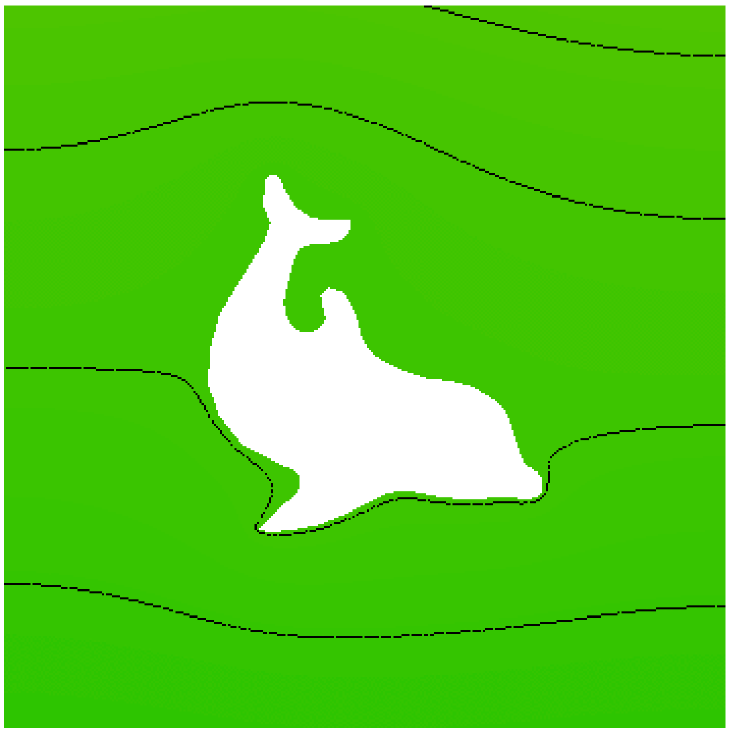
1 Why finite elements?

- Can with ease solve PDEs in domains with *complex geometry*
- Can with ease provide higher-order approximations
- Has (in simpler problems) a rigorus mathematical analysis framework (not much considered here - not powerful enough to uncover the serious limitations of the method in time-dependent problems and the necessary adjustments)

1.1 Domain for flow around a dolphin



1.2 The flow



1.3 Basic ingredients

- Transform the PDE problem to a *variational form*
- Define function approximation over *finite elements*
- Use a machinery to derive *linear systems*
- Solve linear systems

1.4 Learning strategy

- Start with approximation of functions
- Introduce finite element *approximations*
- See how this is applied to PDEs

Reason: the finite element method has many concepts and a jungle of details. This strategy minimizes the mixing of ideas, concepts, and technical details.

2 Approximation set-up

General idea of approximation:

$$u(x) = \sum_{i=0}^{N} c_i \varphi_i(x), \tag{1}$$

where

- $\varphi_i(x)$ are prescribed functions
- c_i, $i = 0, \ldots, N$ are unknown coefficients to be determined

How to determine c_i:

- least squares method
- projection or Galerkin method
- interpolation (or collocation) method

Our mathematical framework for doing this is phrased in a way such that it becomes easy to understand and use the FEniCS[1] software package for finite element computing.

2.1 Approximation of planar vectors

Problem. Given a two-dimensional vector $\boldsymbol{f} = (3, 5)$, find an approximation to $\boldsymbol{f}$ directed along a given line.

In vector space terminology: given a vector in a two-dimensional vector space, find an approximation in a one-dimensional vector space

$$V = \text{span}\{\boldsymbol{\psi}_0\}. \tag{2}$$

- $\boldsymbol{\psi}_0$ is a basis vector in the space V
- Seek $\boldsymbol{u} = c_0\boldsymbol{\psi}_0 \in V$
- Determine c_0 such that $\boldsymbol{u}$ is the "best" approximation to $\boldsymbol{f}$
- Visually, "best" is obvious
- For PDEs, "best" is not so obvious
- Define the error $\boldsymbol{e} = \boldsymbol{f} - \boldsymbol{u}$
- Define the (Eucledian) scalar product of two vectors: $(\boldsymbol{u}, \boldsymbol{v})$
- Define the norm of $\boldsymbol{e}$: $||\boldsymbol{e}|| = \sqrt{(\boldsymbol{e}, \boldsymbol{e})}$

[1] `http://fenicsproject.org`

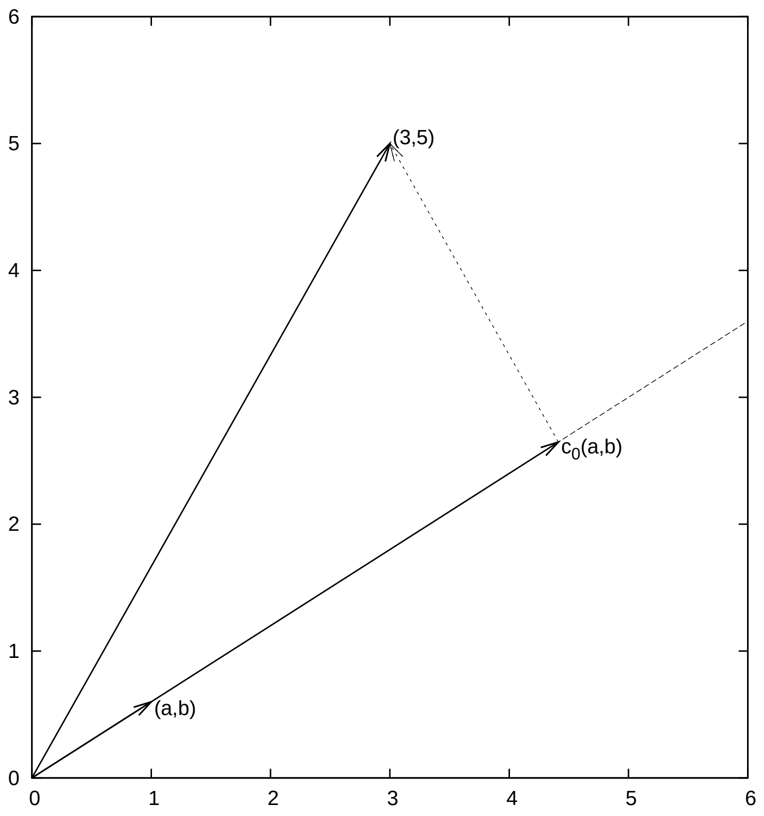


Figure 1: Approximation of a two-dimensional vector by a one-dimensional vector.

The least squares method.

- Idea: find c_0 such that $||\boldsymbol{e}||$ is minimized
- Actually, we always minimize $E = ||\boldsymbol{e}||^2$

$$\frac{\partial E}{\partial c_0} = 0\,.$$

Detailed mathematics:

$$E(c_0) = (\boldsymbol{e},\boldsymbol{e}) = (\boldsymbol{f},\boldsymbol{f}) - 2c_0(\boldsymbol{f},\boldsymbol{\psi}_0) + c_0^2(\boldsymbol{\psi}_0,\boldsymbol{\psi}_0) \tag{3}$$

$$\frac{\partial E}{\partial c_0} = -2(\boldsymbol{f},\boldsymbol{\psi}_0) + 2c_0(\boldsymbol{\psi}_0,\boldsymbol{\psi}_0) = 0\,. \tag{4}$$

$$c_0 = \frac{(\boldsymbol{f},\boldsymbol{\psi}_0)}{(\boldsymbol{\psi}_0,\boldsymbol{\psi}_0)}, \tag{5}$$

$$c_0 = \frac{3a + 5b}{a^2 + b^2}\,. \tag{6}$$

For later, we note that setting (4) to zero can be alternatively written as

$$(\boldsymbol{e}, \boldsymbol{\psi}_0) = 0\,. \tag{7}$$

The Galerkin or projection method.

- Backgrund: minimizing $||\boldsymbol{e}||^2$ implies that $\boldsymbol{e}$ is orthogonal to *any* vector $\boldsymbol{v}$ in the space V (visually clear, but can easily be computed too)
- Alternative idea: demand $(\boldsymbol{e}, \boldsymbol{v}) = 0, \quad \forall \boldsymbol{v} \in V$
- Equivalent statement: $(\boldsymbol{e}, \boldsymbol{\psi}_0) = 0$ (see notes for why)
- Insert $\boldsymbol{e} = \boldsymbol{f} - c_0\boldsymbol{\psi}_0$ and solve for c_0
- Same equation for c_0 and hence same solution

2.2 Approximation of general vectors

Given some vector $\boldsymbol{f}$, seek an approximation $\boldsymbol{u}$ in a vector space V of dimension $N+1$:

$$V = \text{span}\,\{\boldsymbol{\psi}_0, \ldots, \boldsymbol{\psi}_N\}\,.$$

- We have a set of linearly independent basis vectors $\boldsymbol{\psi}_0, \ldots, \boldsymbol{\psi}_N$
- Any $\boldsymbol{u} \in V$ can then be written as $\boldsymbol{u} = \sum_{j=0}^{N} c_j\boldsymbol{\psi}_j$

The least squares method. Idea: find $c_0, \ldots, c_N$ such that $E = ||\boldsymbol{e}||^2$ is minimized, $\boldsymbol{e} = \boldsymbol{f} - \boldsymbol{u}$.

$$\begin{aligned} E(c_0, \ldots, c_N) &= (\boldsymbol{e}, \boldsymbol{e}) = (\boldsymbol{f} - \sum_j c_j\boldsymbol{\psi}_j, \boldsymbol{f} - \sum_j c_j\boldsymbol{\psi}_j) \\ &= (\boldsymbol{f}, \boldsymbol{f}) - 2\sum_{j=0}^{N} c_j(\boldsymbol{f}, \boldsymbol{\psi}_j) + \sum_{p=0}^{N}\sum_{q=0}^{N} c_p c_q(\boldsymbol{\psi}_p, \boldsymbol{\psi}_q)\,. \end{aligned}$$

$$\frac{\partial E}{\partial c_i} = 0, \quad i = 0, \ldots, N\,.$$

After a bit tedious and technical work with sums, we get a *linear system* $Ac = b$ or

$$\sum_{j=0}^{N} A_{i,j}c_j = b_i, \quad i = 0, \ldots, N,$$

where

$$A_{i,j} = (\boldsymbol{\psi}_i, \boldsymbol{\psi}_j), \tag{8}$$
$$b_i = (\boldsymbol{\psi}_i, \boldsymbol{f}) . \tag{9}$$

The Galerkin or projection method. Can be shown that minimizing $||\boldsymbol{e}||$ implies that $\boldsymbol{e}$ is orthogonal to all $\boldsymbol{v} \in V$:

$$(\boldsymbol{e}, \boldsymbol{v}) = 0, \quad \forall \boldsymbol{v} \in V,$$

which implies that $\boldsymbol{e}$ most be orthogonal to each basis vector (see notes):

$$(\boldsymbol{e}, \boldsymbol{\psi}_i) = 0, \quad i = 0, \ldots, N . \tag{10}$$

Inserting for $\boldsymbol{e}$ and ordering terms gives the same linear system as that in the least squares method, implying that the methods are equivalent. However, the linear system will not be the same when apply these principles to solve PDEs.

3 Approximation of functions

Let V be a *function space* spanned by a set of *basis functions* $\varphi_0, \ldots, \varphi_N$,

$$V = \text{span}\,\{\varphi_0, \ldots, \varphi_N\},$$

Any function $u \in V$ can be written as a linear combination of the basis functions:

$$u = \sum_{j\in\mathcal{I}} c_j\varphi_j, \quad \mathcal{I} = \{0, 1, \ldots, N\} \tag{11}$$

For now, we shall look at functions of a single variable x: $u = u(x)$, $\varphi_i = \varphi_i(x)$, $i \in \mathcal{I}$. Easy to generalize to the multi-variate case.

3.1 The least squares method

Try to extend the ideas from the vector case: compute error and minimize error norm.

What norm?

$$(f, g) = \int_\Omega f(x)g(x)\,dx . \tag{12}$$

Restrict attention to some domain Ω.

The error: $e = f - u$.

The squared norm of the error:

$$E = (e, e) = (f - u, f - u) = (f(x) - \sum_{j\in\mathcal{I}} c_j\varphi_j(x), f(x) - \sum_{j\in\mathcal{I}} c_j\varphi_j(x)) . \tag{13}$$

$$E(c_0,\ldots,c_N) = (f,f) - 2\sum_{j\in\mathcal{I}} c_j(f,\varphi_i) + \sum_{p\in\mathcal{I}}\sum_{q\in\mathcal{I}} c_pc_q(\varphi_p,\varphi_q)\,. \tag{14}$$

Minimizing E implies

$$\frac{\partial E}{\partial c_i} = 0,\quad i = in\mathcal{I}\,.$$

After computations identical to the vector case, we get a *linear system*

$$\sum_{j\in\mathcal{I}}^{N} A_{i,j}c_j = b_i,\quad i\in\mathcal{I}, \tag{15}$$

where

$$A_{i,j} = (\varphi_i,\varphi_j) \tag{16}$$
$$b_i = (f,\varphi_i)\,. \tag{17}$$

3.2 The Galerkin or projection method

As before, minimizing (e,e) is equivalent to

$$(e,v) = 0,\quad \forall v\in V, \tag{18}$$

which means (as before)

$$(e,\varphi_i) = 0,\quad i\in\mathcal{I}\,. \tag{19}$$

With the same algebra as in the multi-dimensional vector case, we get the same linear system as arose from the least squares method.

That is, the least squares and Galerkin/projection methods are again equivalent.

3.3 Example: linear approximation

Problem: approximate a parabola by a straight line.

$$V = \text{span}\,\{1,x\}\,.$$

That is, $\varphi_0(x) = 1$, $\varphi_1(x) = x$, and $N = 1$. We seek

$$u = c_0\varphi_0(x) + c_1\varphi_1(x) = c_0 + c_1x,$$

$$A_{0,0} = (\varphi_0, \varphi_0) = \int_1^2 1 \cdot 1 \, dx = 1, \tag{20}$$

$$A_{0,1} = (\varphi_0, \varphi_1) = \int_1^2 1 \cdot x \, dx = 3/2, \tag{21}$$

$$A_{1,0} = A_{0,1} = 3/2, \tag{22}$$

$$A_{1,1} = (\varphi_1, \varphi_1) = \int_1^2 x \cdot x \, dx = 7/3\,. \tag{23}$$

$$b_1 = (f, \varphi_0) = \int_1^2 (10(x-1)^2 - 1) \cdot 1 \, dx = 7/3, \tag{24}$$

$$b_2 = (f, \varphi_1) = \int_1^2 (10(x-1)^2 - 1) \cdot x \, dx = 13/3\,. \tag{25}$$

Solution of 2x2 linear system:

$$c_0 = -38/3, \quad c_1 = 10, \tag{26}$$

$$u(x) = 10x - \frac{38}{3}\,. \tag{27}$$

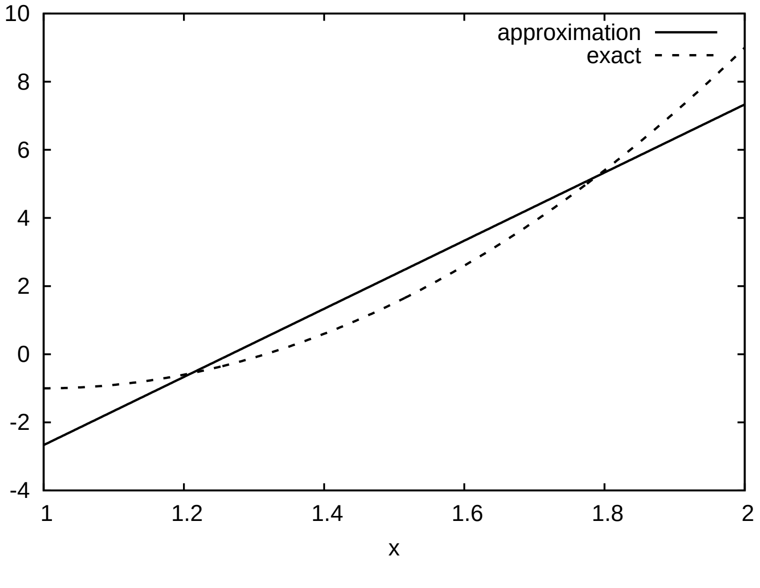


Figure 2: Best approximation of a parabola by a straight line.

3.4 Implementation of the least squares method

Consider symbolic computation of the linear system, where

- $f(x)$ is given as a `sympy` expression `f` (involving the symbol `x`),
- `phi` is a list of φ_i, $i \in \mathcal{I}$,
- `Omega` is a 2-tuple/list holding the domain Ω

```
import sympy as sm

def least_squares(f, phi, Omega):
    N = len(phi) - 1
    A = sm.zeros((N+1, N+1))
    b = sm.zeros((N+1, 1))
    x = sm.Symbol('x')
    for i in range(N+1):
        for j in range(i, N+1):
            A[i,j] = sm.integrate(phi[i]*phi[j],
                                  (x, Omega[0], Omega[1]))
            A[j,i] = A[i,j]
        b[i,0] = sm.integrate(phi[i]*f, (x, Omega[0], Omega[1]))
    c = A.LUsolve(b)
    u = 0
    for i in range(len(phi)):
        u += c[i,0]*phi[i]
    return u
```

Observe: symmetric coefficient matrix.

Compare f and u visually:

```
def comparison_plot(f, u, Omega, filename='tmp.pdf'):
    x = sm.Symbol('x')
    f = sm.lambdify([x], f, modules="numpy")
    u = sm.lambdify([x], u, modules="numpy")
    resolution = 401  # no of points in plot
    xcoor  = linspace(Omega[0], Omega[1], resolution)
    exact  = f(xcoor)
    approx = u(xcoor)
    plot(xcoor, approx)
    hold('on')
    plot(xcoor, exact)
    legend(['approximation', 'exact'])
    savefig(filename)
```

3.5 Perfect approximation

What if we add $\varphi_2 = x^2$ to the space V? That is, approximating a parabola by any parabola? (Hopefully we get the exact parabola!)

```
>>> from approx1D import *
>>> x = sm.Symbol('x')
>>> f = 10*(x-1)**2-1
>>> u = least_squares(f=f, phi=[1, x, x**2], Omega=[1, 2])
>>> print u
10*x**2 - 20*x + 9
>>> print sm.expand(f)
10*x**2 - 20*x + 9
```

Yes!

What if we use $\phi_i(x) = x^i$ for $i = 0, \ldots, N = 40$?

The output from `least_squares` is $c_i = 0$ for $i > 2$.

General result: if $f \in V$, least squares and Galerkin/projection gives $u = f$.

Proof: if $f \in V$, f can be expanded in terms of the basis functions, $f = \sum_{j\in\mathcal{I}} d_j\varphi_j$, for some coefficients d_i, $i \in \mathcal{I}$, and the right-hand side then has entries

$$b_i = (f, \varphi_i) = \sum_{j\in\mathcal{I}} d_j(\varphi_j, \varphi_i) = \sum_{j\in\mathcal{I}} d_j A_{i,j}\,.$$

The linear system $\sum_j A_{i,j}c_j = b_i$, $i \in \mathcal{I}$, is then

$$\sum_{j\in\mathcal{I}} c_j A_{i,j} = \sum_{j\in\mathcal{I}} d_j A_{i,j}, \quad i \in \mathcal{I},$$

which implies that $c_i = d_i$ for $i \in \mathcal{I}$.

3.6 Ill-conditioning

The previous computations were symbolic. What if we do numerical computing with `numpy` arrays?

exact	`sympy`	`numpy32`	`numpy64`
9	9.62	5.57	8.98
-20	-23.39	-7.65	-19.93
10	17.74	-4.50	9.96
0	-9.19	4.13	-0.26
0	5.25	2.99	0.72
0	0.18	-1.21	-0.93
0	-2.48	-0.41	0.73
0	1.81	-0.013	-0.36
0	-0.66	0.08	0.11
0	0.12	0.04	-0.02
0	-0.001	-0.02	0.002

- Column 2: The matrix and vector are converted to the `sympy.mpmath.fp.matrix` data structure and the `sympy.mpmath.fp.lu_solve` function is used to solve the system.

- Column 3: The matrix and vector are converted to `numpy` arrays with data type `numpy.float32` (single precision floating-point number) and solved by the `numpy.linalg.solve` function.
- Column 4: As column 3, but the data type is `numpy.float64` (double precision floating-point number).

Observations:

- Significant round-off errors in the numerical computations (!)
- But now visually appearant in a plot (!)

Reasons: The basis functions x^i become almost linearly dependent for large N.

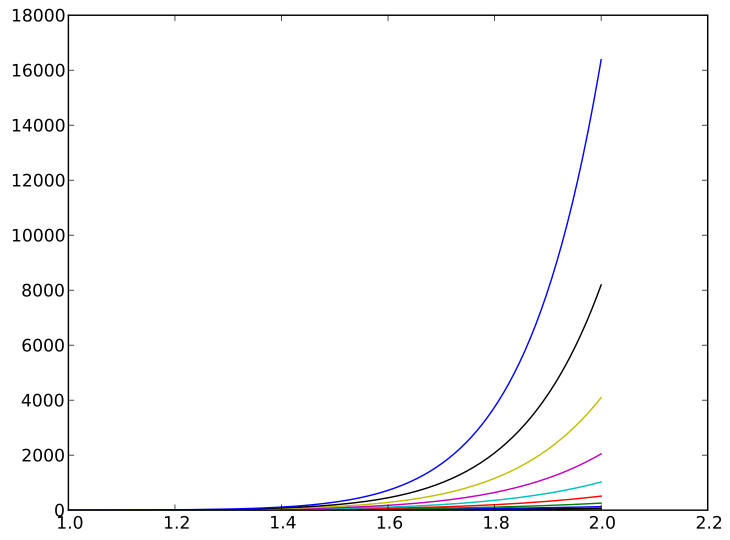


Figure 3: The 15 first basis functions x^i, $i = 0, \ldots, 14$.

Almost linearly dependent basis functions give almost singular matrices. Such matrices are said to be *ill conditioned*, and Gaussian elimination is then prone to round-off errors.

The basis $1, x, x^2, x^3, x^4, \ldots$ is a bad basis. Polynomials are fine as basis, but more orthogonal they are, $(\varphi_i, \varphi_j) \approx 0$, the better.

3.7 Fourier series

Consider

$$V = \text{span}\left\{\sin \pi x, \sin 2\pi x, \ldots, \sin(N+1)\pi x\right\}.$$

$$\varphi_i(x) = \sin((i+1)\pi x), \quad i \in \mathcal{I}\,.$$

```
N = 3
from sympy import sin, pi
phi = [sin(pi*(i+1)*x) for i in range(N+1)]
f = 10*(x-1)**2 - 1
Omega = [0, 1]
u = least_squares(f, phi, Omega)
comparison_plot(f, u, Omega)
```

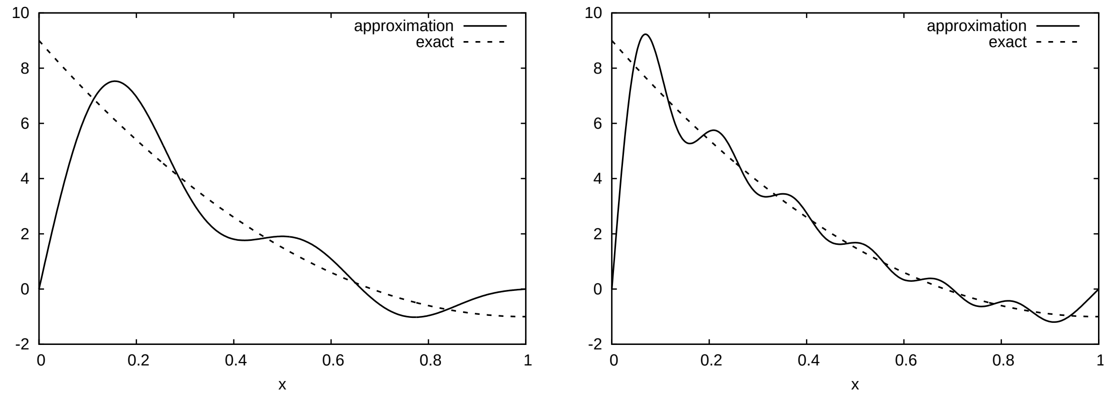


Figure 4: Best approximation of a parabola by a sum of 3 (left) and 11 (right) sine functions.

Considerably improvement by $N = 11$.

However, always discrepancy of $f(0) - u(0) = 9$ at $x = 0$, because all the $\varphi_i(0) = 0$ and hence $u(0) = 0$. Possible remedy:

$$u(x) = f(0)(1-x) + xf(1) + \sum_{j\in\mathcal{I}} c_j\varphi_j(x)\,. \tag{28}$$

The extra term is a strikingly good help to get a good approximation.

3.8 Orthogonal basis functions

This choice of sine functions as basis functions is popular because

- the basis functions are orthogonal: $(\varphi_i, \varphi_j) = 0$
- implying that $A_{i,j}$ is a diagonal matrix
- implying that $c_i = 2\int_0^1 f(x)\sin((i+1)\pi x)dx$

In general for an orthogonal basis, $A_{i,j}$ is diagonal and we can solve for c_i:

$$c_i = \frac{b_i}{A_{i,i}} = \frac{(f, \varphi_i)}{(\varphi_i, \varphi_i)}\,.$$

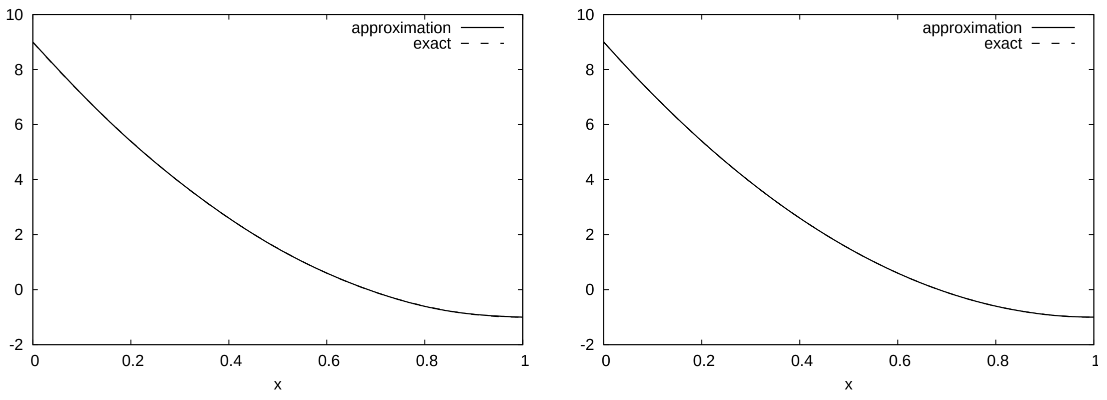


Figure 5: Best approximation of a parabola by a sum of 3 (left) and 11 (right) sine functions with a boundary term.

3.9 The collocation or interpolation method

Here is another idea:

- force $u(x_i) = f(x_i)$ at some selected *collocation* points x_i, $i \in \mathcal{I}$.
- Then u interpolates f.
- Called the *collocation method*

$$u(x_i) = \sum_{j\in\mathcal{I}} c_j\varphi_j(x_i) = f(x_i), \quad i \in \mathcal{I}, N\,. \tag{29}$$

This is a linear system,

$$\sum_{j\in\mathcal{I}} A_{i,j}c_j = b_i, \quad i \in \mathcal{I}, \tag{30}$$

with

$$A_{i,j} = \varphi_j(x_i), \tag{31}$$
$$b_i = f(x_i)\,. \tag{32}$$

No symmetric matrix: $\varphi_j(x_i) \neq \varphi_i(x_j)$ (in general).

With `points` as the collocation/interpolation points we can program the symbolic computations:

```
def interpolation(f, phi, points):
    N = len(phi) - 1
    A = sm.zeros((N+1, N+1))
    b = sm.zeros((N+1, 1))
    x = sm.Symbol('x')
    # Turn phi and f into Python functions
    phi = [sm.lambdify([x], phi[i]) for i in range(N+1)]
    f = sm.lambdify([x], f)
    for i in range(N+1):
        for j in range(N+1):
            A[i,j] = phi[j](points[i])
        b[i,0] = f(points[i])
    c = A.LUsolve(b)
    u = 0
    for i in range(len(phi)):
        u += c[i,0]*phi[i](x)
    return u
```

Features:

- +: no computation of integrals
- -: how to choose x_i?

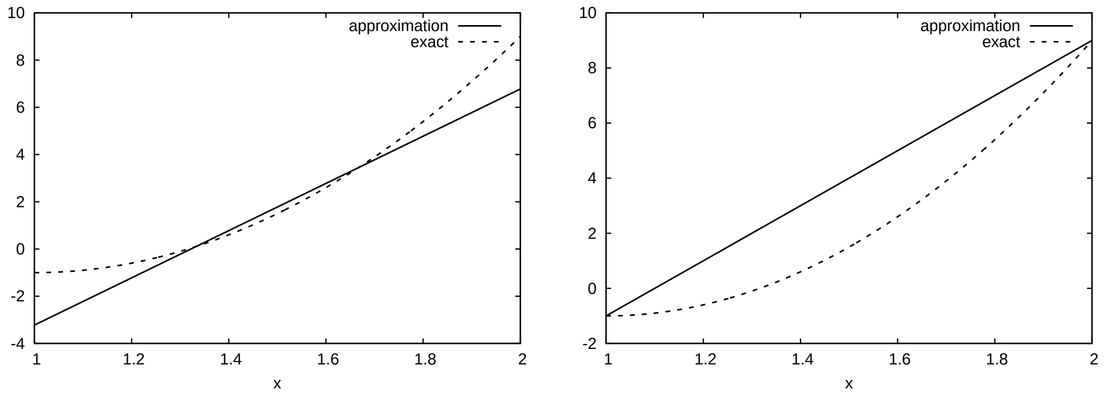


Figure 6: Approximation of a parabola by linear functions computed by two interpolation points: 4/3 and 5/3 (left) versus 1 and 2 (right).

Example.

3.10 Lagrange polynomials

Motivation:

- the interpolation/collocation method avoids integration
- with a diagonal matrix $A_{i,j} = \varphi_j(x_i)$ we can solve the linear system by hand

The *Lagrange interpolating polynomials* φ_j have the property that $\varphi_j(x_i) = 0$ for $i \neq 1$ and $\varphi_i(x_i) = 1$, so that $c_i = f(x_i)$ and hence

$$u(x) = \sum_{j\in\mathcal{I}} f(x_i)\varphi_i(x)\,. \qquad (33)$$

Formula:

$$\varphi_i(x) = \prod_{j=0,j\neq i}^{N} \frac{x - x_j}{x_i - x_j} = \frac{x - x_0}{x_i - x_0} \cdots \frac{x - x_{i-1}}{x_i - x_{i-1}} \frac{x - x_{i+1}}{x_i - x_{i+1}} \cdots \frac{x - x_N}{x_i - x_N}, \qquad (34)$$

```
def Lagrange_polynomial(x, i, points):
    p = 1
    for k in range(len(points)):
        if k != i:
            p *= (x - points[k])/(points[i] - points[k])
    return p
```

These basis functions are very much used in finite element methods.

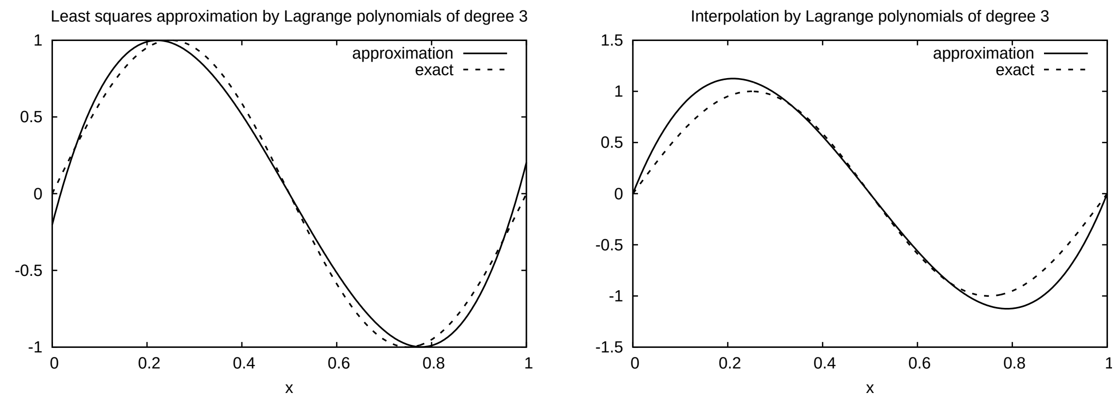


Figure 7: Approximation via least squares (left) and interpolation (right) of a sine function by Lagrange interpolating polynomials of degree 4.

Successful example.

Less successful example. Problem: strong oscillations near the boundaries for larger N values.

Remedy for strong oscillations. The oscillations can be reduced by a more clever choice of interpolation points, called the *Chebyshev nodes*:

$$x_i = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{2i+1}{2(N+1)}pi\right), \quad i = 0\ldots,N, \qquad (35)$$

on an interval $[a,b]$.

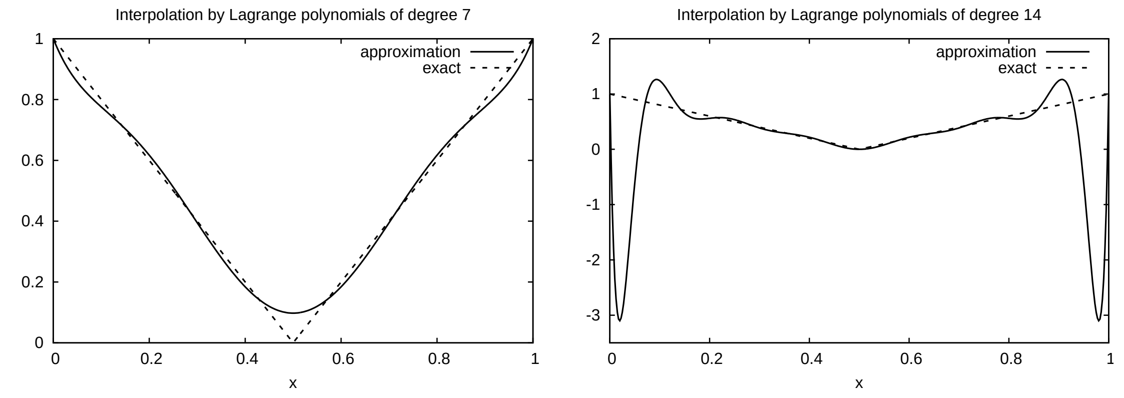


Figure 8: Interpolation of an absolute value function by Lagrange polynomials and uniformly distributed interpolation points: degree 7 (left) and 14 (right).

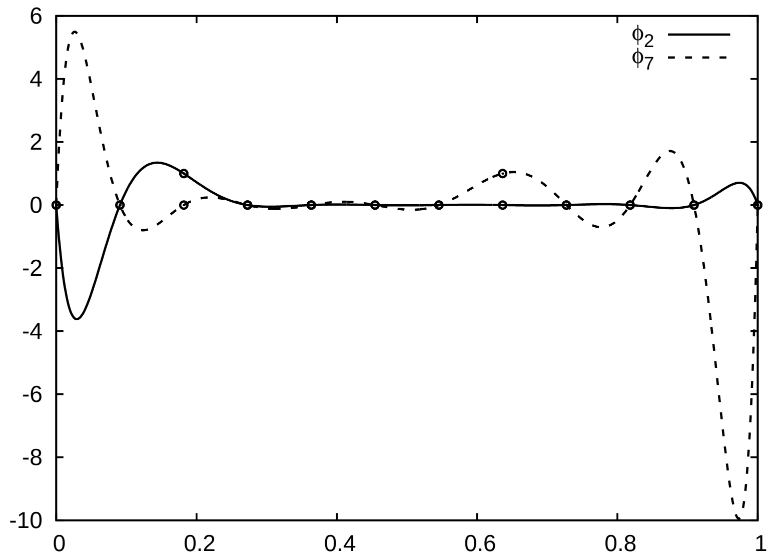


Figure 9: Illustration of the oscillatory behavior of two Lagrange polynomials for 12 uniformly spaced points (marked by circles).

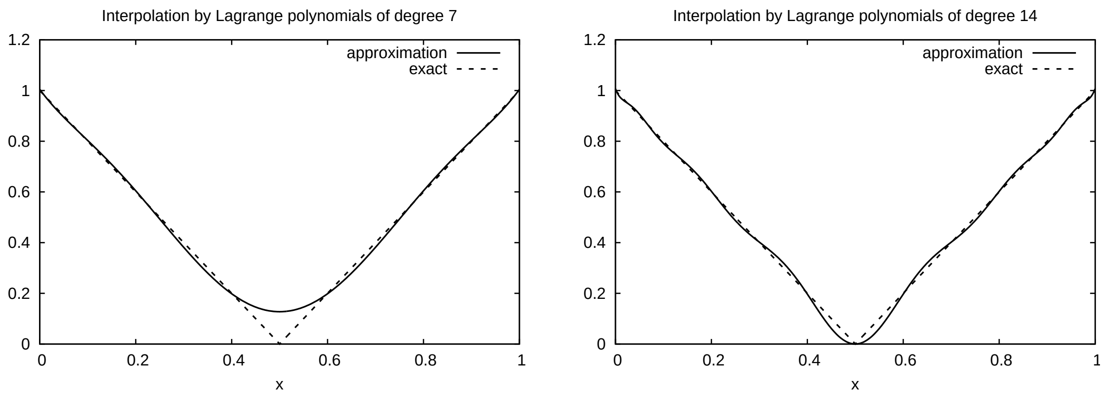


Figure 10: Interpolation of an absolute value function by Lagrange polynomials and Chebyshev nodes as interpolation points: degree 7 (left) and 14 (right).

4 Finite element basis functions

So far: basis functions have been *global*: $\varphi_i(x) \neq 0$ for most $x \in \Omega$.

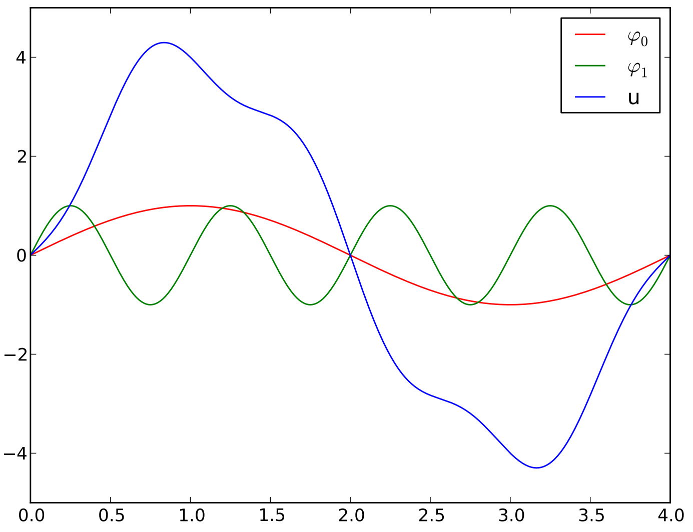


Figure 11: Approximation based on sine basis functions.

In the finite element method, basis functions are *piecewise polynomials* with *local support* ($\varphi_i(x) \neq 0$ for x in a small subdomain of Ω), typically hat-like functions. This makes $u = \sum_j c_j\varphi_j$ a polynomial over (small) subdomains.

4.1 Elements and nodes

Split Ω into non-overlapping subdomains called *elements*:

$$\Omega = \Omega^{(0)} \cup \cdots \cup \Omega^{(n_e)}\,. \tag{36}$$

On each element, introduce points called *nodes*. Below: nodes at the end point of elements.

- $\varphi_i = 1$ at node i and 0 at all other nodes
- φ_i is a Lagrange polynomial on each element
- For nodes at the boundary between two elements, φ_i is made up of a Lagrange polynomial from each element

Important property: c_i is the value of u at node i, x_i:

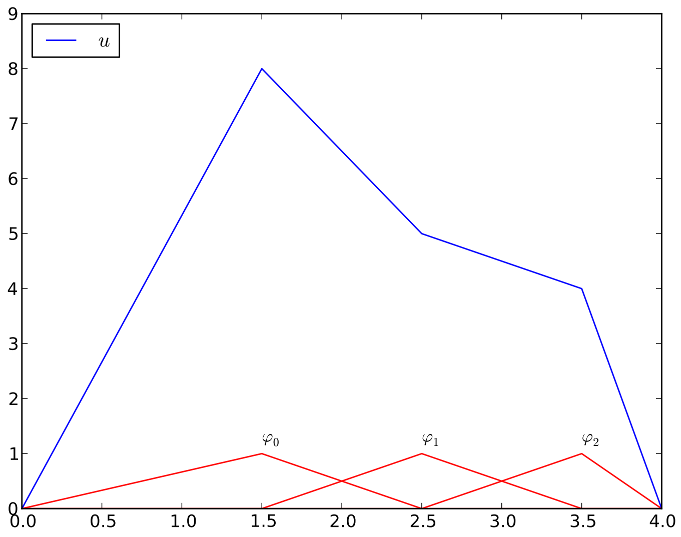


Figure 12: Approximation based on local piecewise linear (hat) functions.

$$u(x_i) = \sum_{j\in\mathcal{I}} c_j\varphi_j(x_i) = c_i\varphi_i(x_i) = c_i\,. \tag{37}$$

$\varphi_i(x)$ is mostly zero throughout the domain:

- $\varphi_i(x) \neq 0$ only on those elements that contain global node i,
- $\varphi_i(x)\varphi_j(x) \neq 0$ if and only if i and j are global node numbers in the same element.

Since $A_{i,j}$ is the integral of $\varphi_i\varphi_j$ it means that *most of the elements in the coefficient matrix will be zero* (important for implementation!).

4.2 Example on quadratic φ_i

Introduce `nodes` and `elements` lists:

```
nodes = [0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0]
elements = [[0, 1, 2], [2, 3, 4], [4, 5, 6], [6, 7, 8]]
```

1. The polynomial that is 1 at local node 1 ($x = 0.375$, global node 3) makes up the basis function $\varphi_3(x)$ over this element, with $\varphi_3(x) = 0$ outside the element.

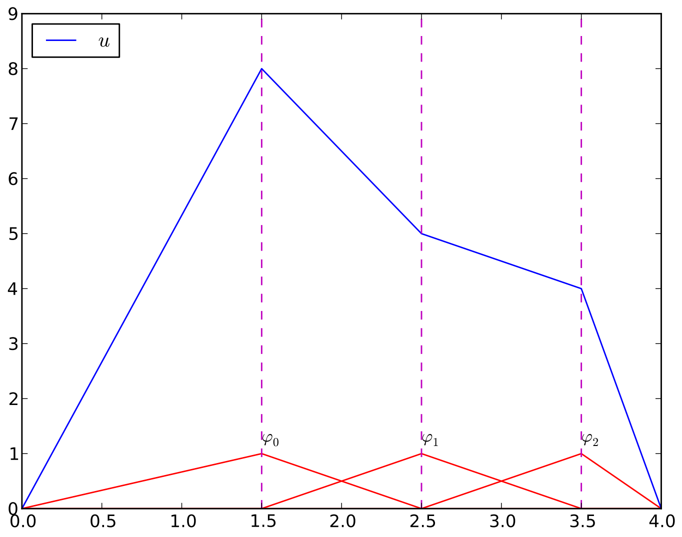


Figure 13: Vertical dashed lines mark element boundaries and nodes.

2. The Lagrange polynomial that is 1 at local node 0 is the "right part" of the global basis function $\varphi_2(x)$. The "left part" of $\varphi_2(x)$ consists of a Lagrange polynomial associated with local node 2 in the neighboring element $\Omega^{(0)} = [0, 0.25]$.

3. Finally, the polynomial that is 1 at local node 2 (global node 4) is the "left part" of the global basis function $\varphi_4(x)$. The "right part" comes from the Lagrange polynomial that is 1 at local node 0 in the neighboring element $\Omega^{(2)} = [0.5, 0.75]$.

4.3 Example on linear φ_i

$$\varphi_i(x) = \begin{cases} 0, & x < x_{i-1}, \\ (x - x_{i-1})/h, & x_{i-1} \leq x < x_i, \\ 1 - (x - x_i)/h, & x_i \leq x < x_{i+1}, \\ 0, & x \geq x_{i+1} \end{cases} \tag{38}$$

4.4 Example on cubic φ_i

4.5 Terminology

- P1 element: piecewise linear φ_i (piecewise linear u)
- P2 element: piecewise quadratic φ_i (piecewise quadratic u)
- P3 element: piecewise cubic φ_i (piecewise cubic u)

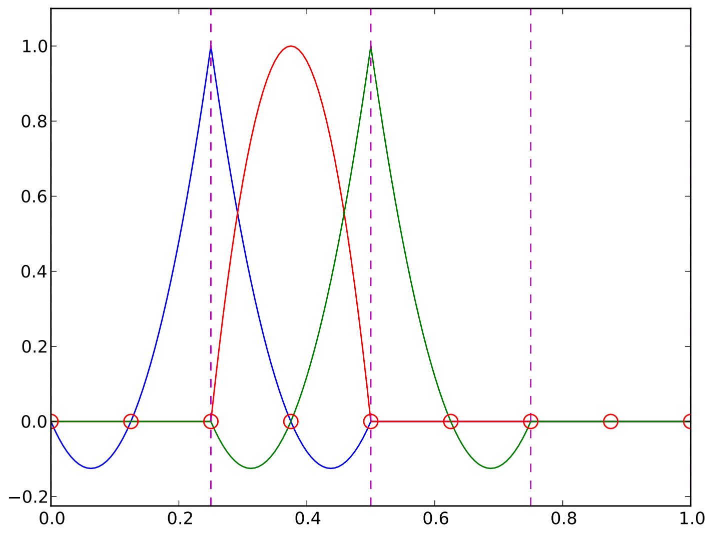


Figure 14: Illustration of the piecewise quadratic basis functions associated with nodes in element 1.

- Pd element: piecewise polynom of degree d

4.6 Back to approximating u: calculating the linear system

Assume uniform element length and P1 elements:

$$\Omega^{(i)} = [x_i, x_{i+1}] = [ih, (i+1)h], \quad i = 0, \ldots, N-1$$

$$A_{i,i-1} = \int_\Omega \varphi_i \varphi_{i-1} dx = \int_{x_{i-1}}^{x_i} \left(1 - \frac{x - x_{i-1}}{h}\right) \frac{x - x_i}{h} dx = \frac{h}{6}\,.$$

Can show that $A_{i,i+1} = h/6$, $A_{i,i} = 2h/3$, but modifications at the boundary of Ω: $A_{0,0} = h/3$ and $A_{N,N} = h/3$.

$$b_i = \int_{x_{i-1}}^{x_i} \frac{x - x_{i-1}}{h} f(x) dx + \int_{x_i}^{x_{i+1}} \left(1 - \frac{x - x_i}{h}\right) f(x) dx\,. \tag{39}$$

With two equal-sized elements in $\Omega = [0,1]$ and $f(x) = x(1-x)$:

$$A = \frac{h}{6}\begin{pmatrix} 2 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 2 \end{pmatrix}, \quad b = \frac{h^2}{12}\begin{pmatrix} 2 - 3h \\ 12 - 14h \\ 10 - 17h \end{pmatrix}.$$

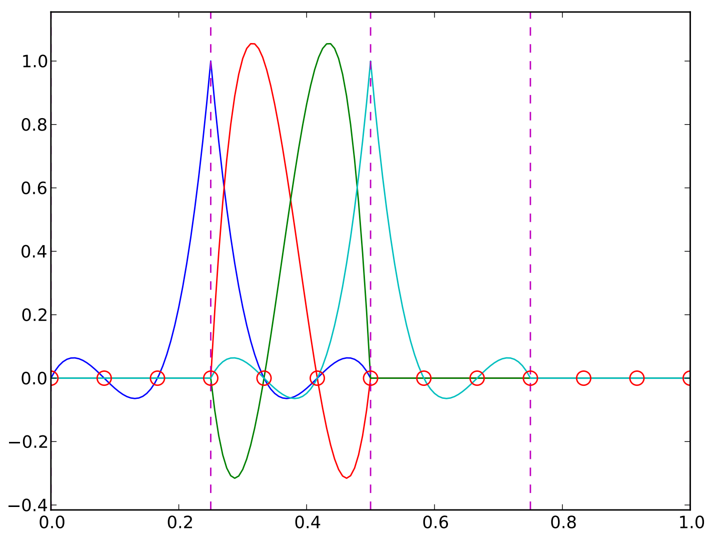


Figure 15: Illustration of the piecewise cubic basis functions associated with nodes in element 1.

$$c_0 = \frac{h^2}{6}, \quad c_1 = h - \frac{5}{6}h^2, \quad c_2 = 2h - \frac{23}{6}h^2 \,.$$

$$u(x) = c_0\varphi_0(x) + c_1\varphi_1(x) + c_2\varphi_2(x)$$

4.7 Assembly of elementwise computations

Split the integral over Ω into elementwise integrals: each element:

$$A_{i,j} = \int_\Omega \varphi_i\varphi_j dx = \sum_e A_{i,j}^{(e)}, \quad A_{i,j}^{(e)} = \int_{\Omega^{(e)}} \varphi_i\varphi_j dx \,. \tag{40}$$

Important:

- $A_{i,j}^{(e)} \neq 0$ if and only if i and j are nodes in element e (otherwise no overlap between the basis functions)
- all the nonzero elements in $A_{i,j}^{(e)}$ are collected in an *element matrix*

$$\tilde{A}^{(e)} = \{\tilde{A}_{r,s}^{(e)}\}, \quad r,s \in \mathcal{I}_d = \{0,\ldots,d\},$$

$$\tilde{A}_{r,s}^{(e)} = \int_{\Omega^{(e)}} \varphi_{q(e,r)}\varphi_{q(e,s)} dx, \quad r,s \in \mathcal{I}_d \,.$$

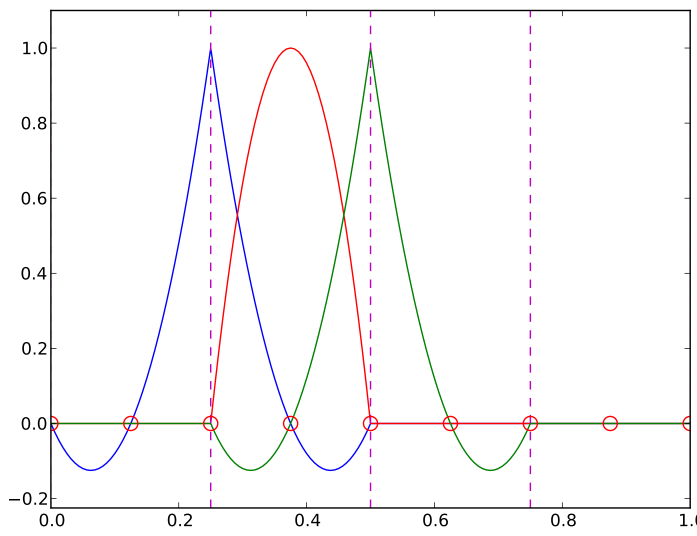


Figure 16: Finite element mesh: nodes are circles and vertical lines denote element boundaries (piecewise quadratic basis functions).

- r, s run over *local node numbers* within an element, while i, j run over *global node numbers*.
- $i = q(e, r)$: mapping of local node number r in element e to the global node number i. Math equivalent to `i=elements[e][r]`.
- Add contribution from an element into the global coefficient matrix (*assembly*):

$$A_{q(e,r),q(e,s)} := A_{q(e,r),q(e,s)} + \tilde{A}^{(e)}_{r,s}, \quad r, s \in \mathcal{I}_d\,. \tag{41}$$

Can also compute the right-hand side of the linear system from elementwise contributions:

$$b_i = \int_\Omega \varphi_i\varphi_j dx = \sum_e b_i^{(e)}, \quad b_i^{(e)} = \int_{\Omega^{(e)}} f(x)\varphi_i(x) dx\,. \tag{42}$$

Important:

- $b_i^{(e)} \neq 0$ if and only if global node i is a node in element e (otherwise $\varphi_i = 0$)
- The $d+1$ nonzero $b_i^{(e)}$ can be collected in an *element vector*

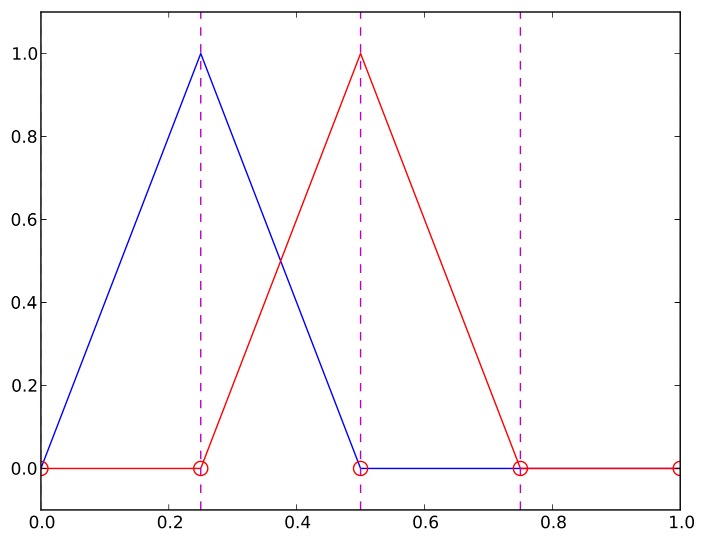


Figure 17: Illustration of the piecewise linear basis functions associated with nodes in element 1.

$$\tilde{b}_r^{(e)} = \{\tilde{b}_r^{(e)}\}, \quad r \in \mathcal{I}_d\,.$$

Assembly:

$$b_{q(e,r)} := b_{q(e,r)} + \tilde{b}_r^{(e)}, \quad r, s \in \mathcal{I}_d\,. \tag{43}$$

4.8 Mapping to a reference element

Instead of computing

$$\tilde{A}_{r,s}^{(e)} = \int_{\Omega^{(e)}} \varphi_{q(e,r)}(x)\varphi_{q(e,s)}(x)dx$$

over some element $\Omega^{(e)} = [x_L, x_R]$, we now map $[x_L, x_R]$ to a standardized reference element domain $[-1, 1]$ with local coordinate X.

Affine mapping:

$$x = \frac{1}{2}(x_L + x_R) + \frac{1}{2}(x_R - x_L)X\,. \tag{44}$$

or rewritten as

$$x = x_m + \frac{1}{2}hX, \qquad x_m = (x_L + x_R)/2 \tag{45}$$

Integrating on the reference element is a matter of just changing the integration variable from x to X. Let

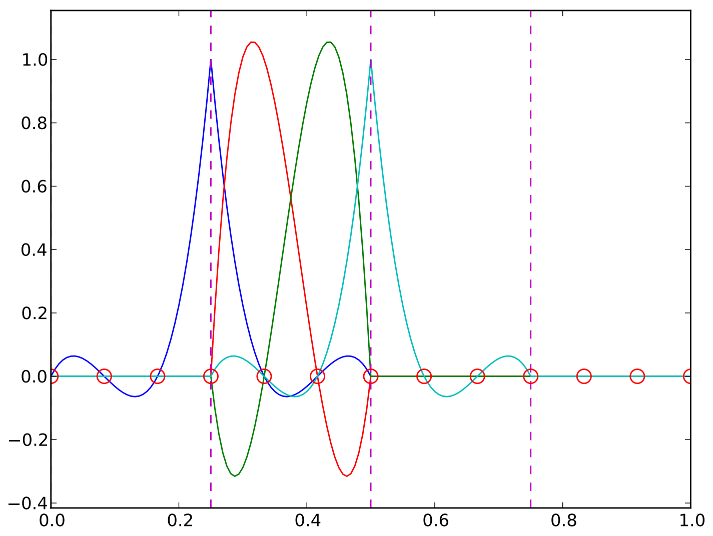


Figure 18: Illustration of the piecewise cubic basis functions associated with nodes in element 1.

$$\tilde{\varphi}_r(X) = \varphi_{q(e,r)}(x(X)) \tag{46}$$

be the basis function associated with local node number r in the reference element. The integral transformation reads

$$\tilde{A}^{(e)}_{r,s} = \int_{\Omega^{(e)}} \varphi_{q(e,r)}(x)\varphi_{q(e,s)}(x)dx = \int_{-1}^{1} \tilde{\varphi}_r(X)\tilde{\varphi}_s(X)\frac{dx}{dX}dX\,. \tag{47}$$

Introduce the notation $\det J = dx/dX = h/2$, because in 2D and 3D we get $\det J$ instead of dx/dX.

$$\tilde{A}^{(e)}_{r,s} = \int_{-1}^{1} \tilde{\varphi}_r(X)\tilde{\varphi}_s(X)\det J\, dX\,. \tag{48}$$

$$\tilde{b}^{(e)}_{r} = \int_{\Omega^{(e)}} f(x)\varphi_{q(e,r)}(x)dx = \int_{-1}^{1} f(x(X))\tilde{\varphi}_r(X)\det J\, dX\,. \tag{49}$$

Advantages:

- Always the same domain for integration: $[-1,1]$
- We only need formulas for $\tilde{\varphi}_r(X)$ on the reference elements (no need for piecewise polynomial definition)

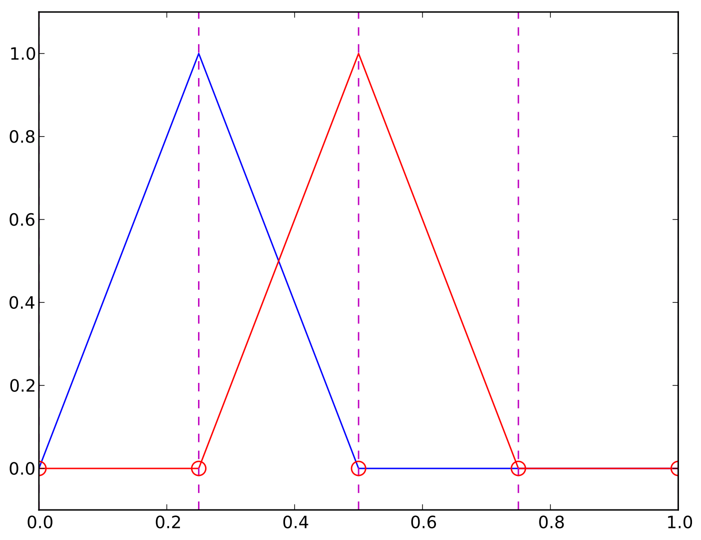


Figure 19: Piecewise linear basis functions φ_1 and φ_2.

P1 elements:

$$\tilde{\varphi}_0(X) = \frac{1}{2}(1 - X) \tag{50}$$

$$\tilde{\varphi}_1(X) = \frac{1}{2}(1 + X) \tag{51}$$

P2 elements:

$$\tilde{\varphi}_0(X) = \frac{1}{2}(X - 1)X \tag{52}$$

$$\tilde{\varphi}_1(X) = 1 - X^2 \tag{53}$$

$$\tilde{\varphi}_2(X) = \frac{1}{2}(X + 1)X \tag{54}$$

4.9 Integration over a reference element

P1 elements and $f(x) = x(1 - x)$.

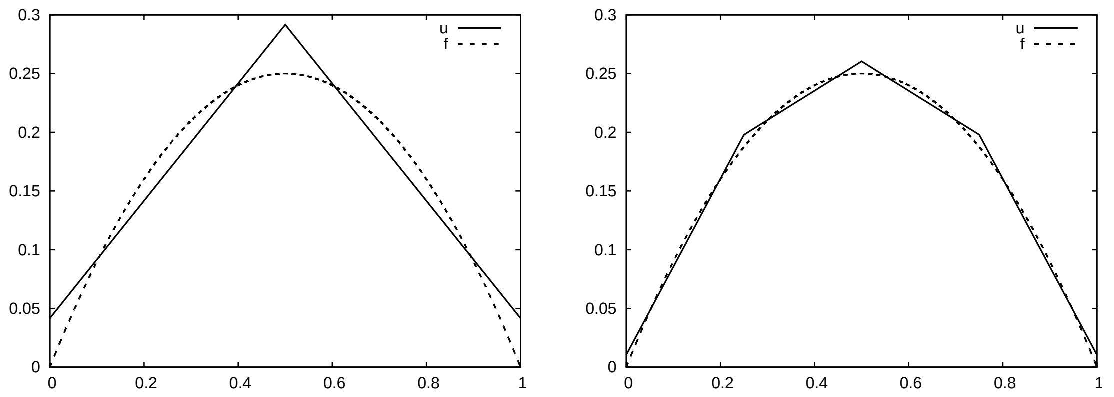


Figure 20: Least squares approximation using 2 (left) and 4 (right) P1 elements.

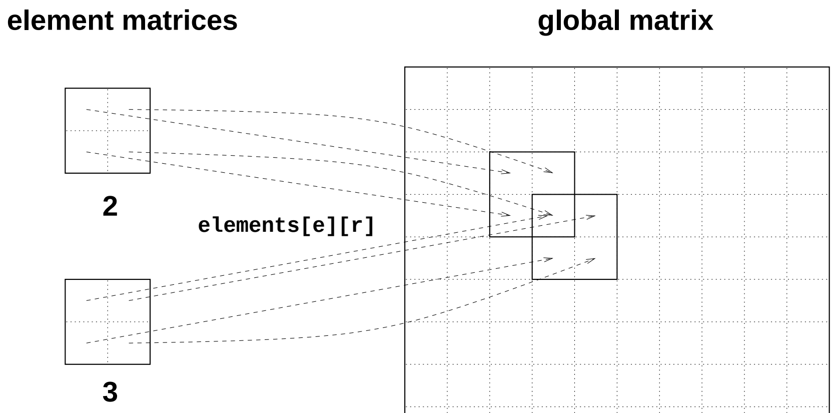


Figure 21: Illustration of matrix assembly.

$$\tilde{A}^{(e)}_{0,0} = \int_{-1}^{1} \tilde{\varphi}_0(X)\tilde{\varphi}_0(X)\frac{h}{2}dX$$

$$= \int_{-1}^{1} \frac{1}{2}(1-X)\frac{1}{2}(1-X)\frac{h}{2}dX = \frac{h}{8}\int_{-1}^{1}(1-X)^2 dX = \frac{h}{3}, \qquad (55)$$

$$\tilde{A}^{(e)}_{1,0} = \int_{-1}^{1} \tilde{\varphi}_1(X)\tilde{\varphi}_0(X)\frac{h}{2}dX$$

$$= \int_{-1}^{1} \frac{1}{2}(1+X)\frac{1}{2}(1-X)\frac{h}{2}dX = \frac{h}{8}\int_{-1}^{1}(1-X^2) dX = \frac{h}{6}, \qquad (56)$$

$$\tilde{A}^{(e)}_{0,1} = \tilde{A}^{(e)}_{1,0}, \qquad (57)$$

$$\tilde{A}^{(e)}_{1,1} = \int_{-1}^{1} \tilde{\varphi}_1(X)\tilde{\varphi}_1(X)\frac{h}{2}dX$$

$$= \int_{-1}^{1} \frac{1}{2}(1+X)\frac{1}{2}(1+X)\frac{h}{2}dX = \frac{h}{8}\int_{-1}^{1}(1+X)^2 dX = \frac{h}{3}. \qquad (58)$$

$$
\begin{aligned}
\tilde{b}_0^{(e)} &= \int_{-1}^{1} f(x(X))\tilde{\varphi}_0(X)\frac{h}{2}dX \\
&= \int_{-1}^{1} (x_m + \frac{1}{2}hX)(1-(x_m+\frac{1}{2}hX))\frac{1}{2}(1-X)\frac{h}{2}dX \\
&= -\frac{1}{24}h^3 + \frac{1}{6}h^2x_m - \frac{1}{12}h^2 - \frac{1}{2}hx_m^2 + \frac{1}{2}hx_m \qquad (59)\\
\tilde{b}_1^{(e)} &= \int_{-1}^{1} f(x(X))\tilde{\varphi}_0(X)\frac{h}{2}dX \\
&= \int_{-1}^{1} (x_m + \frac{1}{2}hX)(1-(x_m+\frac{1}{2}hX))\frac{1}{2}(1+X)\frac{h}{2}dX \\
&= -\frac{1}{24}h^3 - \frac{1}{6}h^2x_m + \frac{1}{12}h^2 - \frac{1}{2}hx_m^2 + \frac{1}{2}hx_m\,. \qquad (60)
\end{aligned}
$$

x_m: element midpoint.

Tedious calculations! Let's use symbolic software:

```
>>> import sympy as sm
>>> x, x_m, h, X = sm.symbols('x x_m h X')
>>> sm.integrate(h/8*(1-X)**2, (X, -1, 1))
h/3
>>> sm.integrate(h/8*(1+X)*(1-X), (X, -1, 1))
h/6
>>> x = x_m + h/2*X
>>> b_0 = sm.integrate(h/4*x*(1-x)*(1-X), (X, -1, 1))
>>> print b_0
-h**3/24 + h**2*x_m/6 - h**2/12 - h*x_m**2/2 + h*x_m/2
```

Can printe out in LaTeX too (convenient for copying into reports):

```
>>> print sm.latex(b_0, mode='plain')
- \frac{1}{24} h^{3} + \frac{1}{6} h^{2} x_{m}
- \frac{1}{12} h^{2} - \frac{1}{2} h x_{m}^{2}
+ \frac{1}{2} h x_{m}
```

5 Implementation

- Coming functions appear in `fe_approx1D.py`[2]
- Functions can operate in symbolic or numeric mode
- The code documents all steps in finite element calculations

5.1 Integration

Compute $\tilde{\varphi}_r(X)$ as a Lagrange polynomial of degree `d`:

[2]http://tinyurl.com/jvzzcfn/fem/fe_approx1D.py

```
import sympy as sm
import numpy as np

def phi_r(r, X, d):
    if isinstance(X, sm.Symbol):
        h = sm.Rational(1, d)  # node spacing
        nodes = [2*i*h - 1 for i in range(d+1)]
    else:
        # assume X is numeric: use floats for nodes
        nodes = np.linspace(-1, 1, d+1)
    return Lagrange_polynomial(X, r, nodes)

def Lagrange_polynomial(x, i, points):
    p = 1
    for k in range(len(points)):
        if k != i:
            p *= (x - points[k])/(points[i] - points[k])
    return p
```

The complete basis:

```
def basis(d=1):
    X = sm.Symbol('X')
    phi = [phi_r(r, X, d) for r in range(d+1)]
    return phi
```

Element matrix:

```
def element_matrix(phi, Omega_e, symbolic=True):
    n = len(phi)
    A_e = sm.zeros((n, n))
    X = sm.Symbol('X')
    if symbolic:
        h = sm.Symbol('h')
    else:
        h = Omega_e[1] - Omega_e[0]
    detJ = h/2  # dx/dX
    for r in range(n):
        for s in range(r, n):
            A_e[r,s] = sm.integrate(phi[r]*phi[s]*detJ, (X, -1, 1))
            A_e[s,r] = A_e[r,s]
    return A_e
```

```
>>> from fe_approx1D import *
>>> phi = basis(d=1)
>>> phi
[1/2 - X/2, 1/2 + X/2]
>>> element_matrix(phi, Omega_e=[0.1, 0.2], symbolic=True)
[h/3, h/6]
[h/6, h/3]
>>> element_matrix(phi, Omega_e=[0.1, 0.2], symbolic=False)
[0.0333333333333333, 0.0166666666666667]
[0.0166666666666667, 0.0333333333333333]
```

Element vector:

```
def element_vector(f, phi, Omega_e, symbolic=True):
    n = len(phi)
    b_e = sm.zeros((n, 1))
    # Make f a function of X
    X = sm.Symbol('X')
    if symbolic:
        h = sm.Symbol('h')
    else:
        h = Omega_e[1] - Omega_e[0]
    x = (Omega_e[0] + Omega_e[1])/2 + h/2*X  # mapping
    f = f.subs('x', x)  # substitute mapping formula for x
    detJ = h/2  # dx/dX
    for r in range(n):
        b_e[r] = sm.integrate(f*phi[r]*detJ, (X, -1, 1))
    return b_e
```

Note: need to replace the symbol `x` in the expression for `f` by the mapping formula such that `f` contains the variable `X`.

Not all $f(x)$ can be integrated by `sympy` so let us fall back on numerical integration:

```
def element_vector(f, phi, Omega_e, symbolic=True):
        ...
        I = sm.integrate(f*phi[r]*detJ, (X, -1, 1))
        if isinstance(I, sm.Integral):
            h = Omega_e[1] - Omega_e[0]  # Ensure h is numerical
            detJ = h/2
            integrand = sm.lambdify([X], f*phi[r]*detJ)
            I = sm.mpmath.quad(integrand, [-1, 1])
        b_e[r] = I
        ...
```

5.2 Linear system assembly and solution

```
def assemble(nodes, elements, phi, f, symbolic=True):
    n_n, n_e = len(nodes), len(elements)
    zeros = sm.zeros if symbolic else np.zeros
    A = zeros((n_n, n_n))
    b = zeros((n_n, 1))
    for e in range(n_e):
        Omega_e = [nodes[elements[e][0]], nodes[elements[e][-1]]]

        A_e = element_matrix(phi, Omega_e, symbolic)
        b_e = element_vector(f, phi, Omega_e, symbolic)

        for r in range(len(elements[e])):
            for s in range(len(elements[e])):
                A[elements[e][r],elements[e][s]] += A_e[r,s]
            b[elements[e][r]] += b_e[r]
    return A, b
```

Linear system solution:

```
if symbolic:
    c = A.LUsolve(b)           # sympy arrays, symbolic Gaussian elim.
else:
    c = np.linalg.solve(A, b)  # numpy arrays, numerical solve
```

5.3 Example on computing approximations

```
>>> h, x = sm.symbols('h x')
>>> nodes = [0, h, 2*h]
>>> elements = [[0, 1], [1, 2]]
>>> phi = basis(d=1)
>>> f = x*(1-x)
>>> A, b = assemble(nodes, elements, phi, f, symbolic=True)
>>> A
[h/3,   h/6,   0]
[h/6, 2*h/3, h/6]
[  0,   h/6, h/3]
>>> b
[     h**2/6 - h**3/12]
[      h**2 - 7*h**3/6]
[5*h**2/6 - 17*h**3/12]
>>> c = A.LUsolve(b)
>>> c
[                           h**2/6]
[12*(7*h**2/12 - 35*h**3/72)/(7*h)]
[  7*(4*h**2/7 - 23*h**3/21)/(2*h)]
```

Numerical computations:

```
>>> nodes = [0, 0.5, 1]
>>> elements = [[0, 1], [1, 2]]
>>> phi = basis(d=1)
>>> x = sm.Symbol('x')
>>> f = x*(1-x)
>>> A, b = assemble(nodes, elements, phi, f, symbolic=False)
>>> A
[ 0.166666666666667, 0.0833333333333333,                  0]
[0.0833333333333333,  0.333333333333333, 0.0833333333333333]
[                 0, 0.0833333333333333,  0.166666666666667]
>>> b
[          0.03125]
[0.104166666666667]
[          0.03125]
>>> c = A.LUsolve(b)
>>> c
[0.0416666666666666]
[ 0.291666666666667]
[0.0416666666666666]
```

5.4 The structure of the coefficient matrix

```
>>> d=1; n_e=8; Omega=[0,1]  # 8 linear elements on [0,1]
>>> phi = basis(d)
>>> f = x*(1-x)
>>> nodes, elements = mesh_symbolic(n_e, d, Omega)
>>> A, b = assemble(nodes, elements, phi, f, symbolic=True)
>>> A
[h/3,   h/6,     0,     0,     0,     0,     0,     0,   0]
[h/6, 2*h/3,   h/6,     0,     0,     0,     0,     0,   0]
[  0,   h/6, 2*h/3,   h/6,     0,     0,     0,     0,   0]
[  0,     0,   h/6, 2*h/3,   h/6,     0,     0,     0,   0]
[  0,     0,     0,   h/6, 2*h/3,   h/6,     0,     0,   0]
[  0,     0,     0,     0,   h/6, 2*h/3,   h/6,     0,   0]
[  0,     0,     0,     0,     0,   h/6, 2*h/3,   h/6,   0]
[  0,     0,     0,     0,     0,     0,   h/6, 2*h/3, h/6]
[  0,     0,     0,     0,     0,     0,     0,   h/6, h/3]
```

Note: do this by hand to understand what is going on!
The coefficient matrix is sparse (means mostly zeros):

$$A = \frac{h}{6}\left(\begin{array}{cccccccccc}
2 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
1 & 4 & 1 & \ddots & & & & & \vdots \\
0 & 1 & 4 & 1 & \ddots & & & & \vdots \\
\vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\
\vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\
\vdots & & & 0 & 1 & 4 & 1 & \ddots & \vdots \\
\vdots & & & & \ddots & \ddots & \ddots & \ddots & 0 \\
\vdots & & & & & \ddots & 1 & 4 & 1 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 & 2
\end{array}\right) \tag{61}$$

For P2 elements:

$$A = \frac{h}{30}\left(\begin{array}{ccccccccc}
4 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 16 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 2 & 8 & 2 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & 16 & 2 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 2 & 8 & 2 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 2 & 16 & 2 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 2 & 8 & 2 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 2 & 16 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & 4
\end{array}\right) \tag{62}$$

Exploiting the sparse structure is important for efficient computations.

5.5 Sparse matrix storage and solution

We have observed that $\varphi_i\varphi_j \neq 0$ only when i and j are nodes in the same element. This means that $A_{i,j} = 0$ for most i and j, and the coefficient matrix is then *sparse*.

- P1 elements: only 3 nonzero entires per row
- P2 elements: only 5 nonzero entires per row

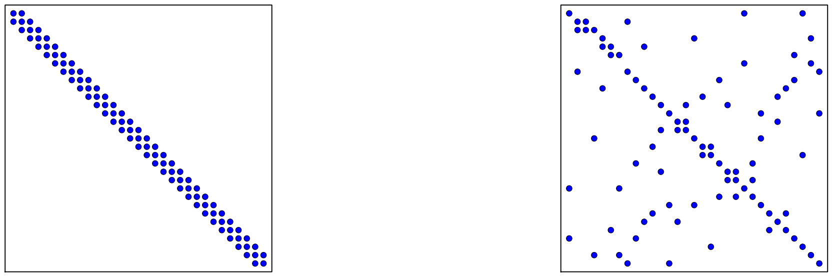


Figure 22: Matrix sparsity pattern for left-to-right numbering (left) and random numbering (right) of nodes in P1 elements.

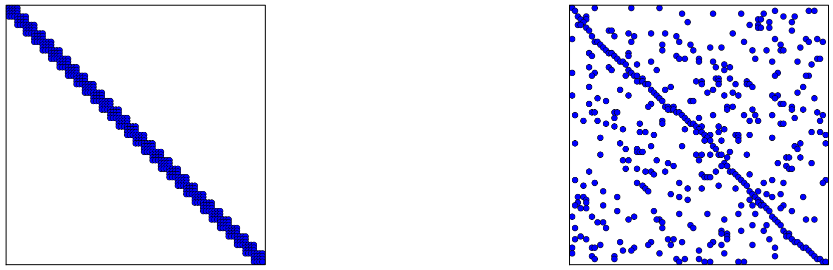


Figure 23: Matrix sparsity pattern for left-to-right numbering (left) and random numbering (right) of nodes in P3 elements.

- P2 elements: only 7 nonzero entires per row
- It is important to utilize sparse storage and sparse solvers
- In Python: `scipy.sparse` package

5.6 Applications

Compute a mesh with `n_e` elements, basis functions of degree `d`, and approximate a given symbolic expression `f` by a finite element expansion $u(x) = \sum_j c_j \varphi_j(x)$:

```
def approximate(f, symbolic=False, d=1, n_e=4, filename='tmp.pdf'):
```

Tests:

```
import sympy as sm
from fe_approx1D import approximate
x = sm.Symbol('x')

approximate(f=x*(1-x)**8, symbolic=False, d=1, n_e=4)
approximate(f=x*(1-x)**8, symbolic=False, d=2, n_e=2)
approximate(f=x*(1-x)**8, symbolic=False, d=1, n_e=8)
approximate(f=x*(1-x)**8, symbolic=False, d=2, n_e=4)
```

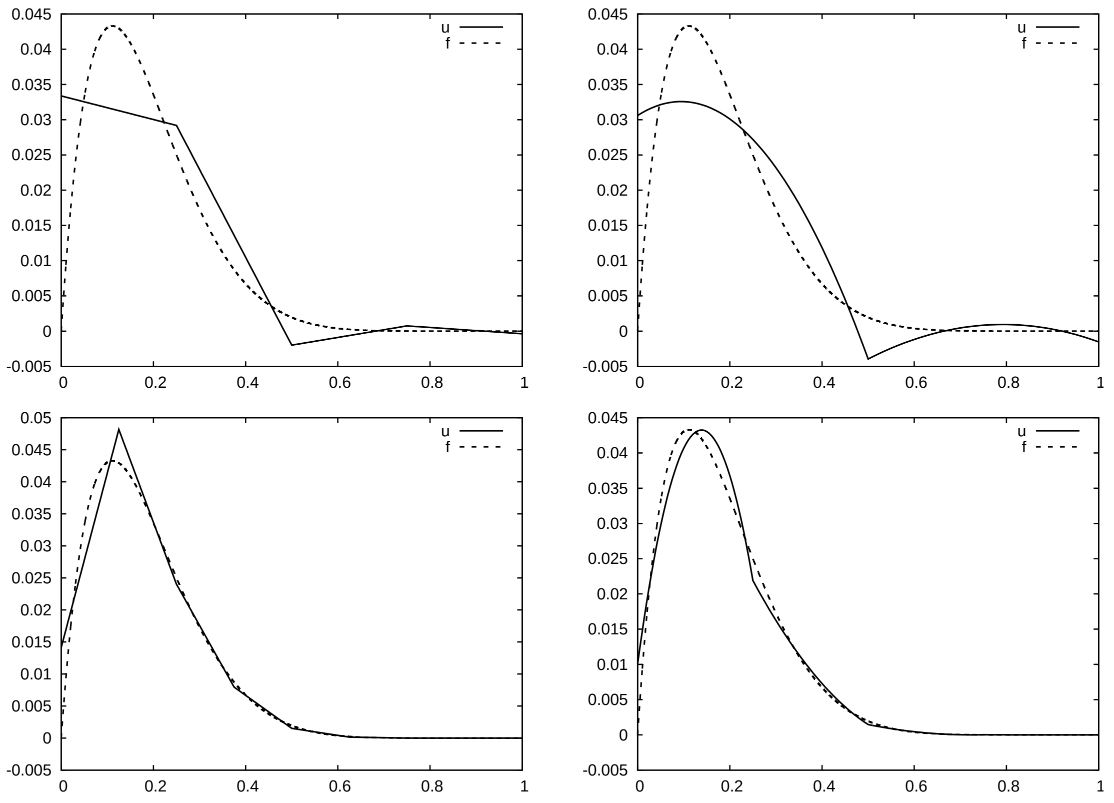


Figure 24: Comparison of the finite element approximations: 4 P1 elements with 5 nodes (upper left), 2 P2 elements with 5 nodes (upper right), 8 P1 elements with 9 nodes (lower left), and 4 P2 elements with 9 nodes (lower right).

6 Comparison of finite element and finite difference approximation

- Finite difference approximation of a function $f(x)$: simply choose $u_i = f(x_i)$ (interpolation).
- Galerkin/projection and least squares method: must derive and solve a linear system.
- What is really the difference?

6.1 Collocation (interpolation)

Let x_i, $i \in \mathcal{I}$, be the nodes in the mesh. Collocation means

$$u(x_i) = f(x_i), \quad i \in \mathcal{I}, \tag{63}$$

which translates to

$$\sum_{j\in\mathcal{I}} c_j\varphi_j(x_i) = f(x_i),$$

but $\varphi_j(x_i) = 0$ if $i \neq j$ so the sum collapses to one term $c_i\varphi_i(x_i) = c_i$, and we have the result

$$c_i = f(x_i)\,. \tag{64}$$

- This yields the same result as the standard finite difference approach
- u *interpolates* f at the node points
- u has a variation between the node points dictated by the φ_i functions
- Collocation (interpolation) is not much used when solving differential equation, except for approximating initial conditions (like here)

6.2 Finite difference interpretation of a finite element approximation

- Scope: work with P1 elements (most similar to finite differences)
- Use Galerkin/project or least squares (equivalent)
- Interpret the resulting linear system as finite difference equations

General formula for computing the linear system:

$$\sum_{j\in\mathcal{I}} c_j(\varphi_i,\varphi_j) = (f,\varphi_i),\quad i\in\mathcal{I}\,.$$

The P1 finite element machinery results in a linear system where equation no i is

$$\frac{h}{6}(u_{i-1} + 4u_i + u_{i+1}) = (f,\varphi_i)\,. \tag{65}$$

Note:

- We have used u_i for c_i to simplify notation with finite differences.
- The finite difference counterpart is just $u_i = f_i$.

Rewrite:

$$h(u_i - \frac{1}{6}(-u_{i-1} + 2u_i - u_{i+1}))\,. \tag{66}$$

This looks like a finite difference approximation of

$$h(u - \frac{h^2}{6}u''),$$

That is, the matrix arises from

$$[h(u - \frac{h^2}{6}D_x D_x u)]_i$$

The right-hand side is more complicated:

$$(f, \varphi_i) = \int_{x_{i-1}}^{x_i} f(x)\frac{1}{h}(x - x_{i-1})dx + \int_{x_i}^{x_{i+1}} f(x)\frac{1}{h}(1 - (x - x_i))dx\,.$$

Can't to much unless we specialize f or use *numerical integration.*

Apply the Trapezoidal rule using all the nodes:

$$(f, \varphi_i) = \int_\Omega f\varphi_i dx \approx h\frac{1}{2}(f(x_0)\varphi_i(x_0) + f(x_N)\varphi_i(x_N)) + h\sum_{j=1}^{N-1} f(x_j)\varphi_i(x_j)\,.$$

Since φ_i is zero at all these points, except at x_i, the Trapezoidal rule collapses to one term:

$$(f, \varphi_i) \approx hf(x_i), \quad i = 1, \ldots, N-1\,. \tag{67}$$

This is the same result as in collocation (interpolation) and the finite difference method!

Simpson's rule:

$$\int_\Omega f(x)dx \approx \frac{\tilde{h}}{3}\left(f(x_0) + 2\sum_{j=2,4,6,\ldots} f(x_j) + 4\sum_{j=1,3,5,\ldots} f(x_j) + f(x_{2N})\right),$$

Here f is sampled at midpoints and endpoints of the elements. The sums collapse because $\varphi_i = 0$ at most of these points.

Result:

$$(f, \varphi_i) \approx \frac{h}{3}(f(x_i - \frac{1}{2}h) + f(x_i) + f(x_i + \frac{1}{2}h)\,. \tag{68}$$

In a finite difference context we would typically express this formula as

$$\frac{h}{3}(f_{i-\frac{1}{2}} + f_i + f_{i+\frac{1}{2}})\,.$$

Conclusions:

- While the finite difference method just samples f at x_i, the finite element method applies an average of f around x_i.

- On the left-hand side we have a term $\sim hu''$, and u'' also contribute to smoothing.
- There are some inherent smoothing elements in the finite element method.

With Trapezoidal integration of (f, φ_i) we essentially solve

$$u + \frac{h^2}{6}u'' = f, \quad u'(0) = u'(L) = 0, \tag{69}$$

expressed with operator notation as

$$[u + \frac{h^2}{6}D_xD_xu = f]_i\,. \tag{70}$$

With Simpson integration of (f, φ_i) we essentially solve

$$[u + \frac{h^2}{6}D_xD_xu = \bar{f}]_i, \tag{71}$$

where

$$\bar{f}_i = \frac{1}{3}(f_{i-1/2} + f_i + f_{i+1/2})$$

Note:

- As $h \rightarrow 0$, $hu'' \rightarrow 0$ and $\bar{f}_i \rightarrow f_i$, and all approaches yield the same result.

6.3 Making finite elements behave as finite differences

- Can we adjust the finite element method so that we do not get the extra hu'' smoothing term and averaging of f?
- This is important in time-dependent problems to incorporate good properties of finite differences into finite elements.

Result:

- By computing all integrals by the Trapezoidal method, P1 elements recovers the same formulas as in the finite difference method ($u_i = f_i$).
- Specifically: the coefficient matrix becomes diagonal ("lumped")
- Loss of accuracy? The Trapezoidal rule has error $\mathcal{O}(h^2)$, the same as the approximation error in P1 elements (integrated exactly).

Reason:

- Integration rules sample the integrand in nodes will sample φ_i at points where it is 0. A lot of terms vanish.

7 A generalized element concept

So far,

- *Nodes*: points for defining φ_i and compute u values
- *Elements*: subdomain (containing some nodes)
- This is a common notion of nodes and elements.

An extended and generalized element concept:

- An *element* is the collection of the subdomain (previous "element"), points where we seek function values, basis functions, numberings, mappings, etc.
- We introduce *cell* for the subdomain that we up to now called element.
- A cell has *vertices* (interval end points).
- *Nodes* are, almost as before, points where we want to compute unknown functions.
- *Degrees of freedom* is what the c_j represent (usually function values at nodes).

The concept of a finite element.

- a *reference cell* in a local reference coordinate system;
- a set of *basis functions* $\tilde{\varphi}_i$ defined on the cell;
- a set of *degrees of freedom* that uniquely determine the basis functions such that $\tilde{\varphi}_i = 1$ for degree of freedom number i and $\tilde{\varphi}_i = 0$ for all other degrees of freedom;
- a mapping between local and global degree of freedom numbers;
- a *mapping* of the reference cell onto to cell in the physical domain.

7.1 Implementation

- We replace `nodes` by `vertices`.
- We introduce `cells` such that `cell[e][r]` gives the mapping from local vertex `r` in cell `e` to the global vertex number in `vertices`.
- We replace `elements` by `dof_map` (the contents are the same).

Example: $\Omega = [0, 1]$ is divided into two cells, $\Omega^{(0)} = [0, 0.4]$ and $\Omega^{(1)} = [0.4, 1]$. Define P2 elements in each cell.

```
vertices = [0, 0.4, 1]
cells = [[0, 1], [1, 2]]
dof_map = [[0, 1, 2], [1, 2, 3]]
```

Example: u is piecewise constant in each cell (P0 element). Same `vertices` and `cells`, but

```
dof_map = [[0], [1], [2]]
```

May think of nodes in the middle of each element.

The assembly process must now use the `dof_map` (no `elements` data structure anymore):

```
A[dof_map[e][r], dof_map[e][s]] += A_e[r,s]
b[dof_map[e][r]] += b_e[r]
```

We will hereafter work with `cells`, `vertices`, and `dof_map`.

7.2 Cubic Hermite polynomials

- Can we construct $\varphi_i(x)$ with continuous derivatives?
- Yes!

Consider a reference cell $[-1, 1]$. We introduce two nodes, $X = -1$ and $X = 1$. The degrees of freedom are

- 0: value of function at $X = -1$
- 1: value of first derivative at $X = -1$
- 2: value of function at $X = 1$
- 3: value of first derivative at $X = 1$

Derivatives as unknowns ensure the same $\varphi_i'(x)$ value at nodes!

The 4 degrees of freedom, applied to the 4 $\varphi_i(x)$, give four 4×4 linear systems to determine the 4 cubic $\varphi_i(x)$.

$$\tilde{\varphi}_0(X) = 1 - \frac{3}{4}(X+1)^2 + \frac{1}{4}(X+1)^3 \tag{72}$$

$$\tilde{\varphi}_1(X) = -(X+1)(1 - \frac{1}{2}(X+1))^2 \tag{73}$$

$$\tilde{\varphi}_2(X) = \frac{3}{4}(X+1)^2 - \frac{1}{2}(X+1)^3 \tag{74}$$

$$\tilde{\varphi}_3(X) = -\frac{1}{2}(X+1)(\frac{1}{2}(X+1)^2 - (X+1)) \tag{75}$$

$$\tag{76}$$

8 Numerical integration

$\int_\Omega f\varphi_i dx$ must in general be computed by numerical integration.

Common form:

$$\int_{-1}^{1} g(X)dX \approx \sum_{j=0}^{M} w_j \bar{X}_j, \tag{77}$$

where

- $\bar{X}_j$ are *integration points*
- w_j are *integration weights*
- Different rules correspond to different choices of points and weights

Simplest possibility: the Midpoint rule,

$$\int_{-1}^{1} g(X)dX \approx 2g(0), \quad \bar{X}_0 = 0,\ w_0 = 2, \tag{78}$$

8.1 Newton-Cotes rules

Idea: use a fixed, uniformly distributed set of points. The points usually coincides with nodes (in higher-order elements). Very useful for making $\varphi_i\varphi_j = 0$ and get diagonal (mass) matrices ("lumping").

The Trapezoidal rule:

$$\int_{-1}^{1} g(X)dX \approx g(-1) + g(1), \quad \bar{X}_0 = -1,\ \bar{X}_1 = 1,\ w_0 = w_1 = 1, \tag{79}$$

Simpson's rule:

$$\int_{-1}^{1} g(X)dX \approx \frac{1}{3}\left(g(-1) + 4g(0) + g(1)\right), \tag{80}$$

where

$$\bar{X}_0 = -1,\ \bar{X}_1 = 0,\ \bar{X}_2 = 1,\ w_0 = w_2 = \frac{1}{3},\ w_1 = \frac{4}{3}. \tag{81}$$

8.2 Gauss-Legendre rules with optimized points

- Do not fix points, e.g., uniform distribution
- Optimize the location of points
- Gauss-Legendre rules (quadrature) adjust points and weights to integrate polynomials exactly

$$M = 1: \quad \bar{X}_0 = -\frac{1}{\sqrt{3}},\ \bar{X}_1 = \frac{1}{\sqrt{3}},\ w_0 = w_1 = 1 \tag{82}$$

$$M = 2: \quad \bar{X}_0 = -\sqrt{\frac{3}{5}},\ \bar{X}_0 = 0,\ \bar{X}_2 = \sqrt{\frac{3}{5}},\ w_0 = w_2 = \frac{5}{9},\ w_1 = \frac{8}{9}\,. \tag{83}$$

- $M = 1$: integrates 3rd degree polynomials exactly
- $M = 2$: integrates 5th degree polynomials exactly
- In general, M-point rule integrates a polynomial of degree $2M + 1$ exactly.

See `numint.py`[3] for a large collection of Gauss-Legendre rules.

9 Approximation of functions in 2D

All the concepts and algorithms developed for approximation of 1D functions $f(x)$ can readily be extended to 2D functions $f(x, y)$ and 3D functions $f(x, y, z)$. Key formulas stay the same.

Inner product in 2D:

$$(f, g) = \int_\Omega f(x, y)g(x, y)dxdy \tag{84}$$

Constructing 2D basis functions from 1D functions. Given 1D basis functions

$$\{\hat{\varphi}_0(x), \ldots, \hat{\varphi}_{N_x}(x)\},$$

we can combine these two form 2D basis functions: $\hat{\varphi}_p(x)\hat{\varphi}_q(y)$ (tensor-product definition).

Either double index (p, q),

$$u = \sum_{p=0}^{N_y}\sum_{q=0}^{N_x} c_{p,q}\varphi_{p,q}(x, y), \quad \varphi_{p,q}(x, y) = \hat{\varphi}_p(x)\hat{\varphi}_q(y),$$

or we may transform the double index (p, q) to a single index i, using $i = pN_y + q$ or $i = qN_x + p$.

Simple example:

$$\{1, x\}$$

$$\varphi_{0,0} = 1, \quad \varphi_{1,0} = x, \quad \varphi_{0,1} = y, \quad \varphi_{1,1} = xy,$$

or with a single index:

$$\varphi_0 = 1, \quad \varphi_1 = x, \quad \varphi_2 = y, \quad \varphi_3 = xy\,.$$

See notes for details of a hand-calculation.

[3] `http://tinyurl.com/jvzzcfn/fem/numint.py`

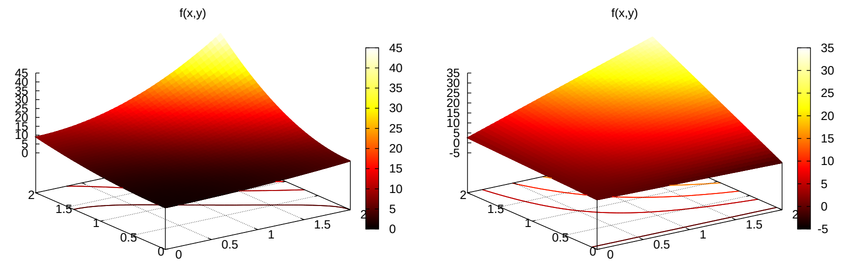


Figure 25: Approximation of a 2D quadratic function (left) by a 2D bilinear function (right) using the Galerkin or least squares method.

10 Finite elements in 2D and 3D

The two great advantages of the finite element method:

- Can handle complex-shaped domains in 2D and 3D
- Can easily provide higher-order polynomials in the approximation

Typical cell types: triangles and quadrilaterals in 2D, tetrahetra and hexahedra in 3D.

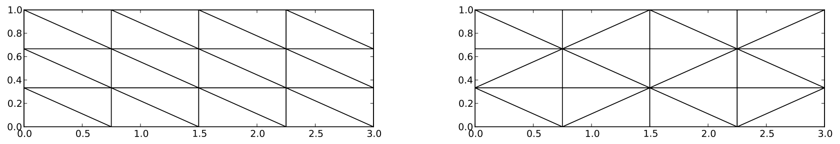


Figure 26: Examples on 2D P1 elements.

10.1 Basis functions over triangles in the physical domain

The P1 triangular 2D element: u is linear $ax + by + c$ over each triangular cell.

- Cells: triangles
- Vertices: corners of the cells
- Nodes = vertices
- Degrees of freedom: function values at the nodes
- Linear mapping of reference element onto general triangular cell
- φ_i: pyramid shape, composed of planes.
- $\varphi_i = 1$ at vertex (node) i, 0 at all other vertices (nodes).

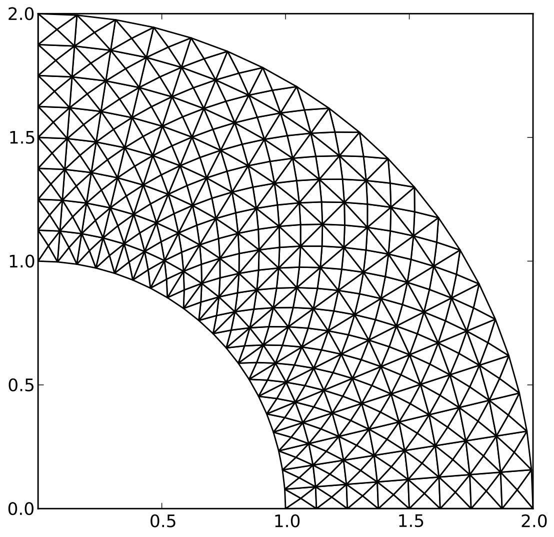


Figure 27: Examples on 2D P1 elements in a deformed geometry.

Element matrices and vectors. $\varphi_i\varphi_j \neq 0$ only if i and j are degrees of freedom (vertices/nodes) in the same element. Element matrix: 3×3.

10.2 Basis functions over triangles in the reference cell

$$\tilde{\varphi}_0(X, Y) = 1 - X - Y, \tag{85}$$
$$\tilde{\varphi}_1(X, Y) = X, \tag{86}$$
$$\tilde{\varphi}_2(X, Y) = Y \tag{87}$$

Higher-order elements introduce more nodes. Degrees of freedom are the function values at the nodes.

Higher-order means higher-degree polynomials.

- Interval, triangle, tetrahedron: *simplex* element
- Plural quick-form: *simplices*
- Side of the cell is called *face*
- Thetrahedron has also *edges*

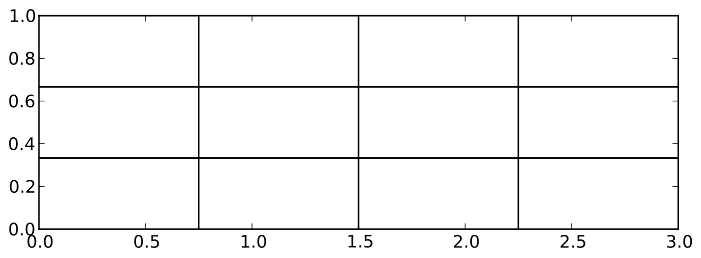


Figure 28: Examples on 2D Q1 elements.

10.3 Affine mapping of the reference cell

Mapping of local (X, Y) coordinates in the reference cell to global, physical (x, y) coordinates:

$$\boldsymbol{x} = \sum_r \tilde{\varphi}_r^{(1)}(\boldsymbol{X})\boldsymbol{x}_{q(e,r)}, \tag{88}$$

where

- r runs over the local vertex numbers in the cell
- $\boldsymbol{x}_i$ are the (x, y) coordinates of vertex i
- $\tilde{\varphi}_r^{(1)}$ are P1 basis functions

This mapping preserves the straight/planar faces and edges.

10.4 Isoparametric mapping of the reference cell

Idea: Use the basis functions of the element to map the element

$$\boldsymbol{x} = \sum_r \tilde{\varphi}_r(\boldsymbol{X})\boldsymbol{x}_{q(e,r)}, \tag{89}$$

Advantage: higher-order polynomial basis functions now maps the reference cell to a *curved* triangle or tetrahedron.

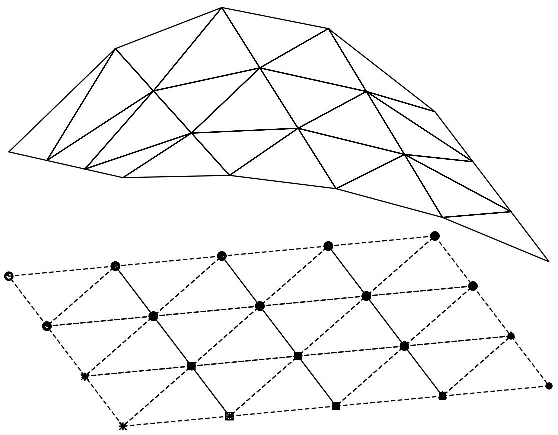


Figure 29: Example on piecewise linear 2D functions defined on triangles.

10.5 Computing integrals

10.6 Differential equation models

Abstract differential equation:

$$\mathcal{L}(u) = 0, \quad x \in \Omega\,. \tag{90}$$

Examples:

$$\mathcal{L}(u) = \frac{d^2u}{dx^2} - f(x), \tag{91}$$

$$\mathcal{L}(u) = \frac{d}{dx}\left(a(x)\frac{du}{dx}\right) + f(x), \tag{92}$$

$$\mathcal{L}(u) = \frac{d}{dx}\left(a(u)\frac{du}{dx}\right) - \alpha u + f(x), \tag{93}$$

$$\mathcal{L}(u) = \frac{d}{dx}\left(a(u)\frac{du}{dx}\right) + f(u,x)\,. \tag{94}$$

$$\mathcal{B}_0(u) = 0,\ x = 0, \quad \mathcal{B}_1(u) = 0,\ x = L \tag{95}$$

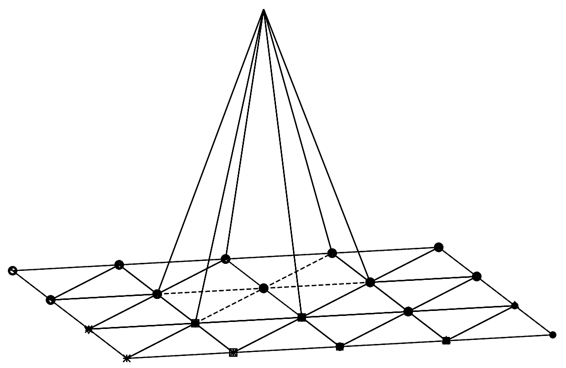


Figure 30: Example on a piecewise linear 2D basis function over a patch of triangles.

There are three common choices of boundary conditions:

$$\mathcal{B}_i(u) = u - g, \qquad \text{Dirichlet condition,} \tag{96}$$

$$\mathcal{B}_i(u) = -a\frac{du}{dx} - g, \qquad \text{Neumann condition,} \tag{97}$$

$$\mathcal{B}_i(u) = -a\frac{du}{dx} - a(u - g), \qquad \text{Robin condition}\,. \tag{98}$$

From now on we shall use $u_{\text{e}}(x)$ as symbol for the *exact* solution, fulfilling

$$\mathcal{L}(u_{\text{e}}) = 0, \quad x \in \Omega, \tag{99}$$

while $u(x)$ denotes an *approximate* solution of the differential equation.

10.7 Residual-minimizing principles

The fundamental idea is to seek an approximate solution u in some space V with basis

$$\{\psi_0(x), \ldots, \psi_N(x)\},$$

which means that u can always be expressed as

$$u(x) = \sum_{j\in\mathcal{I}} c_j\psi_j(x),$$

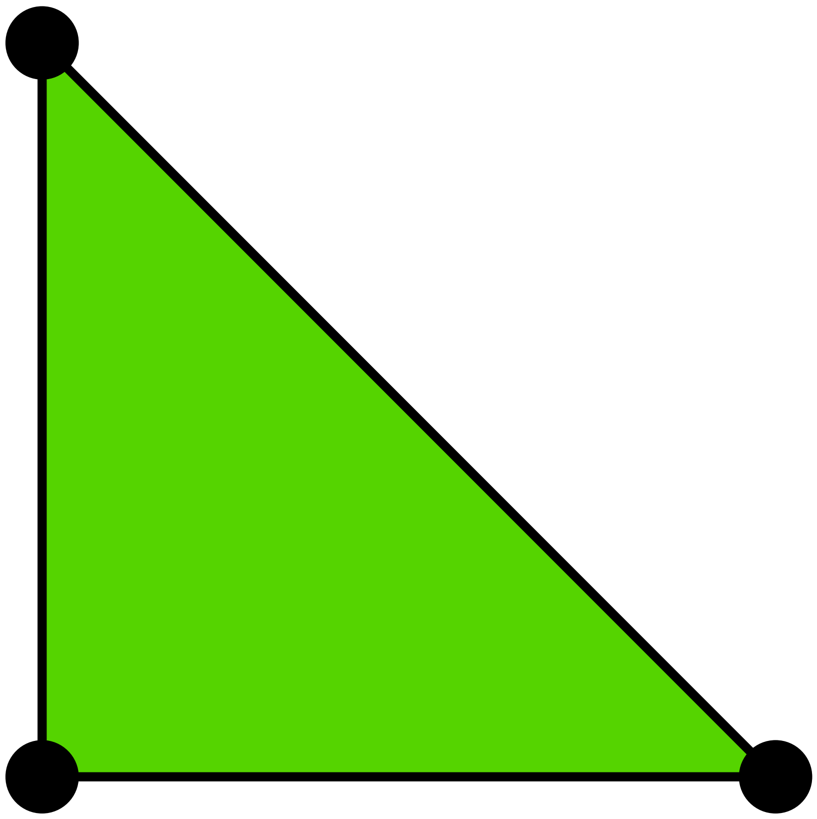


Figure 31: 2D P1 element.

for some unknown coefficients $c_0, \ldots, c_N$.

Inserting this u in the equation gives a nonzero *residual* R:

$$R = \mathcal{L}(u) = \mathcal{L}(\sum_j c_j \psi_j), \tag{100}$$

- R measures how well u fulfills the differential equation, but says nothing about the *error* $u_{\rm e} - u$
- We cannot know $u_{\rm e} - u$
- Therefore, we aim to minimize R
- Find $c_0, \ldots, c_N$ such that $R(x; c_0, \ldots, c_N)$ is small

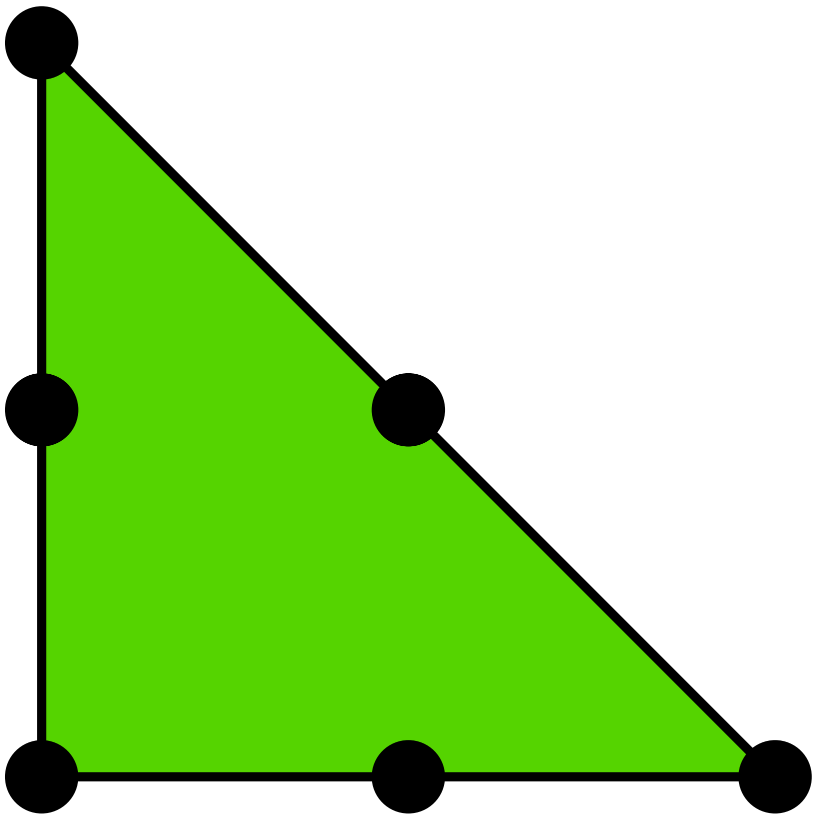


Figure 32: 2D P2 element.

The least squares method. Idea: minimize

$$\int_\Omega R^2 dx \tag{101}$$

With the inner product

$$(f,g) = \int_\Omega f(x)g(x)dx, \tag{102}$$

the least-squares method can be defined as

$$\min_{c_0,\ldots,c_N} E = (R,R)\,. \tag{103}$$

Differentiating with respect to the free parameters $c_0,\ldots,c_N$ gives the $N+1$ equations

$$\int_\Omega 2R\frac{\partial R}{\partial c_i}dx = 0 \quad \Leftrightarrow \quad (R, \frac{\partial R}{\partial c_i}) = 0, \quad i\in\mathcal{I}\,. \tag{104}$$

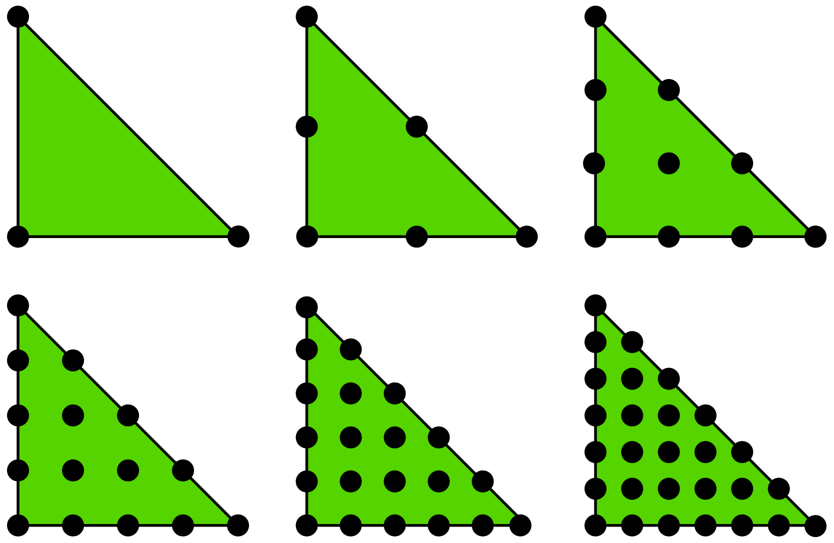


Figure 33: 2D P1, P2, P3, P4, P5, and P6 elements.

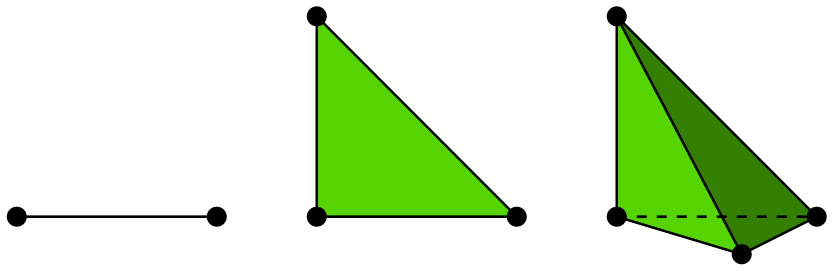


Figure 34: P1 elements in 1D, 2D, and 3D.

The Galerkin method. Idea: make R orthogonal to V,

$$(R, v) = 0, \quad \forall v \in V\,. \tag{105}$$

Equivalent statement:

$$(R, \psi_i) = 0, \quad i \in \mathcal{I}, \tag{106}$$

This statement generates $N + 1$ equations for $c_0, \ldots, c_N$.

The Method of Weighted Residuals. Generalization of the Galerkin method: demand R orthogonal to some space W, possibly $W \neq V$:

$$(R, v) = 0, \quad \forall v \in W\,. \tag{107}$$

If $\{w_0, \ldots, w_N\}$ is a basis for W, we can equivalently express the method of weighted residuals as

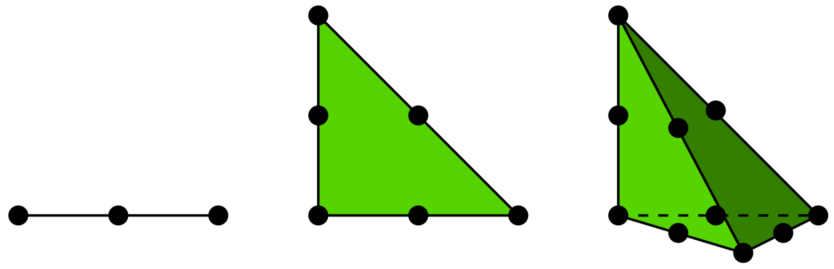


Figure 35: P2 elements in 1D, 2D, and 3D.

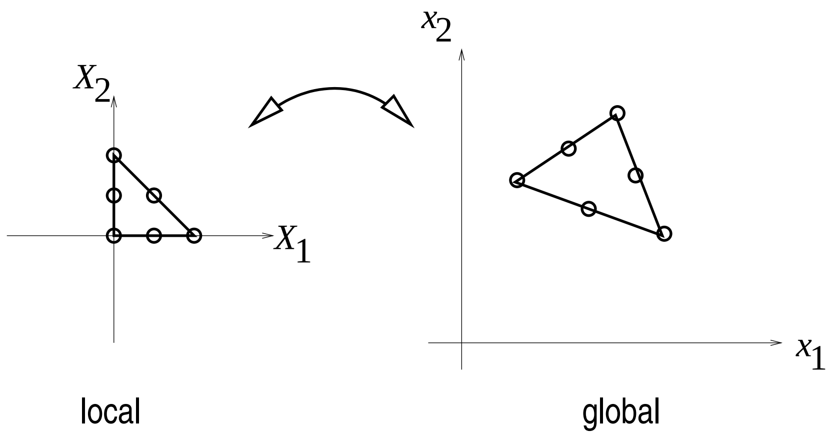


Figure 36: Affine mapping of a P1 element.

$$(R, w_i) = 0, \quad i \in \mathcal{I}\,. \tag{108}$$

This gives $N+1$ equations for $c_0, \ldots, c_N$.

Note: The least-squares method can also be viewed as a weighted residual method with $w_i = \partial R/\partial c_i$.

Test and Trial Functions.

- ψ_j used in $\sum_j c_j\psi_j$: *trial function*
- ψ_i or w_i used as weight in Galerkin's method: *test function*

The collocation method. Idea: demand $R = 0$ at $N+1$ points.

$$R(x_i; c_0, \ldots, c_N) = 0, \quad i \in \mathcal{I}\,. \tag{109}$$

Note: The collocation method is a weighted residual method with delta functions as weights.

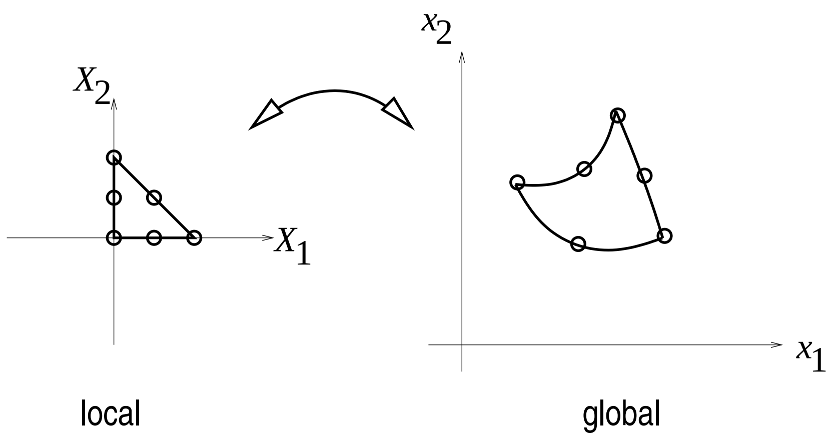


Figure 37: Isoparametric mapping of a P2 element.

$$\int_\Omega f(x)\delta(x-x_i)dx = f(x_i), \quad x_i \in \Omega\,. \tag{110}$$

10.8 Examples on using the principles

The model problem.

$$-u''(x) = f(x), \quad x \in \Omega = [0,L], \quad u(0)=0,\ u(L)=0\,. \tag{111}$$

Basis functions.

$$\psi_i(x) = \sin\left((i+1)\pi\frac{x}{L}\right), \quad i \in \mathcal{I}\,. \tag{112}$$

Note: $\psi_i(0)=\psi_i(L)=0$, which ensures that u fulfills the boundary conditions:

$$u(0) = \sum_j c_j\psi_j(0) = 0, \quad u(L) = \sum_j c_j\psi_j(L)\,.$$

Another useful property is the orthogonality on Ω:

$$\int\limits_0^L \sin\left((i+1)\pi\frac{x}{L}\right)\sin\left((j+1)\pi\frac{x}{L}\right)\,dx = \left\{\begin{array}{ll}\frac{1}{2}L & i=j\\ 0, & i\neq j\end{array}\right. \tag{113}$$

That is, the coefficient matrix becomes diagonal ($\psi_i\psi_j = 0$).

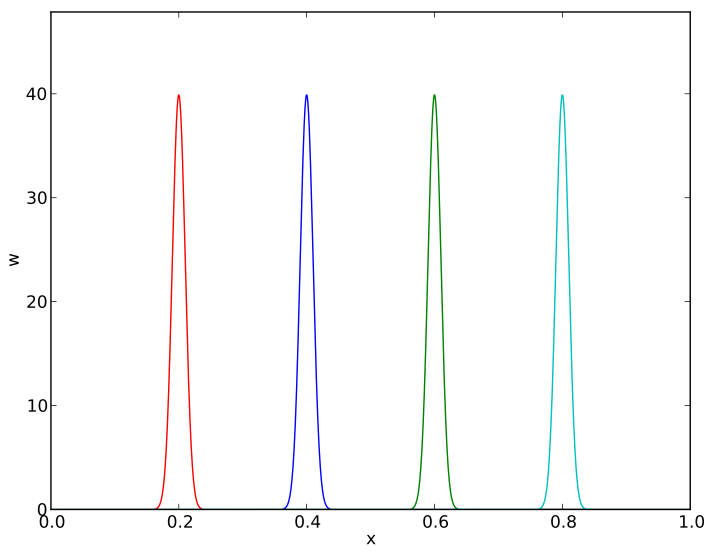


Figure 38: Approximation of delta functions by narrow Gaussian functions.

The residual.

$$
\begin{aligned}
R(x; c_0, \ldots, c_N) &= u''(x) + f(x),\\
&= \frac{d^2}{dx^2}\left(\sum_{j\in\mathcal{I}} c_j\psi_j(x)\right) + f(x),\\
&= -\sum_{j\in\mathcal{I}} c_j\psi_j''(x) + f(x)\,. \qquad (114)
\end{aligned}
$$

The least squares method.

$$
(R, \frac{\partial R}{\partial c_i}) = 0, \quad i\in\mathcal{I}\,.
$$

We need an expression for $\partial R/\partial c_i$:

$$
\frac{\partial R}{\partial c_i} = \frac{\partial}{\partial c_i}\left(\sum_{j\in\mathcal{I}} c_j\psi_j''(x) + f(x)\right) = \psi_i''(x)\,. \qquad (115)
$$

Because:

$$
\frac{\partial}{\partial c_i}\left(c_0\psi_0'' + c_1\psi_1'' + \cdots + c_{i-1}\psi_{i-1}'' + c_i\psi_i'' + c_{i+1}\psi_{i+1}'' + \cdots + c_N\psi_N''\right) = \psi_i''
$$

The governing equations for $c_0, \ldots, c_N$ are then

$$(\sum_j c_j\psi_j'' + f, \psi_i'') = 0, \quad i \in \mathcal{I}, \tag{116}$$

which can be rearranged as

$$\sum_{j\in\mathcal{I}} (\psi_i'', \psi_j'')c_j = -(f, \psi_i''), \quad i \in \mathcal{I}\,. \tag{117}$$

This is nothing but a linear system

$$\sum_{j\in\mathcal{I}} A_{i,j}c_j = b_i, \quad i \in \mathcal{I},$$

with

$$\begin{aligned}
A_{i,j} &= (\psi_i'', \psi_j'') \\
&= \pi^4(i+1)^2(j+1)^2L^{-4}\int_0^L \sin\left((i+1)\pi\frac{x}{L}\right)\sin\left((j+1)\pi\frac{x}{L}\right)\,dx \\
&= \left\{ \begin{array}{ll} \frac{1}{2}L^{-3}\pi^4(i+1)^4 & i = j \\ 0, & i \neq j \end{array} \right. && (118) \\
b_i &= -(f, \psi_i'') = (i+1)^2\pi^2L^{-2}\int_0^L f(x)\sin\left((i+1)\pi\frac{x}{L}\right)\,dx && (119)
\end{aligned}$$

Since the coefficient matrix is diagonal we can easily solve for

$$c_i = \frac{2L}{\pi^2(i+1)^2}\int_0^L f(x)\sin\left((i+1)\pi\frac{x}{L}\right)\,dx\,. \tag{120}$$

With the special choice of $f(x) = 2$ the integral becomes

$$\frac{L\cos(\pi i) + L}{\pi(i+1)},$$

The solution becomes:

$$u(x) = \sum_{k=0}^{N/2} \frac{8L^2}{\pi^3(2k+1)^3}\sin\left((2k+1)\pi\frac{x}{L}\right)\,. \tag{121}$$

The coefficients decay very fast: $c_2 = c_0/27$, $c_4 = c_0/125$. The first term therefore suffices:

$$u(x) \approx \frac{8L^2}{\pi^3}\sin\left(\pi\frac{x}{L}\right)\,.$$

The Galerkin method.

$$(u'' + f, v) = 0, \quad \forall v \in V,$$

or

$$(u'', v) = -(f, v), \quad \forall v \in V\,. \tag{122}$$

This is called a *variational formulation* of the differential equation problem. $\forall v \in V$ means for all basis functions:

$$(\sum_{j\in\mathcal{I}} c_j\psi_j'', \psi_i) = -(f, \psi_i), \quad i \in \mathcal{I}\,. \tag{123}$$

For the particular choice of the sine basis functions, we get in fact the same linear system as in the least squares method (because $\psi'' = -(i+1)^2\pi^2L^{-2}\psi$).

The collocation method. Residual must vanish at selected points, or equivalently, the differential equation with approximation u inserted, must be fulfilled at selected points:

$$-\sum_{j\in\mathcal{I}} c_j\psi_j''(x_i) = f(x_i), \quad i \in \mathcal{I}\,. \tag{124}$$

This is a linear system with entries

$$A_{i,j} = -\psi_j''(x_i) = (j+1)^2\pi^2L^{-2}\sin\left((j+1)\pi\frac{x_i}{L}\right),$$

and $b_i = 2$.

Special case: $N = 0$, $x_0 = L/2$

$$c_0 = 2L^2/\pi^2$$

Comparison.

- Exact solution: $u(x) = x(L - x)$
- Galerkin or least squares ($N = 0$): $u(x) = 8L^2\pi^{-3}\sin(\pi x/L)$
- Collocation method ($N = 0$): $u(x) = 2L^2\pi^{-2}\sin(\pi x/L)$.
- Max error in Galerkin/least sq.: $-0.008L^2$
- Max error in collocation: $0.047L^2$

10.9 Integration by parts

- Finite elements: $\psi_i = \psi_i$
- Problem: ψ_i' is discontinuous (at cell boundaries) and we need ψ_i'' in the Galerkin or least squares methods
- Remedy: integrate by parts - then we only need ψ_i'

Given

$$-(u'', v) = (f, v) \quad \forall v \in V\,.$$

Integrate by parts:

$$\begin{aligned}\int_0^L u''(x)v(x)dx &= -\int_0^L u'(x)v'(x)dx + [vu']_0^L \\ &= -\int_0^L u'(x)v'(x)dx + u'(L)v(L) - u'(0)v(0)\,. \end{aligned} \qquad (125)$$

Recall that $v(0) = v(L) = 0$, i.e., $\psi_i(0) = \psi_i(L) = 0$ because we demand so where we have Dirichlet conditions.

Advantageous features of integration by parts:

- Only first-order derivatives
- Symmatric coefficient matrix
- Incorporation of u' boundary conditions (later)

10.10 Boundary function

- What about nonzero Dirichlet conditions?
- E.g. $u(L) = D$
- Problem: $u(L) = \sum_j c_j\psi_j(L) = 0$ - always
- Remedy: $u(x) = B(x) + \sum_j c_j\psi_j(x)$
- $u(0) = B(0)$, $u(L) = B(L)$
- $B(x)$ must fulfill the Dirichlet conditions on u
- No restrictions of how $B(x)$ varies in the interior

Example. $u(0) = 0$ and $u(L) = D$. Choose

$$B(x) = \frac{D}{L}x: \qquad B(0) = 0,\ B(L) = D\,.$$

$$u(x) = \frac{x}{L}D + \sum_{j\in\mathcal{I}} c_j\psi_j(x), \qquad (126)$$

$$u(0) = 0, \quad u(L) = 0\,.$$

10.11 Abstract notation for variational formulations

The finite element literature (and much FEniCS documentation) applies an abstract notation for the variational formulation: *Find $u - B \in V$ such that

$$a(u,v) = L(v) \quad \forall v \in V\,.$$

Example. Given a variational formulation for $-u'' = f$:

$$\int_\Omega u'v'dx = \int_\Omega fvdx \quad \text{or} \quad (u',v') = (f,v) \quad \forall v \in V$$

we identify

$$a(u,v) = (u',v'), \quad L(v) = (f,v)\,.$$

Then we can write

$$a(u,v) = L(v) \quad \forall v \in V,$$

if

Bilinear and linear forms. $a(u,v)$ is a *bilinear form* and $L(v)$ is a *linear form.*

Linear form:

$$L(\alpha_1 v_1 + \alpha_2 v_2) = \alpha_1 L(v_1) + \alpha_2 L(v_2),$$

Bilinear form:

$$\begin{aligned} a(\alpha_1 u_1 + \alpha_2 u_2, v) &= \alpha_1 a(u_1,v) + \alpha_2 a(u_2,v),\\ a(u, \alpha_1 v_1 + \alpha_2 v_2) &= \alpha_1 a(u,v_1) + \alpha_2 a(u,v_2)\,. \end{aligned}$$

In nonlinear problems the abstract form is $F(u;v) = 0\ \forall v \in V$.

The abstract form $a(u,v) = L(v)$ is equivalent with a linear system

$$\sum_{j\in\mathcal{I}} A_{i,j}c_j = b_i, \quad i \in \mathcal{I}$$

with

$$\begin{aligned} A_{i,j} &= a(\psi_j,\psi_i),\\ b_i &= L(\psi_i)\,. \end{aligned}$$

10.12 More examples on variational formulations

Variable coefficient. Consider the problem

$$-\frac{d}{dx}\left(a(x)\frac{du}{dx}\right) = f(x), \quad x \in \Omega = [0,L],\ u(0) = C,\ u(L) = D\,. \tag{127}$$

Two new features:

- a variable coefficient $a(x)$
- nonzero Dirichlet conditions at $x = 0$ *and* $x = L$

A boundary function handles nonzero Dirichlet conditions:

$$u(x) = B(x) + \sum_{j\in\mathcal{I}} c_j\psi_i(x), \quad \psi_i(0) = \psi_i(L) = 0$$

One possible choice of B is:

$$B(x) = C + \frac{1}{L}(D - C)x\,.$$

The residual:

$$R = -\frac{d}{dx}\left(a\frac{du}{dx}\right) - f\,.$$

Galerkin's method:

$$(R, v) = 0, \quad \forall v \in V,$$

Written in terms of integrals:

$$\int_\Omega \left(\frac{d}{dx}\left(a\frac{du}{dx}\right) - f\right) v dx = 0, \quad \forall v \in V\,.$$

Integration by parts:

$$-\int_\Omega \frac{d}{dx}\left(a(x)\frac{du}{dx}\right) v dx = \int_\Omega a(x)\frac{du}{dx}\frac{dv}{dx}dx - \left[a\frac{du}{dx}v\right]_0^L\,.$$

Must have $v = 0$ where we have Dirichlet conditions: boundary terms vanish.
The final variational formulation:

$$\int_\Omega a(x)\frac{du}{dx}\frac{dv}{dx}dx = \int_\Omega f(x)v dx, \quad \forall v \in V,$$

Alternative, compact notation:

$$(au', v') = (f, v), \quad \forall v \in V\,.$$

The abstract notation is

$$a(u, v) = L(v) \quad \forall v \in V,$$

with

$$a(u, v) = (au', v'), \quad L(v) = (f, v)\,.$$

Do not mix the a in $a(\cdot,\cdot)$ (notation) and $a(x)$ (function name).
Can derive the linear system by inserting $u = B + \sum_j c_j\psi_j$ and $v = \psi_i$:

$$\sum_{j\in\mathcal{I}}(a\psi_j', \psi_i')c_j = (f, \psi_i) + (a(D-C)L^{-1}, \psi_i'), \quad i\in\mathcal{I},$$

or $\sum_j A_{i,j}c_j = b_i$ with

$$A_{i,j} = (a\psi_j', \psi_i') = \int_\Omega a(x)\psi_j'(x), \psi_i'(x)dx,$$
$$b_i = (f, \psi_i) + (a(D-C)L^{-1}, \psi_i') = \int_\Omega \left(f(x)\psi_i(x) + a(x)\frac{D-C}{L}\psi_i'(x)\right) dx\,.$$

First-order derivative in the equation and boundary condition. Model:

$$-u''(x) + bu'(x) = f(x), \quad x\in\Omega = [0, L],\ u(0) = C,\ u'(L) = E\,. \tag{128}$$

New features:

- first-order derivative u' in the equation
- boundary condition with u': $u'(L) = E$

Initial steps:

- Must force $\psi_i(0) = 0$ because of Dirichlet condition at $x = 0$
- Boundary function: $B(x) = C(L-x)/L$
- No requirements on $\psi_i(L)$ (no Dirichlet condition at $x = L$)

$$u = \frac{C}{L}(L-x) + \sum_{j\in\mathcal{I}} c_j\psi_i(x)\,.$$

Galerkin's method: multiply by v, integrate over Ω, integrate by parts.

$$(-u'' + bu' - f, v) = 0, \quad \forall v\in V,$$

$$(-u'', v) + (bu', v) - (f, v) = 0, \quad \forall v\in V,$$

$$(u', v') + (bu', v) = (f, v) + [u'v]_0^L, \quad \forall v\in V,$$

$$(u'v') + (bu', v) = (f, v) + Ev(L), \quad \forall v\in V,$$

when $[u'v]_0^L = u'(L)v(L) = Ev(L)$ because $v(0) = 0$ and $u'(L) = E$.

Important:

- The boundary term can be used to implement Neumann conditions

- Forgetting the boundary term implies the condition $u' = 0$ (!)
- Such conditions are called *natural boundary conditions*

Abstract notation:

$$a(u, v) = L(v) \quad \forall v \in V,$$

with the particular formulas

$$a(u, v) = (u', v') + (bu', v), \quad L(v) = (f + C, v) + Ev(L)\,.$$

Linear system: insert $u = B + \sum_j c_j \psi_j$ and $v = \psi_i$,

$$\sum_{j\in\mathcal{I}} \underbrace{((\psi_j', \psi_i') + (b\psi_j', \psi_i))}_{A_{i,j}} c_j = \underbrace{(f, \psi_i) + (bCL^{-1}, \psi_i') + E\psi_i(L)}_{b_i}\,.$$

Observation: $A_{i,j}$ is not symmetric because of the term

$$(b\psi_j', \psi_i) = \int_\Omega b\psi_j' \psi_i dx \neq \int_\Omega b\psi_i' \psi_j dx = (\psi_i', b\psi_j)\,.$$

10.13 Example on computing with Dirichlet and Neumann conditions

Let us solve

$$-u''(x) = f(x), \quad x \in \Omega = [0, 1], \quad u'(0) = C,\ u(1) = D,$$

- Use a *global* polynomial basis $\psi_i \sim x^i$ on $[0, 1]$
- Because of $u(1) = D$: $\psi_i(1) = 0$
- Basis: $\psi_i(x) = (1 - x)^{i+1}$, $i \in \mathcal{I}$
- $B(x) = Dx$

We have

$$A_{i,j} = (\psi_j, \psi_i) = \int_0^1 \psi_i'(x)\psi_j'(x)dx = \int_0^1 (i + 1)(j + 1)(1 - x)^{i+j} dx,$$

and

$$\begin{aligned} b_i &= (2, \psi_i) - (D, \psi_i') - C\psi_i(0) \\ &= \int_0^1 \left(2(1 - x)^{i+1} - D(i + 1)(1 - x)^i\right) dx - C\psi_i(0) \end{aligned}$$

With $N = 1$:

$$\left(\begin{array}{cc} 1 & 1 \\ 1 & 4/3 \end{array}\right)\left(\begin{array}{c} c_0 \\ c_1 \end{array}\right) = \left(\begin{array}{c} -C+D+1 \\ 2/3-C+D \end{array}\right)$$

$$c_0 = -C + D + 2, \quad c_1 = -1,$$

$$u(x) = 1 - x^2 + D + C(x-1)\,.$$

This is also the exact solution (as expected when V contains second-degree polynomials).

Nonlinear terms. The techniques used to derive variational forms also apply in nonlinear cases.

Consider

$$-\left(a(u)u'\right)' = f(u), \quad x \in [0,L],\ u(0)=0,\ u'(L)=E\,. \tag{129}$$

Using the Galerkin principle, we multiply by $v \in V$ and integrate,

$$-\int_0^L \frac{d}{dx}\left(a(u)\frac{du}{dx}\right) v dx = \int_0^L f(u) v\, dx \quad \forall v \in V\,.$$

Integration by parts is not affected by $a(u)$:

$$\int_0^L a(u)\frac{du}{dx}\frac{dv}{dx} dx = \int_0^L f(u)v\, dx + [avu']_0^L \quad \forall v \in V\,.$$

$[vu']_0^L = v(L)E$ since $v(0)=0$ and $u'(L)=E$.

$$(a(u)u', v') = (f(u), v) + a(L)v(L)E \quad \forall v \in V\,.$$

Since the problem is nonlinear, we cannot identify a *bilinear* form $a(u,v)$ and a *linear* form $L(v)$. An abstract notation is typically *find u such that*

$$F(u;v) = 0 \quad \forall v \in V,$$

here with

$$F(u;v) = (a(u)u', v') - (f(u), v) - a(L)v(L)E\,.$$

By inserting $u = \sum_j c_j \psi_j$ we get a *nonlinear system of algebraic equations* for the unknowns $c_0, \ldots, c_N$. Such systems must be solved by constructing a sequence of linear systems whose solutions converge to the solution of the nonlinear system. Frequently applied methods are Picard iteration and Newton's method.

10.14 Variational problems and optimization of functionals

If $a(u,v)=a(v,u)$, it can be shown that the variational statement $a(u,v)=L(v)\ \forall v\in V$ is equivalent to minimizing the functional

$$F(v)=\frac{1}{2}a(v,v)-L(v)$$

That is, find u such that

$$F(u)\le F(v)\quad \forall v\in V\,.$$

Traditional use of finite elements, especially in structural analysis, often starts with $F(v)$ and then derives $a(u,v)=L(v)$.

11 Computing with finite elements

Given

$$-u''(x)=2,\quad x\in(0,L),\ u(0)=u(L)=0,$$

with variational formulation

$$(u',v')=(2,v)\quad \forall v\in V\,.$$

Tasks:

- Solve for u using finite elements
- show all details
- Uniformly spaced nodes
- P1 elements

Since $u(0)=0$ and $u(L)=0$, $c_0=c_N=0$, and we can use a sum over basis functions associated with internal nodes only:

$$u(x)=\sum_{j=1}^{N-1}c_j\varphi_j(x)\,.$$

11.1 Computation in the global physical domain

We are to compute

$$A_{i,j}=\int_0^L\varphi_i'(x)\varphi_j'(x)dx,\quad b_i=\int_0^L 2\varphi_i(x)dx\,.$$

Need $\varphi_i'(x)$ in the formulas:

$$
\varphi_i'(x) = \begin{cases} 0, & x < x_{i-1}, \\ h^{-1}, & x_{i-1} \leq x < x_i, \\ -h^{-1}, & x_i \leq x < x_{i+1}, \\ 0, & x \geq x_{i+1} \end{cases} \tag{130}
$$

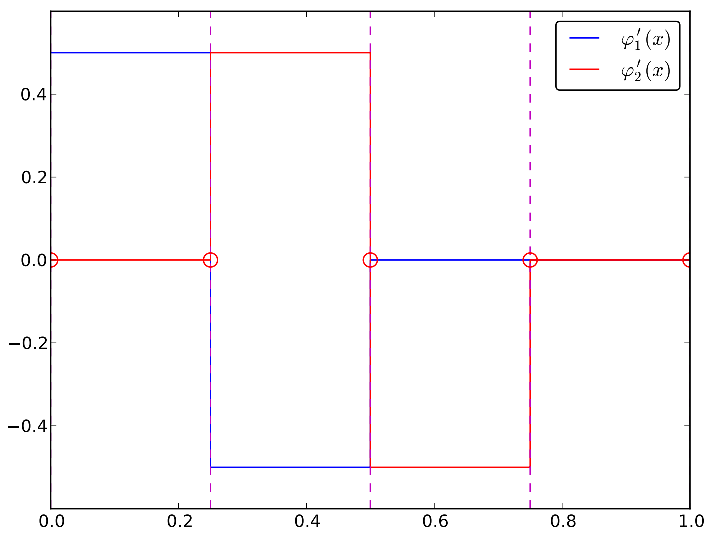


Figure 39: Illustration of the derivative of piecewise linear basis functions associated with nodes in cell 1.

We realize that φ_i' and φ_j' has no overlap, and hence their product vanishes, unless i and j are nodes belonging to the same element. The only nonzero contributions to the coefficient matrix are therefore

$$
\frac{1}{h}\begin{pmatrix}
2 & -1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
-1 & 2 & -1 & \ddots & & & & & \vdots \\
0 & -1 & 2 & -1 & \ddots & & & & \vdots \\
\vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\
\vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\
\vdots & & & 0 & -1 & 2 & -1 & \ddots & \vdots \\
\vdots & & & & \ddots & \ddots & \ddots & \ddots & 0 \\
\vdots & & & & & \ddots & \ddots & \ddots & -1 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & -1 & 2
\end{pmatrix}
\begin{pmatrix} c_1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_{N-1} \end{pmatrix}
=
\begin{pmatrix} 2h \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 2h \end{pmatrix}
\tag{131}
$$

$c_j = u(x_j)$ so we introduce $u_j = c_j$ to easily compare with the finite difference method. The equation corresponding to row i:

$$
-\frac{1}{h}u_{i-1} + \frac{2}{h}u_i - \frac{1}{h}u_{i+1} = 2h\,. \tag{132}
$$

Standard finite difference approximation of $-u''(x) = 2$, with $u''(x_i) \approx [D_xD_xu]_i$ and $\Delta x = h$, yields

$$
-\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} = 2, \tag{133}
$$

- The finite element and the finite difference method give the same equation (in this example)

11.2 Elementwise computations

We follow the same elementwise set-up as for approximating f by u.

Present element matrix:

$$
A^{(e)}_{i,j} = \int_{\Omega^{(e)}} \varphi_i'(x)\varphi_j'(x)dx = \int_{-1}^{1} \frac{d}{dx}\tilde{\varphi}_r(X)\frac{d}{dx}\tilde{\varphi}_s(X)\frac{h}{2}dX, \quad i = q(e,r),\ j = q(e,s),\ r,s = 1,2\,.
$$

$\tilde{\varphi}_r(X)$ are known as functions of X, but we need $d\tilde{\varphi}_r(X)/dx$.

Given

$$
\tilde{\varphi}_0(X) = \frac{1}{2}(1-X), \quad \tilde{\varphi}_1(X) = \frac{1}{2}(1+X),
$$

we can easily compute $d\tilde{\varphi}_r/dX$:

$$
\frac{d\tilde{\varphi}_0}{dX} = -\frac{1}{2}, \quad \frac{d\tilde{\varphi}_1}{dX} = \frac{1}{2}\,.
$$

From the chain rule,

$$\frac{d\tilde{\varphi}_r}{dx} = \frac{d\tilde{\varphi}_r}{dX}\frac{dX}{dx} = \frac{2}{h}\frac{d\tilde{\varphi}_r}{dX}\,. \tag{134}$$

The transformed integral is then:

$$A_{i,j}^{(e)} = \int_{\Omega^{(e)}} \varphi_i'(x)\varphi_j'(x)dx = \int_{-1}^{1} \frac{2}{h}\frac{d\tilde{\varphi}_r}{dX}\frac{2}{h}\frac{d\tilde{\varphi}_s}{dX}\frac{h}{2}dX\,.$$

The right-hand side is transformed according to

$$b_i^{(e)} = \int_{\Omega^{(e)}} 2\varphi_i(x)dx = \int_{-1}^{1} 2\tilde{\varphi}_r(X)\frac{h}{2}dX, \quad i = q(e,r),\ r = 1,2\,.$$

We have to compute the matrix entries one by one...

$$\begin{aligned}
\tilde{A}_{0,0}^{(e)} &= \int_{-1}^{1} \frac{2}{h}\left(-\frac{1}{2}\right)\frac{2}{h}\left(-\frac{1}{2}\right)\frac{2}{h}dX = \frac{1}{h}\\
\tilde{A}_{0,1}^{(e)} &= \int_{-1}^{1} \frac{2}{h}\left(-\frac{1}{2}\right)\frac{2}{h}\left(\frac{1}{2}\right)\frac{2}{h}dX = -\frac{1}{h}\\
\tilde{A}_{1,0}^{(e)} &= \int_{-1}^{1} \frac{2}{h}\left(\frac{1}{2}\right)\frac{2}{h}\left(-\frac{1}{2}\right)\frac{2}{h}dX = -\frac{1}{h}\\
\tilde{A}_{1,1}^{(e)} &= \int_{-1}^{1} \frac{2}{h}\left(\frac{1}{2}\right)\frac{2}{h}\left(\frac{1}{2}\right)\frac{2}{h}dX = \frac{1}{h}
\end{aligned}$$

The element vector entries become

$$\begin{aligned}
\tilde{b}_0^{(e)} &= \int_{-1}^{1} 2\frac{1}{2}(1-X)\frac{h}{2}dX = h\\
\tilde{b}_1^{(e)} &= \int_{-1}^{1} 2\frac{1}{2}(1+X)\frac{h}{2}dX = h\,.
\end{aligned}$$

In matrix/vector notation:

$$\tilde{A}^{(e)} = \frac{1}{h}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \quad \tilde{b}^{(e)} = h\begin{pmatrix} 1 \\ 1 \end{pmatrix}\,. \tag{135}$$

Must assemble - but first see how to incorporate boundary conditions.

12 Boundary conditions: specified value

12.1 General construction of a boundary function

- $B(x)$ is not always easy to construct (extend to the interior of Ω), at least not in 2D and 3D

- With finite element φ_i, $B(x)$ can be constructed in a completely general way

$$B(x) = \sum_{j\in D} U_j\varphi_j(x), \tag{136}$$

where D are the nodes with Dirichlet conditions and U_j the known values.

In 1D

$$B(x) = U_0\varphi_0(x) + U_N\varphi_N(x)\,. \tag{137}$$

Unknowns: $c_1,\ldots,c_{N-1}$,

$$u(x) = U_0\varphi_0(x) + U_N\varphi_N(x) + \sum_{j=1}^{N-1} c_j\varphi_j(x)\,. \tag{138}$$

Example.

$$-u'' = 2,\quad u(0)=0,\ u(L)=D\,.$$

The expansion for $u(x)$ reads

$$u(x) = 0\cdot\varphi_0(x) + D\varphi_N(x) + \sum_{j=1}^{N-1} c_j\varphi_j(x)\,.$$

Inserting this expression in $-(u'',\varphi_i)=(f,\varphi_i)$ and integrating by parts results in a linear system with

$$A_{i,j} = \int_0^L \varphi_i'(x)\varphi_j'(x)dx,\quad b_i = \int_0^L (f(x) - D\varphi_N'(x))\varphi_i(x)dx,$$

for $i,j=1,\ldots,N-1$.

12.2 Modification of the linear system

- $B(x)$ and a reduced set of unknowns (e.g., $c_1,\ldots,c_{N-1}$) are not so convenient in implementations
- We shall look at a less strict mathematical procedure that gives simpler impelementation
- Step 1: compute everything as there were no Dirichlet conditions
- Step 2: modify the linear system such that all known c_j get their right boundary values

Linear system from $-u''=f$ without taking Dirichlet conditions into account ($u=\sum_{j\in\mathcal{I}} c_j\varphi_j$):

$$
\frac{1}{h}\left(\begin{array}{ccccccccc}
1 & -1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
-1 & 2 & -1 & \ddots & & & & & \vdots \\
0 & -1 & 2 & -1 & \ddots & & & & \vdots \\
\vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\
\vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\
\vdots & & & 0 & -1 & 2 & -1 & \ddots & \vdots \\
\vdots & & & & \ddots & \ddots & \ddots & \ddots & 0 \\
\vdots & & & & & \ddots & \ddots & \ddots & -1 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & -1 & 1
\end{array}\right)
\left(\begin{array}{c}
c_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_N
\end{array}\right)
=
\left(\begin{array}{c}
h \\ 2h \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 2h \\ h
\end{array}\right)
\tag{139}
$$

Actions:

- General: replace row i by $c_i = K$ if u at x_i is prescrined as K
- Here: replace the first and last row by $c_0 = 0$ and $c_N = D$

$$
\frac{1}{h}\left(\begin{array}{ccccccccc}
1 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
-1 & 2 & -1 & \ddots & & & & & \vdots \\
0 & -1 & 2 & -1 & \ddots & & & & \vdots \\
\vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\
\vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\
\vdots & & & 0 & -1 & 2 & -1 & \ddots & \vdots \\
\vdots & & & & \ddots & \ddots & \ddots & \ddots & 0 \\
\vdots & & & & & \ddots & \ddots & \ddots & -1 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 & 1
\end{array}\right)
\left(\begin{array}{c}
c_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_N
\end{array}\right)
=
\left(\begin{array}{c}
0 \\ 2h \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 2h \\ D
\end{array}\right)
\tag{140}
$$

12.3 Symmetric modification of the linear system

- The modification above destroys symmetry of the matrix ($A_{0,1} \neq A_{1,0}$)
- Symmetry is often important in 2D and 3D (faster computations)
- A more complex modification preserves symmetry

Algorithm for incorporating $c_i = K$:

1. Subtract column i times K from the right-hand side
2. Zero out column and row no i

3. Place 1 on the diagonal
4. Set $b_i = K$

$$\frac{1}{h}\left(\begin{array}{cccccccccc}
1 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & 2 & -1 & \ddots & & & & & \vdots \\
0 & -1 & 2 & -1 & \ddots & & & & \vdots \\
\vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\
\vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\
\vdots & & & 0 & -1 & 2 & -1 & \ddots & \vdots \\
\vdots & & & & \ddots & \ddots & \ddots & \ddots & 0 \\
\vdots & & & & & \ddots & \ddots & \ddots & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 & 1
\end{array}\right)
\left(\begin{array}{c} c_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_N \end{array}\right)
=
\left(\begin{array}{c} 0 \\ 2h \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 2h + D/h \\ D \end{array}\right) \tag{141}$$

12.4 Modification of the element matrix and vector

- Modification of the linear system can be done in the the element matrix and vector instead
- Exactly the same procedure

Last degree of freedom in the last element is prescribed:

$$\tilde{A}^{(N-1)} = A = \frac{1}{h}\left(\begin{array}{cc} 1 & -1 \\ 0 & 1 \end{array}\right), \quad \tilde{b}^{(N-1)} = \left(\begin{array}{c} h \\ D \end{array}\right). \tag{142}$$

Or symmetric modification:

$$\tilde{A}^{(N-1)} = A = \frac{1}{h}\left(\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array}\right), \quad \tilde{b}^{(N-1)} = \left(\begin{array}{c} h + D/h \\ D \end{array}\right). \tag{143}$$

13 Boundary conditions: specified derivative

Focus now: how to incorporate $u'(0) = C$ with finite elements.

13.1 The variational formulation

Start with the Galerkin method:

$$\int_0^L (u''(x) + f(x))\varphi_i(x)dx = 0, \quad i \in \mathcal{I},$$

Integration of $u''\varphi_i$ by parts:

$$\int_0^L u'(x)'\varphi_i'(x)dx - (u'(L)\varphi_i(L) - u'(0)\varphi_i(0)) = \int_0^L f(x)\varphi_i(x)dx\,.$$

- Since $\varphi_i(L) = 0$, $u'(L)\varphi_i(L) = 0$
- $u'(0)\varphi_i(0) = C\varphi_i(0)$ since $u'(0) = C$

$$\int_0^L u'(x)\varphi_i'(x)dx + C\varphi_i(0) = \int_0^L f(x)\varphi_i(x)dx, \quad i \in \mathcal{I}\,.$$

Inserting

$$u(x) = B(x) + \sum_{j=0}^{N-1} c_j\varphi_j(x), \quad B(x) = D\varphi_N(x),$$

leads to the linear system

$$\sum_{j=0}^{N-1}\left(\int_0^L \varphi_i'(x)\varphi_j'(x)dx\right)c_j = \int_0^L \left(f(x)\varphi_i(x) - D\varphi_N'(x)\varphi_i(x)\right)dx - C\varphi_i(0), \tag{144}$$

for $i = 0, \ldots, N-1$.

Alternatively, we may just work with

$$u(x) = \sum_{j=0}^{N} c_j\varphi_j(x),$$

and modify the last equation to $c_N = D$ in the linear system.

The extra term with C affects only the element vector from the first element:

$$\tilde{A}^{(0)} = A = \frac{1}{h}\left(\begin{array}{cc} 1 & 1 \\ -1 & 1 \end{array}\right), \quad \tilde{b}^{(0)} = \left(\begin{array}{c} h - C \\ h \end{array}\right). \tag{145}$$

14 The finite element algorithm

The problem at hand determines the integrals in the variational formulation.

Request these functions from the user:

```
integrand_lhs(phi, r, s, x)
boundary_lhs(phi, r, s, x)
integrand_rhs(phi, r, x)
boundary_rhs(phi, r, x)
```

Given a mesh in terms of `vertices`, `cells`, and `dof_map`, the rest is (almost) automatic.

```
<Declare global matrix and rhs: A, b>

for e in range(len(cells)):

    # Compute element matrix and vector
    n = len(dof_map[e])  # no of dofs in this element
    h = vertices[cells[e][1]] - vertices[cells[e][1]]
    <Declare element matrix and vector: A_e, b_e>

    # Integrate over the reference cell
    points, weights = <numerical integration rule>
    for X, w in zip(points, weights):
        phi = <basis functions and derivatives at X>
        detJ = h/2
        x = <affine mapping from X>
        for r in range(n):
            for s in range(n):
                A_e[r,s] += integrand_lhs(phi, r, s, x)*detJ*w
            b_e[r] += integrand_rhs(phi, r, x)*detJ*w

    # Add boundary terms
    for r in range(n):
        for s in range(n):
            A_e[r,s] += boundary_lhs(phi, r, s, x)*detJ*w
        b_e[r] += boundary_rhs(phi, r, x)*detJ*w

    # Incorporate essential boundary conditions
    for r in range(n):
        global_dof = dof_map[e][r]
        if global_dof in essbc_dofs:
            # dof r is subject to an essential condition
            value = essbc_docs[global_dof]
            # Symmetric modification
            b_e -= value*A_e[:,r]
            A_e[r,:] = 0
            A_e[:,r] = 0
            A_e[r,r] = 1
            b_e[r] = value

    # Assemble
    for r in range(n):
        for s in range(n):
            A[dof_map[e][r], dof_map[e][r]] += A_e[r,s]
        b[dof_map[e][r] += b_e[r]

<solve linear system>
```

15 Variational formulations in 2D and 3D

How to do integration by parts is the major difference when moving to 2D and 3D.

Consider

$$\nabla^2 u \quad \text{or} \quad \nabla\cdot(a(\boldsymbol{x})\nabla u)\ .$$

with explicit 2D expressions

$$\nabla^2 u = \nabla\cdot\nabla u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2},$$

and

$$\nabla\cdot(a(\boldsymbol{x})\nabla u) = \frac{\partial}{\partial x}\left(a(x,y)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(a(x,y)\frac{\partial u}{\partial y}\right)\ .$$

The general rule for integrating by parts is

$$-\int_{\Omega} \nabla\cdot(a(\boldsymbol{x})\nabla u)v\,\mathrm{d}x = \int_{\Omega} a(\boldsymbol{x})\nabla u\cdot\nabla v\,\mathrm{d}x - \int_{\partial\Omega} a\frac{\partial u}{\partial n}v\,\mathrm{d}s, \tag{146}$$

- $\int_{\Omega}()\,\mathrm{d}x$: area (2D) or volume (3D) integral
- $\int_{\partial\Omega}()\,\mathrm{d}s$: line(2D) or surface (3D) integral

Let us divide the boundary into two parts:

- $\partial\Omega_N$, where we have Neumann conditions $-a\frac{\partial u}{\partial n} = g$, and
- $\partial\Omega_D$, where we have Dirichlet conditions $u = u_0$.

The test functions v are required to vanish on $\partial\Omega_D$.

Example. A general and widely appearing PDE problem:

$$\boldsymbol{v}\cdot\nabla u + \alpha u = \nabla\cdot(a\nabla u) + f, \quad \boldsymbol{x}\in\Omega, \tag{147}$$

$$u = u_0, \quad \boldsymbol{x}\in\partial\Omega_D, \tag{148}$$

$$-a\frac{\partial u}{\partial n} = g, \quad \boldsymbol{x}\in\partial\Omega_N\,. \tag{149}$$

- Known: a, α, f, u_0, and g.
- Second-order PDE: must have *exactly one boundary condition at each point of the boundary*
- $\partial\Omega_N \cup \partial\Omega_D =$ entire boundary

The unknown function can be expanded as

$$u = u_0 + \sum_{j\in\mathcal{I}} c_j\varphi_j\,.$$

Galerkin's method: multiply by $v\in V$ and integrate over Ω,

$$\int_{\Omega}(\boldsymbol{v}\cdot\nabla u + \alpha u)v\,\mathrm{d}x = \int_{\Omega}\nabla\cdot(a\nabla u)\;\mathrm{d}x + \int_{\Omega} fv\,\mathrm{d}x\,.$$

Integrate second-order term by parts,

$$\int_{\Omega}\nabla\cdot(a\nabla u)\,v\,\mathrm{d}x = -\int_{\Omega} a\nabla u\cdot\nabla v\,\mathrm{d}x + \int_{\partial\Omega} a\frac{\partial u}{\partial n}v\,\mathrm{d}s,$$

resulting in

$$\int_{\Omega}(\boldsymbol{v}\cdot\nabla u + \alpha u)v\,\mathrm{d}x = -\int_{\Omega} a\nabla u\cdot\nabla v\,\mathrm{d}x + \int_{\partial\Omega} a\frac{\partial u}{\partial n}v\,\mathrm{d}s + \int_{\Omega} fv\,\mathrm{d}x\,.$$

Note: $v \neq 0$ only on $\partial\Omega_N$:

$$\int_{\partial\Omega} a\frac{\partial u}{\partial n} v \,\mathrm{d}s = \int_{\partial\Omega_N} a\frac{\partial u}{\partial n} v \,\mathrm{d}s,$$

Insert flux condition $a\frac{\partial u}{\partial n} = -g$ on $\partial\Omega_N$:

$$-\int_{\partial\Omega_N} gv \,\mathrm{d}s\,.$$

The final variational form:

$$\int_{\Omega} (\boldsymbol{v}\cdot\nabla u + \alpha u)v \,\mathrm{d}x = -\int_{\Omega} a\nabla u\cdot\nabla v \,\mathrm{d}x - \int_{\partial\Omega} gv \,\mathrm{d}s + \int_{\Omega} fv \,\mathrm{d}x\,.$$

With inner product notation:

$$(\boldsymbol{v}\cdot\nabla u, v) + (\alpha u, v) = -(a\nabla u, \nabla v) - (g, v)_N + (f, v)\,.$$

$(g, v)_N$: line or surface integral over $\partial\Omega_N$.

Inserting the u expansion results in a linear system with

$$A_{i,j} = (\boldsymbol{v}\cdot\nabla\varphi_j, \varphi_i) + (\alpha\varphi_j, \varphi_i) + (a\nabla\varphi_j, \nabla\varphi_i)$$

$$b_i = (g, \varphi_i)_N + (f, \varphi_i) - (\boldsymbol{v}\cdot\nabla u_0, \varphi_i) + (\alpha u_0, \varphi_i) + (a\nabla u_0, \nabla\varphi_i),$$

15.1 Transformation to a reference cell in 2D and 3D

We consider an integral of the type

$$\int_{\Omega^{(e)}} a(\boldsymbol{x})\nabla\varphi_i\cdot\nabla\varphi_j \,\mathrm{d}x \qquad (150)$$

in the physical domain.

Goal: integrate this term over the reference cell.

Mapping from reference to physical coordinates:

$$\boldsymbol{x}(\boldsymbol{X}),$$

with Jacobian, J, given by

$$J_{i,j} = \frac{\partial x_j}{\partial X_i}\,.$$

- Step 1: $\mathrm{d}x \to \det J\,\mathrm{d}X$.
- Step 2: express $\nabla\varphi_i$ by an expression with $\tilde{\varphi}_r$ ($i = q(e, r)$)
- We want $\nabla_{\boldsymbol{x}}\tilde{\varphi}_r(\boldsymbol{X})$ (derivatives wrt $\boldsymbol{x}$)

- What we readily have: $\nabla_{\boldsymbol{X}}\tilde{\varphi}_r(\boldsymbol{X})$ (derivative wrt $\boldsymbol{X}$)
- Need to transform $\nabla_{\boldsymbol{X}}\tilde{\varphi}_r(\boldsymbol{X})$ to $\nabla_{\boldsymbol{x}}\tilde{\varphi}_r(\boldsymbol{X})$

Can derive

$$\begin{aligned}\nabla_{\boldsymbol{X}}\tilde{\varphi}_r &= J\cdot\nabla_{\boldsymbol{x}}\varphi_i,\\ \nabla_{\boldsymbol{x}}\varphi_i &= J^{-1}\cdot\nabla_{\boldsymbol{X}}\tilde{\varphi}_r\,.\end{aligned}$$

Integral transformation from physical to reference coordinates:

$$\int_{\Omega}^{(e)} a(\boldsymbol{x})\nabla_{\boldsymbol{x}}\varphi_i\cdot\nabla_{\boldsymbol{x}}\varphi_j\,\mathrm{d}x \int_{\tilde{\Omega}^r} a(\boldsymbol{x}(\boldsymbol{X}))(J^{-1}\cdot\nabla_{\boldsymbol{X}}\tilde{\varphi}_r)\cdot(J^{-1}\cdot\nabla\tilde{\varphi}_s)\det J\,\mathrm{d}X \quad (151)$$

16 Systems of differential equations

Consider $m+1$ unknown functions: $u^{(0)},\ldots,u^{(m)}$ governed by $m+1$ differential equations:

$$\begin{aligned}\mathcal{L}_0(u^{(0)},\ldots,u^{(m)}) &= 0,\\ &\vdots\\ \mathcal{L}_m(u^{(0)},\ldots,u^{(m)}) &= 0,\end{aligned}$$

16.1 Variational forms

- First approach: treat each equation as a scalar equation
- For equation no. i, use test function $v^{(i)}\in V^{(i)}$

$$\int_\Omega \mathcal{L}^{(0)}(u^{(0)},\ldots,u^{(m)})v^{(0)}\,\mathrm{d}x = 0, \quad (152)$$

$$\vdots \quad (153)$$

$$\int_\Omega \mathcal{L}^{(m)}(u^{(0)},\ldots,u^{(m)})v^{(m)}\,\mathrm{d}x = 0\,. \quad (154)$$

Terms with second-order derivatives may be integrated by parts, with Neumann conditions inserted in boundary integrals.

$$V^{(i)} = \mathrm{span}\{\varphi_0^{(i)},\ldots,\varphi_{N_i}^{(i)}\},$$

$$u^{(i)} = B^{(i)}(\boldsymbol{x}) + \sum_{j=0}^{N_i} c_j^{(i)}\varphi_j^{(i)}(\boldsymbol{x}),$$

Can derive m coupled linear systems for the unknowns $c_j^{(i)}$, $j=0,\ldots,N_i$, $i=0,\ldots,m$.

- Second approach: work with vectors (and vector notation)
- $\boldsymbol{u} = (u^{(0)}, \ldots, u^{(m)})$
- $\boldsymbol{v} = (u^{(0)}, \ldots, u^{(m)})$
- $\boldsymbol{u}, \boldsymbol{v} \in \boldsymbol{V} = V^{(0)} \times \cdots \times V^{(m)}$
- Note: if $\boldsymbol{B} = (B^{(0)}, \ldots, B^{(m)})$ is needed for nonzero Dirichlet conditions, $\boldsymbol{u} - \boldsymbol{B} \in \boldsymbol{V}$ (not $\boldsymbol{u}$ in $\boldsymbol{V}$)
- $\boldsymbol{\mathcal{L}}(\boldsymbol{u}) = 0$
- $\boldsymbol{\mathcal{L}}(\boldsymbol{u}) = (\mathcal{L}^{(0)}(\boldsymbol{u}), \ldots, \mathcal{L}^{(m)}(\boldsymbol{u}))$

The variational form is derived by taking the *inner product* of $\boldsymbol{\mathcal{L}}(\boldsymbol{u})$ and $\boldsymbol{v}$:

$$\int_\Omega \boldsymbol{\mathcal{L}}(\boldsymbol{u}) \cdot \boldsymbol{v} = 0 \quad \forall \boldsymbol{v} \in \boldsymbol{V}\,. \tag{155}$$

- Observe: this is a scalar equation (!).
- Can derive m independent equation by choosing m independent $\boldsymbol{v}$
- E.g.: $\boldsymbol{v} = (v^{(0)}, 0, \ldots, 0)$ recovers (152)
- E.g.: $\boldsymbol{v} = (0, \ldots, 0, v^{(m)}$ recovers (154)

16.2 A worked example

$$\mu\nabla^2 w = -\beta, \tag{156}$$

$$\kappa\nabla^2 T = -\mu||\nabla w||^2 \quad (= \mu\nabla w \cdot \nabla w)\,. \tag{157}$$

- Unknowns: $w(x, y)$, $T(x, y)$
- Known constants: μ, β, κ
- Application: fluid flow in a straight pipe, w is velocity, T is temperature
- Ω: cross section of the pipe
- Boundary conditions: $w = 0$ and $T = T_0$ on $\partial\Omega$
- Note: T depends on w, but w does not depend on T (one-way coupling)

16.3 Identical function spaces for the unknowns

Let $w, (T - T_0) \in V$ with test functions $v \in V$.

$$V = \text{span}\{\varphi_0(x, y), \ldots, \varphi_N(x, y)\},$$

$$w = \sum_{j=0}^{N} c_j^{(w)}\varphi_j, \quad T = T_0 + \sum_{j=0}^{N} c_j^{(T)}\varphi_j\,. \tag{158}$$

Variational form of each individual PDE. Inserting (158) in the PDEs, results in the residuals

$$R_w = \mu\nabla^2 w + \beta, \tag{159}$$
$$R_T = \kappa\nabla^2 T + \mu||\nabla w||^2\,. \tag{160}$$

Galerkin's method: make residual orthogonal to V,

$$\int_\Omega R_w v\,\mathrm{d}x = 0 \quad \forall v \in V,$$
$$\int_\Omega R_T v\,\mathrm{d}x = 0 \quad \forall v \in V\,.$$

Integrate by parts and use $v = 0$ on $\partial\Omega$ (Dirichlet conditions!):

$$\int_\Omega \mu\nabla w\cdot\nabla v\,\mathrm{d}x = \int_\Omega \beta v\,\mathrm{d}x \quad \forall v \in V, \tag{161}$$
$$\int_\Omega \kappa\nabla T\cdot\nabla v\,\mathrm{d}x = \int_\Omega \mu\nabla w\cdot\nabla w\,v\,\mathrm{d}x \quad \forall v \in V\,. \tag{162}$$

Compound scalar variational form.

- Test vector function $\boldsymbol{v} \in \boldsymbol{V} = V \times V$
- Take the inner product of $\boldsymbol{v}$ and the system of PDEs (and integrate)

$$\int_\Omega (R_w, R_T)\cdot\boldsymbol{v}\,\mathrm{d}x = 0 \quad \forall\boldsymbol{v} \in \boldsymbol{V}\,.$$

With $\boldsymbol{v} = (v_0, v_1)$:

$$\int_\Omega (R_w v_0 + R_T v_1)\,\mathrm{d}x = 0 \quad \forall\boldsymbol{v} \in \boldsymbol{V}\,.$$

$$\int_\Omega (\mu\nabla w\cdot\nabla v_0 + \kappa\nabla T\cdot\nabla v_1)\,\mathrm{d}x = \int_\Omega (\beta v_0 + \mu\nabla w\cdot\nabla w\,v_1)\,\mathrm{d}x, \quad \forall\boldsymbol{v} \in \boldsymbol{V} \tag{163}$$

Choosing $v_0 = v$ and $v_1 = 0$ gives the variational form (161), while $v_0 = 0$ and $v_1 = v$ gives (162).

Alternative inner product notation:

$$\mu(\nabla w, \nabla v) = (\beta, v) \quad \forall v \in V, \tag{164}$$
$$\kappa(\nabla T, \nabla v) = \mu(\nabla w\cdot\nabla w, v) \quad \forall v \in V\,. \tag{165}$$

Decoupled linear systems.

$$\sum_{j=0}^{N} A_{i,j}^{(w)} c_j^{(w)} = b_i^{(w)}, \quad i = 0,\ldots,N, \tag{166}$$

$$\sum_{j=0}^{N} A_{i,j}^{(T)} c_j^{(T)} = b_i^{(T)}, \quad i = 0,\ldots,N, \tag{167}$$

$$A_{i,j}^{(w)} = \mu(\nabla\varphi_j, \nabla\varphi_i), \tag{168}$$

$$b_i^{(w)} = (\beta, \varphi_i), \tag{169}$$

$$A_{i,j}^{(T)} = \kappa(\nabla\varphi_j, \nabla\varphi_i), \tag{170}$$

$$b_i^{(T)} = (\mu\nabla w_- \cdot (\sum_k c_k^{(w)}\nabla\varphi_k), \varphi_i)\,. \tag{171}$$

Matrix-vector form (alternative notation):

$$\mu K c^{(w)} = b^{(w)}, \tag{172}$$

$$\kappa K c^{(T)} = b^{(T)}, \tag{173}$$

where

$$\begin{aligned}
K_{i,j} &= (\nabla\varphi_j, \nabla\varphi_i),\\
b^{(w)} &= (b_0^{(w)},\ldots,b_N^{(w)}),\\
b^{(T)} &= (b_0^{(T)},\ldots,b_N^{(T)}),\\
c^{(w)} &= (c_0^{(w)},\ldots,c_N^{(w)}),\\
c^{(T)} &= (c_0^{(T)},\ldots,c_N^{(T)})\,.
\end{aligned}$$

- First solve the system for $c^{(w)}$
- Then solve the system for $c^{(T)}$

Coupled linear systems.

- Pretend two-way coupling, i.e., need to solve for w and T simultaneously
- Want to derive *one system* for $c_j^{(w)}$ and $c_j^{(T)}$, $j = 0,\ldots,N$
- The system is nonlinear because of $\nabla w \cdot \nabla w$
- Linearization: pretend an iteration where $\hat{w}$ is computed in the previous iteration and set $\nabla w \cdot \nabla w \approx \nabla\hat{w} \cdot \nabla w$ (so the term becomes linear in w)

$$\sum_{j=0}^{N} A_{i,j}^{(w,w)} c_j^{(w)} + \sum_{j=0}^{N} A_{i,j}^{(w,T)} c_j^{(T)} = b_i^{(w)}, \quad i = 0, \ldots, N, \tag{174}$$
$$\sum_{j=0}^{N} A_{i,j}^{(T,w)} c_j^{(w)} + \sum_{j=0}^{N} A_{i,j}^{(T,T)} c_j^{(T)} = b_i^{(T)}, \quad i = 0, \ldots, N, \tag{175}$$
$$A_{i,j}^{(w,w)} = \mu(\nabla\varphi_j, \varphi_i), \tag{176}$$
$$A_{i,j}^{(w,T)} = 0, \tag{177}$$
$$b_i^{(w)} = (\beta, \varphi_i), \tag{178}$$
$$A_{i,j}^{(w,T)} = \mu(\nabla w_- \cdot \nabla\varphi_j), \varphi_i), \tag{179}$$
$$A_{i,j}^{(T,T)} = \kappa(\nabla\varphi_j, \varphi_i), \tag{180}$$
$$b_i^{(T)} = 0\,. \tag{181}$$

Alternative notation:

$$\mu K c^{(w)} = b^{(w)}, \tag{182}$$
$$L c^{(w)} + \kappa K c^{(T)} = 0, \tag{183}$$

L is the matrix from the $\nabla w_- \cdot \nabla$ operator: $L_{i,j} = A_{i,j}^{(w,T)}$.

Corresponding block form:

$$\left(\begin{array}{cc} \mu K & 0 \\ L & \kappa K \end{array}\right) \left(\begin{array}{c} c^{(w)} \\ c^{(T)} \end{array}\right) = \left(\begin{array}{c} b^{(w)} \\ 0 \end{array}\right).$$

16.4 Different function spaces for the unknowns

- Generalization: $w \in V^{(w)}$ and $T \in V^{(T)}$, $V^{(w)} \neq V^{(T)}$
- This is called a *mixed finite element method*

$$V^{(w)} = \text{span}\{\varphi_0^{(w)}, \ldots, \varphi_{N_w}^{(w)}\},$$
$$V^{(T)} = \text{span}\{\varphi_0^{(T)}, \ldots, \varphi_{N_T}^{(T)}\}\,.$$

$$\int_\Omega \mu \nabla w \cdot \nabla v^{(w)}\,\mathrm{d}x = \int_\Omega \beta v^{(w)}\,\mathrm{d}x \quad \forall v^{(w)} \in V^{(w)}, \tag{184}$$
$$\int_\Omega \kappa \nabla T \cdot \nabla v^{(T)}\,\mathrm{d}x = \int_\Omega \mu \nabla w \cdot \nabla w\, v^{(T)}\,\mathrm{d}x \quad \forall v^{(T)} \in V^{(T)}\,. \tag{185}$$

Take the inner product with $\boldsymbol{v} = (v^{(w)}, v^{(T)})$ and integrate:

$$\int_\Omega (\mu \nabla w \cdot \nabla v^{(w)} + \kappa \nabla T \cdot \nabla v^{(T)})\,\mathrm{d}x = \int_\Omega (\beta v^{(w)} + \mu \nabla w \cdot \nabla w\, v^{(T)})\,\mathrm{d}x, \tag{186}$$

valid $\forall \boldsymbol{v} \in \boldsymbol{V} = V^{(w)} \times V^{(T)}$.

Index