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

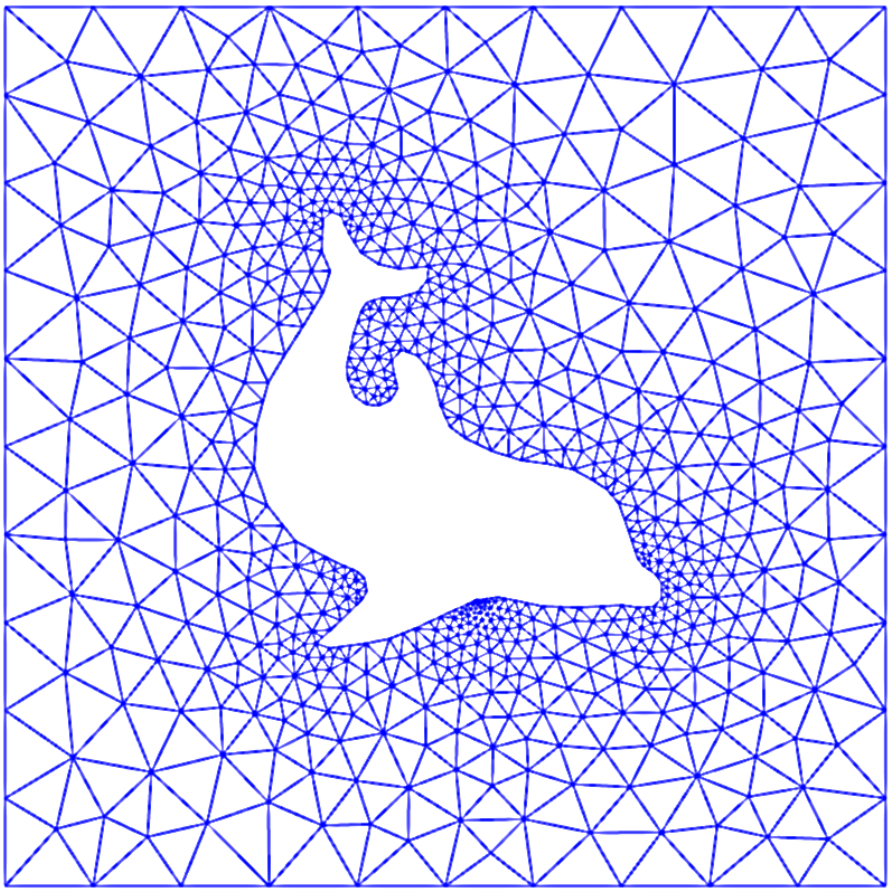
Oct 16, 2013

Contents

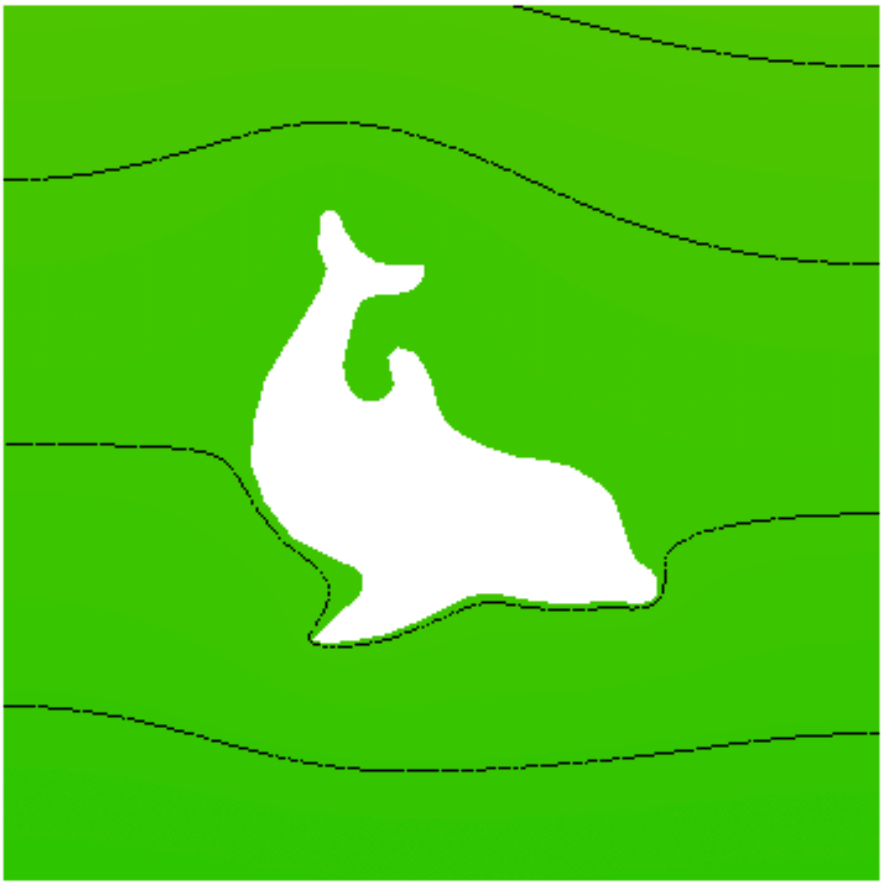
1 Why finite elements?

- Can with ease solve PDEs in domains with *complex geometry*
- Can with ease provide higher-order approximations
- Has (in simpler stationary problems) a rigorus mathematical analysis framework (not much considered here)

1.1 Domain for flow around a dolphin



1.2 The flow



1.3 Basic ingredients of the finite element method

- 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 Our learning strategy

- Start with approximation of functions, not PDEs
- Introduce finite element *approximations*
- See later 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.

1.5 Approximation set-up

General idea of finding an approximation $u(x)$ to some given $f(x)$:

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

where

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

1.6 How to determine the coefficients?

We shall address three approaches:

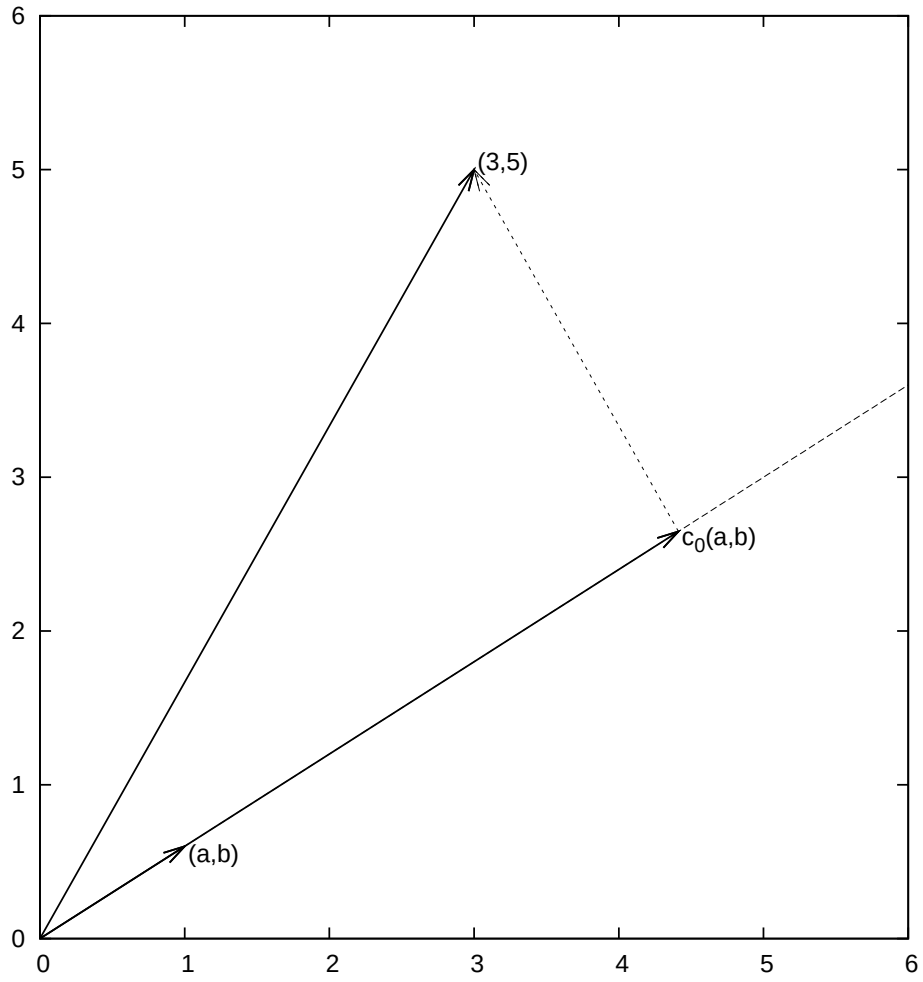
- The least squares method
- The projection (or Galerkin) method
- The interpolation (or collocation) method

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

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

1.7 Approximation of planar vectors; problem

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



1.8 Approximation of planar vectors; vector space terminology

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

- $\boldsymbol{\psi}_0$ 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

Define

- the error $\boldsymbol{e} = \boldsymbol{f} - \boldsymbol{u}$
- the (Eucledian) scalar product of two vectors: $(\boldsymbol{u}, \boldsymbol{v})$
- the norm of $\boldsymbol{e}$: $||\boldsymbol{e}|| = \sqrt{(\boldsymbol{e}, \boldsymbol{e})}$

1.9 The least squares method; principle

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

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

1.10 The least squares method; calculations

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

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

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

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

Observation for later: the vanishing derivative (4) can be alternatively written as

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

1.11 The projection (or Galerkin) 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 as in the least squares method

1.12 Approximation of general vectors

Given a vector $\boldsymbol{f}$, find an approximation $\boldsymbol{u} \in V$:

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

- 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 $\boldsymbol{u} = \sum_{j=0}^{N} c_j\boldsymbol{\psi}_j$

1.13 The least squares method

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

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

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

After some work we end up with a *linear system*

$$\sum_{j=0}^{N} A_{i,j} c_j = b_i, \quad i = 0, \ldots, N \tag{8}$$
$$A_{i,j} = (\boldsymbol{\psi}_i, \boldsymbol{\psi}_j) \tag{9}$$
$$b_i = (\boldsymbol{\psi}_i, \boldsymbol{f}) \tag{10}$$

1.14 The projection (or Galerkin) 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:

$$(\boldsymbol{e}, \boldsymbol{\psi}_i) = 0, \quad i = 0, \ldots, N \tag{11}$$

This orthogonality condition is the principle of the projection (or Galerkin) method. Leads to the same linear system as in the least squares method.

2 Approximation of functions

Let V be a *function space* spanned by a set of *basis functions* $\psi_0, \ldots, \psi_N$,

$$V = \text{span}\,\{\psi_0, \ldots, \psi_N\}$$

Find $u \in V$ as a linear combination of the basis functions:

$$u = \sum_{j \in I} c_j \psi_j, \quad I = \{0, 1, \ldots, N\} \tag{12}$$

2.1 The least squares method

- Extend the ideas from the vector case: minimize the (square) norm of the error.
- What norm? $(f, g) = \int_\Omega f(x)g(x)\,dx$

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

$$E(c_0, \ldots, c_N) = (f, f) - 2 \sum_{j \in I} c_j (f, \psi_i) + \sum_{p \in I} \sum_{q \in I} c_p c_q (\psi_p, \psi_q) \tag{14}$$

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

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

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

$$A_{i,j} = (\psi_i, \psi_j) \tag{16}$$

$$b_i = (f, \psi_i) \tag{17}$$

2.2 The projection (or Galerkin) method

As before, minimizing (e, e) is equivalent to the projection (or Galerkin) method

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

which means, as before,

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

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

2.3 Example: linear approximation; problem

Problem.
Approximate a parabola $f(x) = 10(x - 1)^2 - 1$ by a straight line.

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

That is, $\psi_0(x) = 1$, $\psi_1(x) = x$, and $N = 1$. We seek

$$u = c_0 \psi_0(x) + c_1 \psi_1(x) = c_0 + c_1 x$$

2.4 Example: linear approximation; solution

$$A_{0,0} = (\psi_0, \psi_0) = \int_1^2 1 \cdot 1 \, dx = 1 \tag{20}$$

$$A_{0,1} = (\psi_0, \psi_1) = \int_1^2 1 \cdot x \, dx = 3/2 \tag{21}$$

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

$$A_{1,1} = (\psi_1, \psi_1) = \int_1^2 x \cdot x \, dx = 7/3 \tag{23}$$

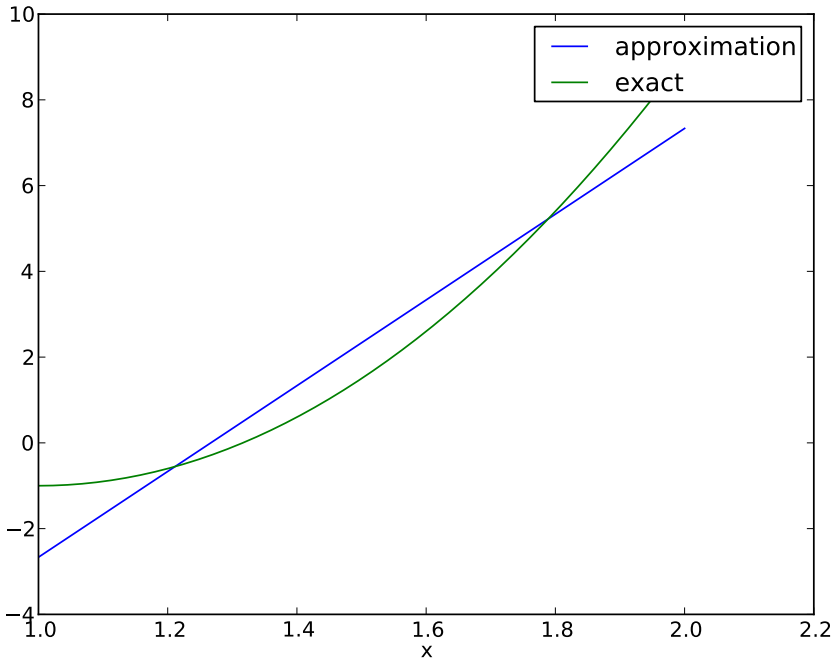
$$b_1 = (f, \psi_0) = \int_1^2 (10(x-1)^2 - 1) \cdot 1 \, dx = 7/3 \tag{24}$$

$$b_2 = (f, \psi_1) = \int_1^2 (10(x-1)^2 - 1) \cdot x \, dx = 13/3 \tag{25}$$

Solution of 2x2 linear system:

$$c_0 = -38/3, \quad c_1 = 10, \quad u(x) = 10x - \frac{38}{3} \tag{26}$$

2.5 Example: linear approximation; plot



2.6 Implementation of the least squares method; ideas

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 $\{\psi_i\}_{i \in I}$,
- `Omega` is a 2-tuple/list holding the domain Ω

Carry out the integrations, solve the linear system, and return $u(x) = \sum_j c_j \psi_j(x)$

2.7 Implementation of the least squares method; code

```
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 so we can halve the integrations.

2.8 Implementation of the least squares method; plotting

Compare f and u visually:

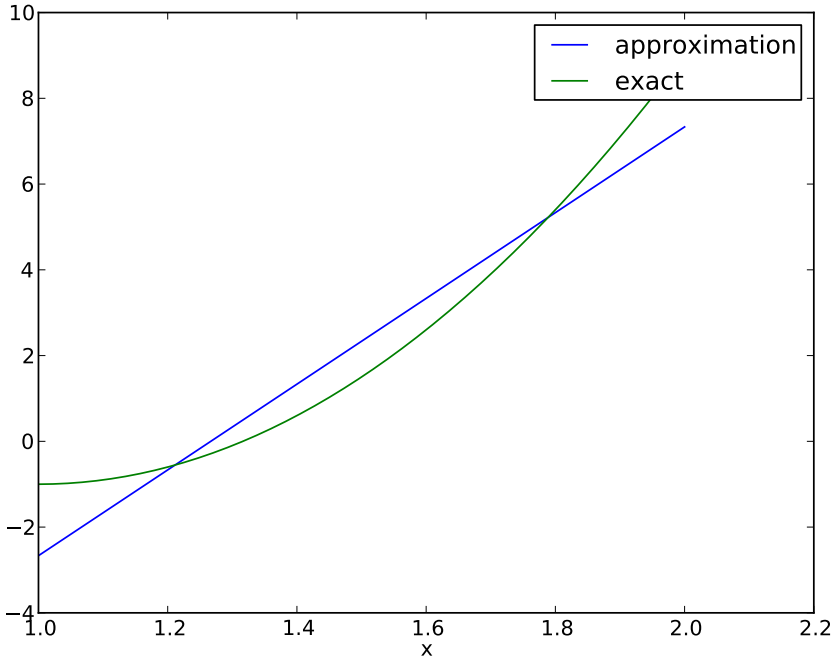
```
def comparison_plot(f, u, Omega, filename='tmp.pdf'):
    x = sm.Symbol('x')
    # Turn f and u to ordinary Python functions
    f = sm.lambdify([x], f, modules="numpy")
    u = sm.lambdify([x], u, modules="numpy")
    resolution = 401  # no of points in plot
    xcoor  = linspace(Omega[0], Omega[1],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

All code in module approx1D.py[1]

2.9 Implementation of the least squares method; application

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

[1] http://tinyurl.com/jvzzcfn/fem/approx1D.py



2.10 Perfect approximation; parabola approximating parabola

- What if we add $\psi_2 = x^2$ 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
```

2.11 Perfect approximation; the general result

- 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 projection/Galerkin give $u = f$.

2.12 Perfect approximation; proof of the general result

If $f \in V$, $f = \sum_{j\in I} d_j\psi_j$, for some $\{d_i\}_{i\in I}$. Then

$$b_i = (f, \psi_i) = \sum_{j\in I} d_j(\psi_j, \psi_i) = \sum_{j\in I} d_j A_{i,j}$$

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

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

which implies that $c_i = d_i$ for $i \in I$ and u is identical to f.

2.13 Finite-precision/numerical computations

The previous computations were symbolic. What if we solve the linear system numerically with standard 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

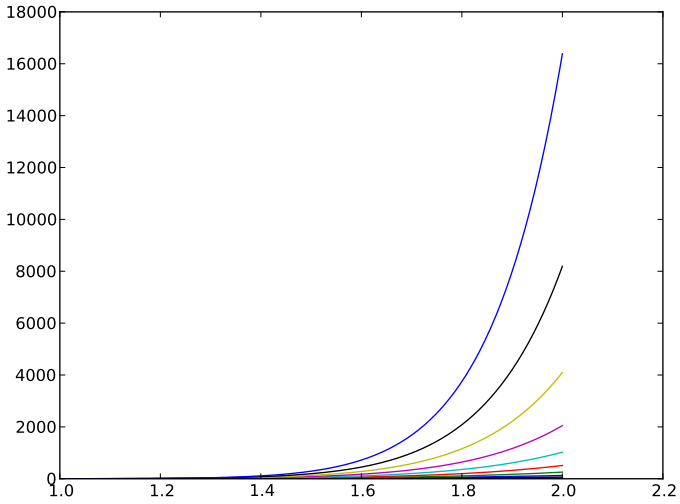
- Column 2: `sympy.mpmath.fp.matrix` and `sympy.mpmath.fp.lu_solve`
- Column 3: `numpy` arrays with `numpy.float32` entries
- Column 4: `numpy` arrays with `numpy.float64` entries

2.14 Ill-conditioning (1)

Observations:

- Significant round-off errors in the numerical computations (!)
- But if we plot the approximations they look good (!)

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



2.15 Ill-conditioning (2)

- Almost linearly dependent basis functions give almost singular matrices
- Such matrices are said to be *ill conditioned*, and Gaussian elimination is severely affected by round-off errors
- The basis $1, x, x^2, x^3, x^4, \ldots$ is a bad basis
- Polynomials are fine as basis, but the more orthogonal they are, $(\psi_i, \psi_j) \approx 0$, the better

2.16 Fourier series approximation; problem and code

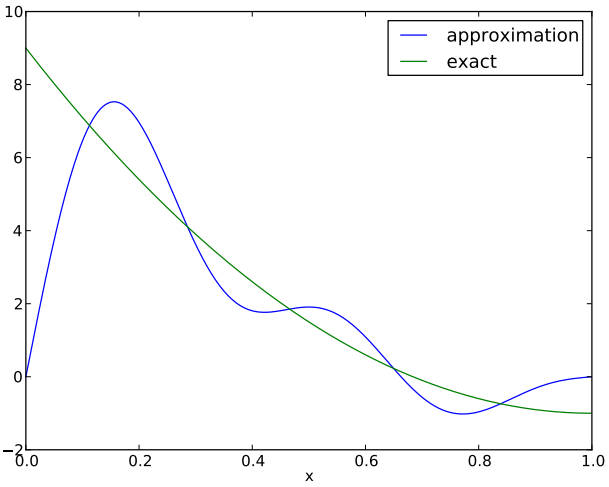
Consider

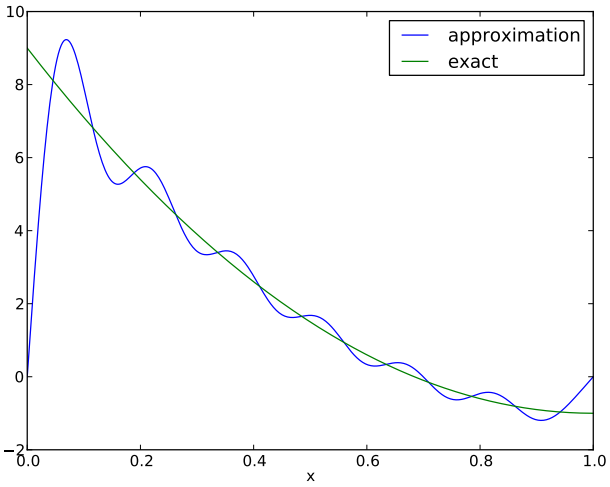
$$V = \text{span}\,\{\sin \pi x, \sin 2\pi x, \ldots, \sin(N+1)\pi x\}$$

```
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)
```

2.17 Fourier series approximation; plot

$N = 3$ vs $N = 11$:





2.18 Fourier series approximation; improvements

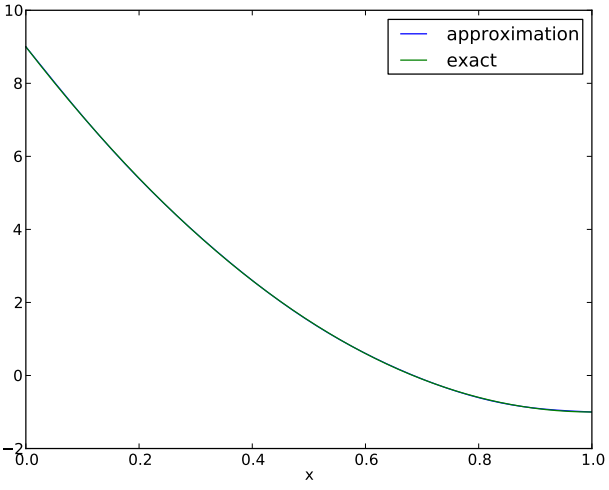
- Considerably improvement by $N = 11$
- But always discrepancy of $f(0) - u(0) = 9$ at $x = 0$, because all the $\psi_i(0) = 0$ and hence $u(0) = 0$
- Possible remedy: add a term that leads to correct boundary values

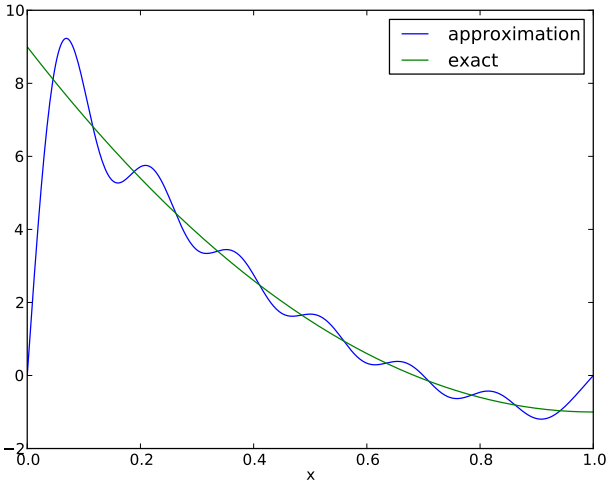
$$u(x) = f(0)(1 - x) + xf(1) + \sum_{j \in I} c_j \psi_j(x) \tag{27}$$

The extra term ensures $u(0) = f(0)$ and $u(1) = f(1)$ and is a strikingly good help to get a good approximation!

2.19 Fourier series approximation; final results

$N = 3$ vs $N = 11$:





2.20 Orthogonal basis functions

This choice of sine functions as basis functions is popular because

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

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

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

2.21 The collocation or interpolation method; ideas and math

Here is another idea for approximating $f(x)$ by $u(x) = \sum_j c_j\psi_j$:

- Force $u(x_i) = f(x_i)$ at some selected *collocation* points $\{x_i\}_{i\in I}$
- Then u interpolates f
- The method is known as *interpolation* or *collocation*

$$u(x_i) = \sum_{j\in I} c_j\psi_j(x_i) = f(x_i) \quad i \in I, N \tag{28}$$

This is a linear system with no need for integration:

$$\sum_{j\in I} A_{i,j}c_j = b_i, \quad i \in I \tag{29}$$

$$A_{i,j} = \psi_j(x_i) \tag{30}$$

$$b_i = f(x_i) \tag{31}$$

No symmetric matrix: $\psi_j(x_i) \neq \psi_i(x_j)$ in general

2.22 The collocation or interpolation method; implementation

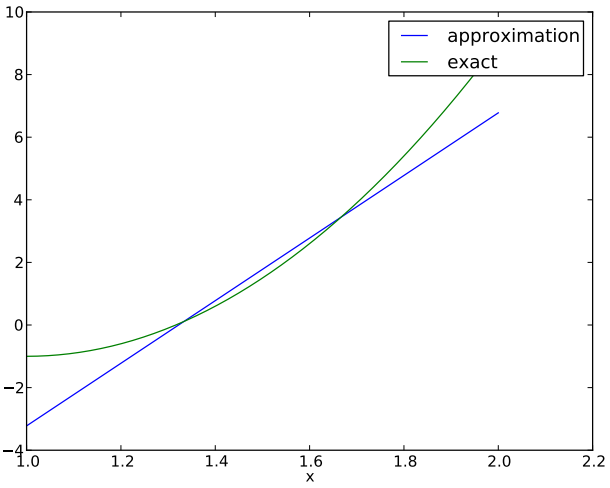
`points` holds the interpolation/collocation points

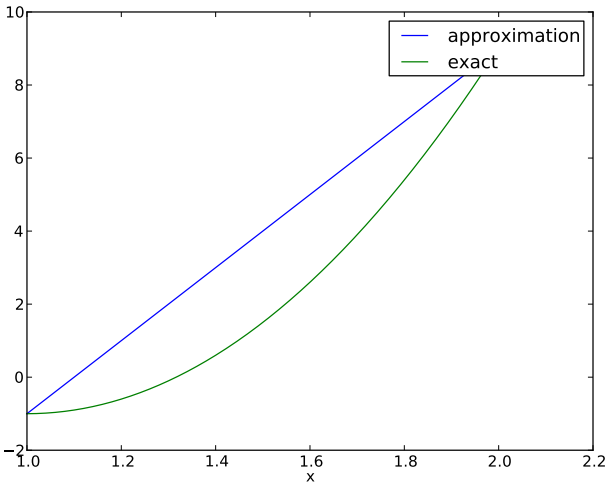
```
def interpolation(f, phi, points):
    N = len(phi) - 1
    A = sm.zeros((N+1, N+1))
    b = sm.zeros((N+1, 1))
    x = sm.Symbol('x')
    # Turn phi and f into Python functions
    phi = [sm.lambdify([x], phi[i]) for i in range(N+1)]
    f = sm.lambdify([x], f)
    for i in range(N+1):
        for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

2.23 The collocation or interpolation method; approximating a parabola by linear functions

- Potential difficulty: how to choose x_i?
- The results are sensitive to the points!

$(4/3, 5/3)$ vs $(1, 2)$:





2.24 Lagrange polynomials; motivation and ideas

Motivation:

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

The *Lagrange interpolating polynomials* ψ_j have the property that

$$\varphi_i(x_j) = \delta_{ij}, \quad \delta_{ij} = \begin{cases} 1, & i = j, \\ 0, & i \neq j, \end{cases}$$

Hence, $c_i = f(x_i)$ and

$$u(x) = \sum_{j \in I} f(x_i)\psi_i(x) \tag{32}$$

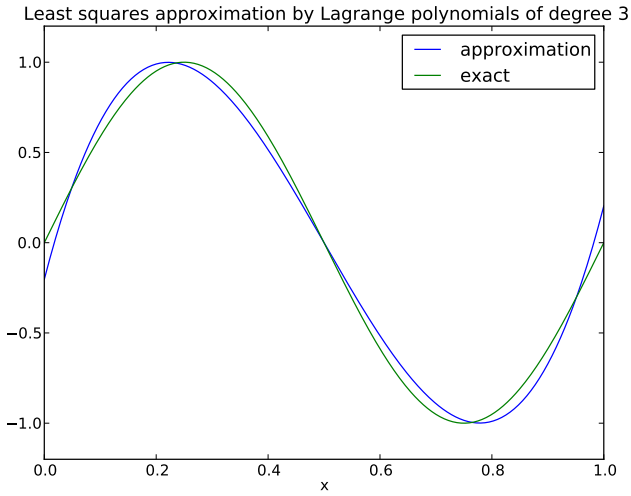
- Lagrange polynomials and interpolation/collocation look convenient
- Lagrange polynomials are very much used in the finite element method

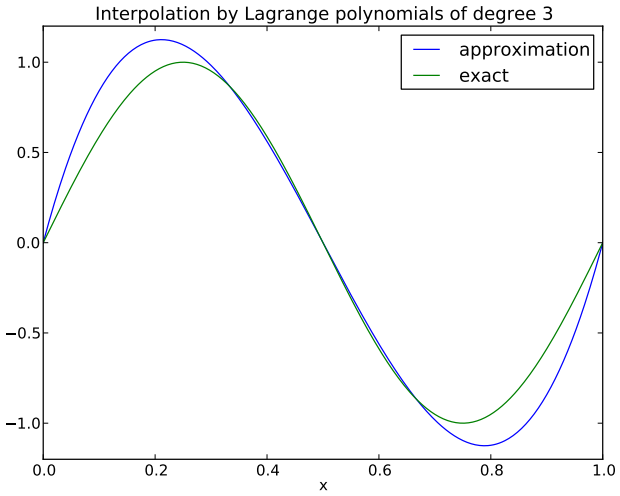
2.25 Lagrange polynomials; formula and code

$$\psi_i(x) = \prod_{j=0, j\neq i}^{N} \frac{x - x_j}{x_i - x_j} = \frac{x - x_0}{x_i - x_0} \cdots \frac{x - x_{i-1}}{x_i - x_{i-1}} \frac{x - x_{i+1}}{x_i - x_{i+1}} \cdots \frac{x - x_N}{x_i - x_N} \tag{33}$$

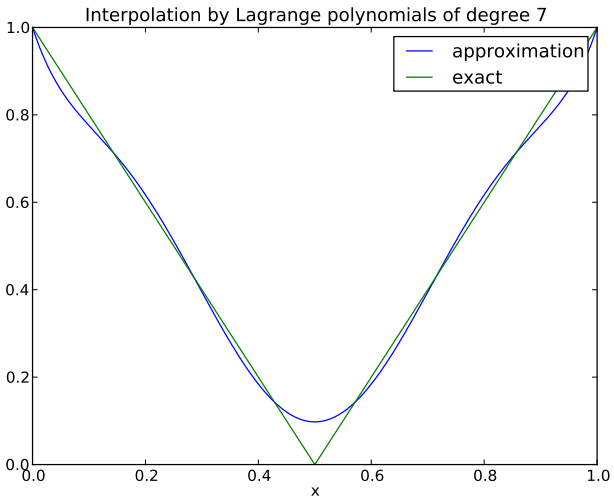
```
def Lagrange_polynomial(x, i, points):
    p = 1
    for k in range(len(points)):
        if k != i:
            p *= (x - points[k])/(points[i] - points[k])
    return p
```

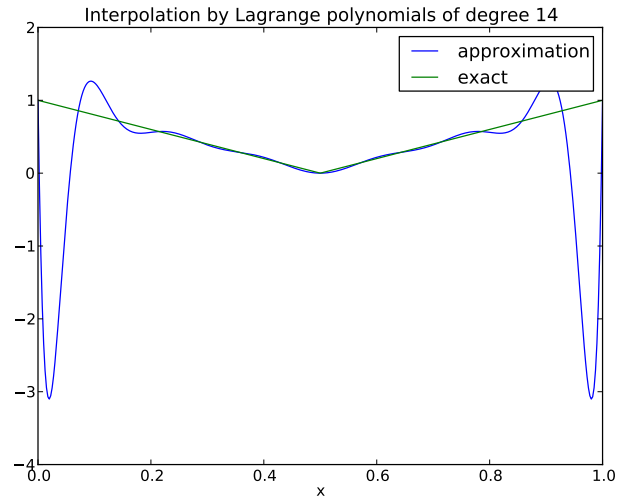
2.26 Lagrange polynomials; successful example





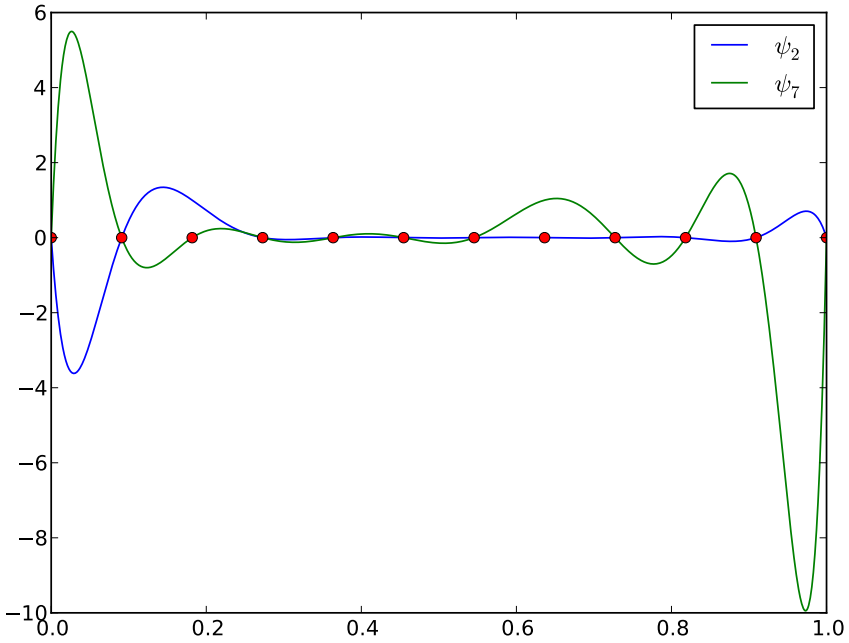
2.27 Lagrange polynomials; a less successful example





2.28 Lagrange polynomials; oscillatory behavior

12 points, degree 11, plot of two of the Lagrange polynomials - note that they are zero at all points except one.



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

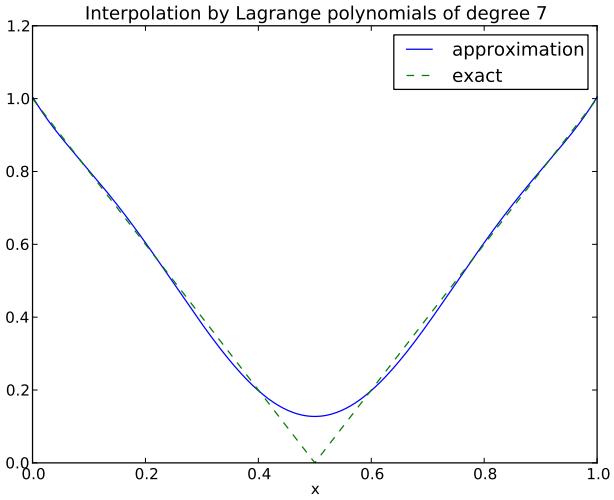
2.29 Lagrange polynomials; remedy for strong oscillations

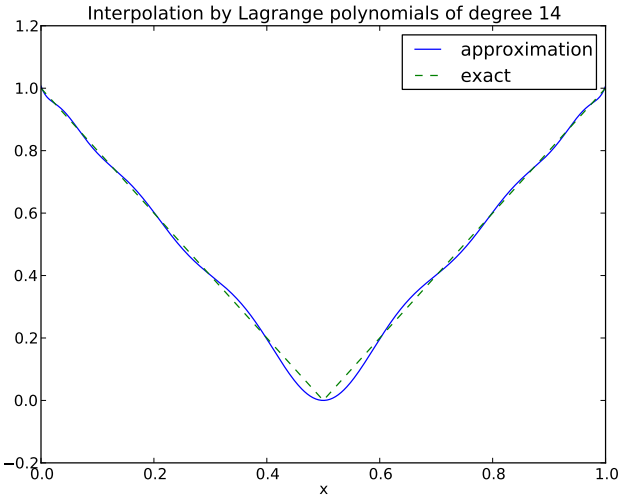
The oscillations can be reduced by a more clever choice of interpolation points, called the *Chebyshev nodes*:

$$x_i = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{2i+1}{2(N+1)}pi\right), \quad i = 0\ldots, N \tag{34}$$

on an interval $[a, b]$.

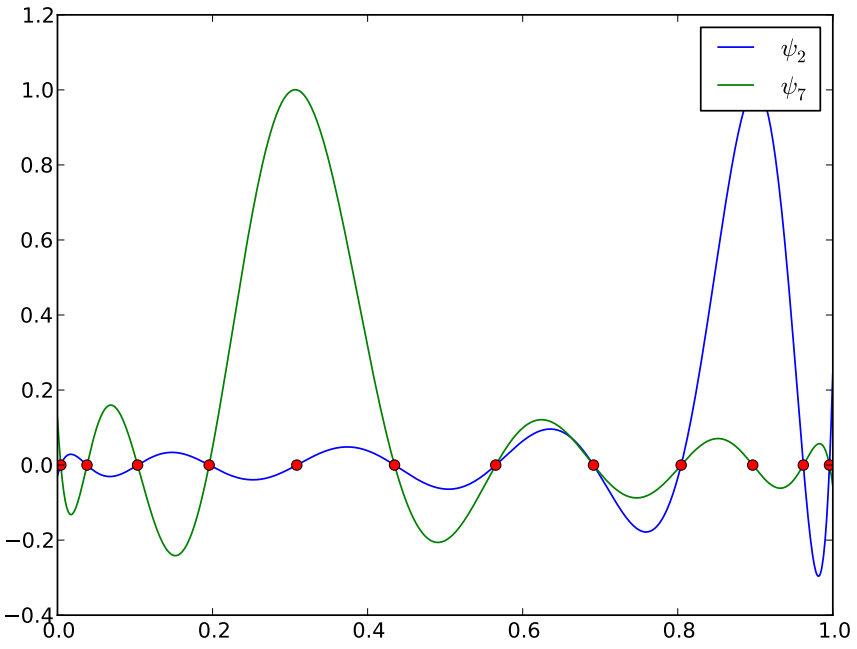
2.30 Lagrange polynomials; recalculation with Chebyshev nodes





2.31 Lagrange polynomials; less oscillations with Chebyshev nodes

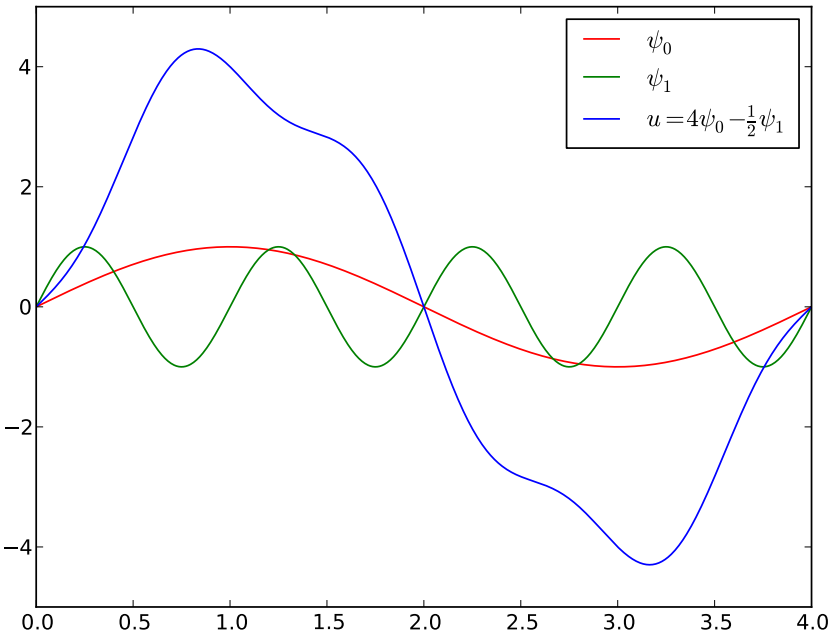
12 points, degree 11, plot of two of the Lagrange polynomials - note that they are zero at all points except one.



3 Finite element basis functions

3.1 So far: basis functions have been global

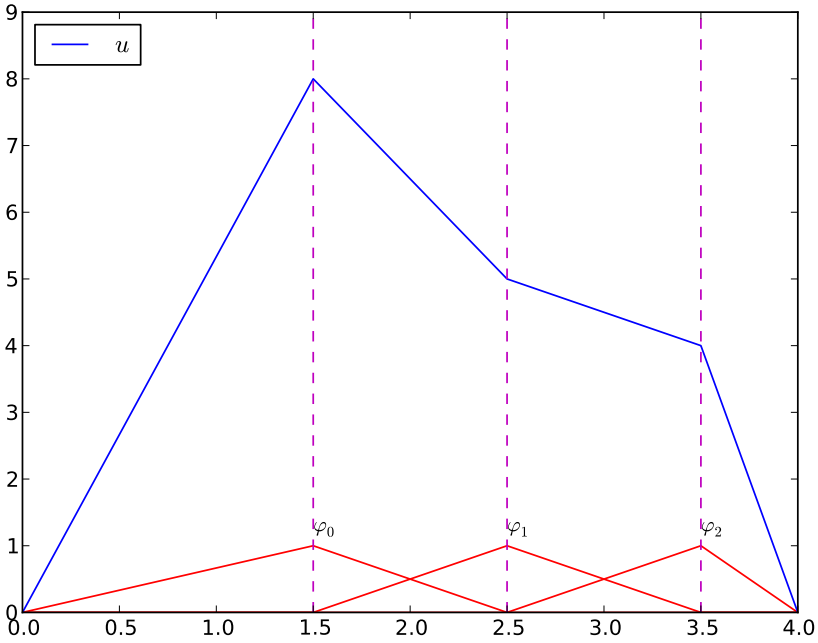
$\psi_i(x) \neq 0$ for most $x \in \Omega$



3.2 In the finite element method we use basis functions with local support

- *Local support*: $\psi_i(x) \neq 0$ for x in a small subdomain of Ω
- Typically hat-shaped
- $u(x)$ based on these ψ_i is a piecewise polynomial defined over many (small) subdomains

3.3 The linear combination of hat functions is a piecewise linear function



3.4 Elements and nodes

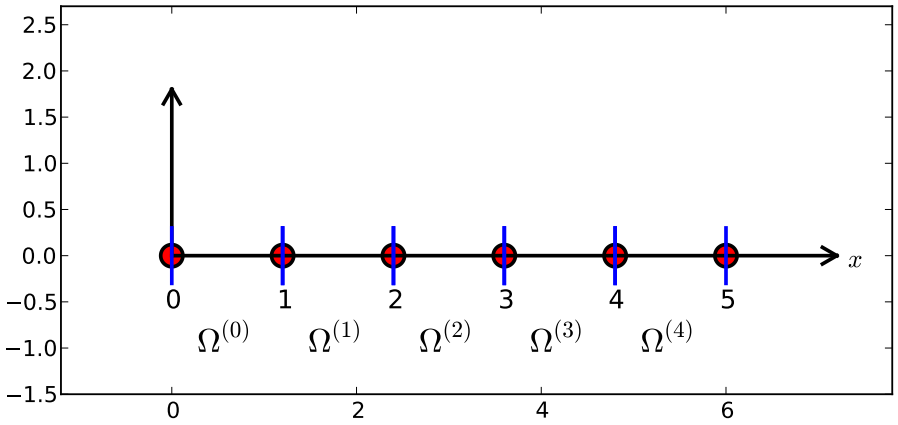
Split Ω into non-overlapping subdomains called *elements*:

$$\Omega = \Omega^{(0)} \cup \cdots \cup \Omega^{(N_e)} \tag{35}$$

On each element, introduce points called *nodes*: $x_0, \ldots, x_{N_n}$

- The finite element basis functions are named $\varphi_i(x)$
- $\varphi_i = 1$ 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 over each element

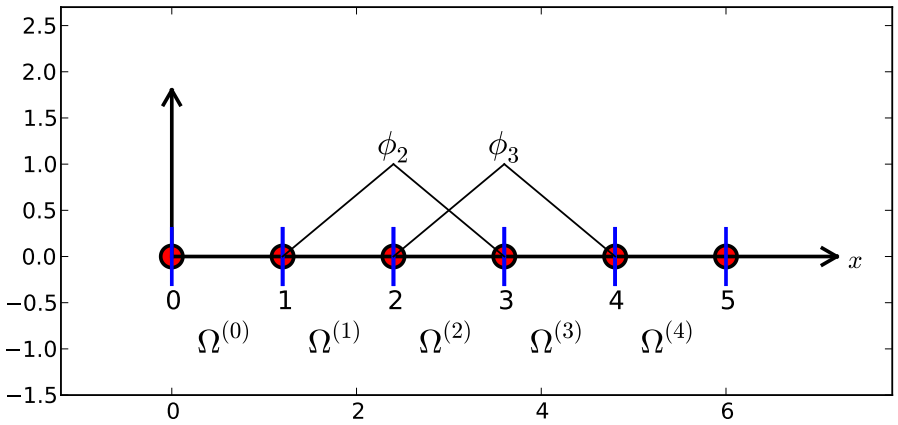
3.5 Example on elements with two nodes (P1 elements)



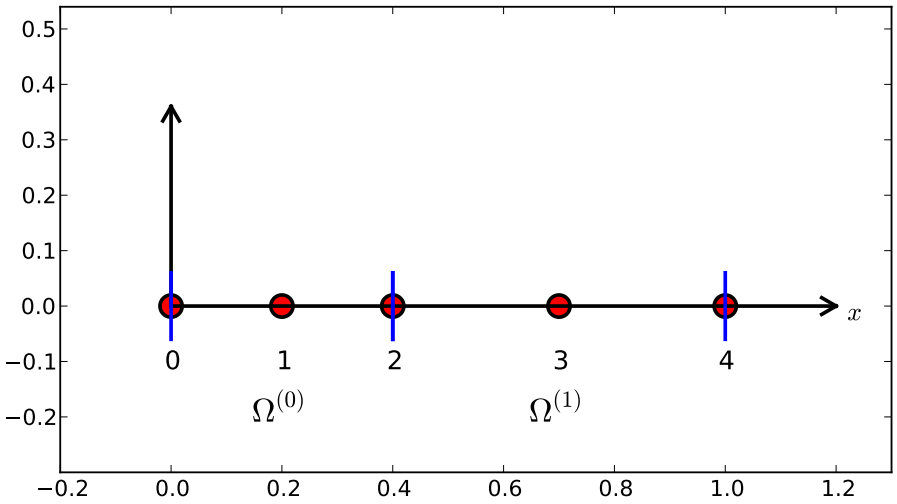
Data structure: `nodes` holds coordinates or nodes, `elements` holds the node numbers in each element

```
nodes = [0, 1.2, 2.4, 3.6, 4.8, 5]
elements = [[0, 1], [1, 2], [2, 3], [3, 4], [4, 5]]
```

3.6 Illustration of two basis functions on the mesh

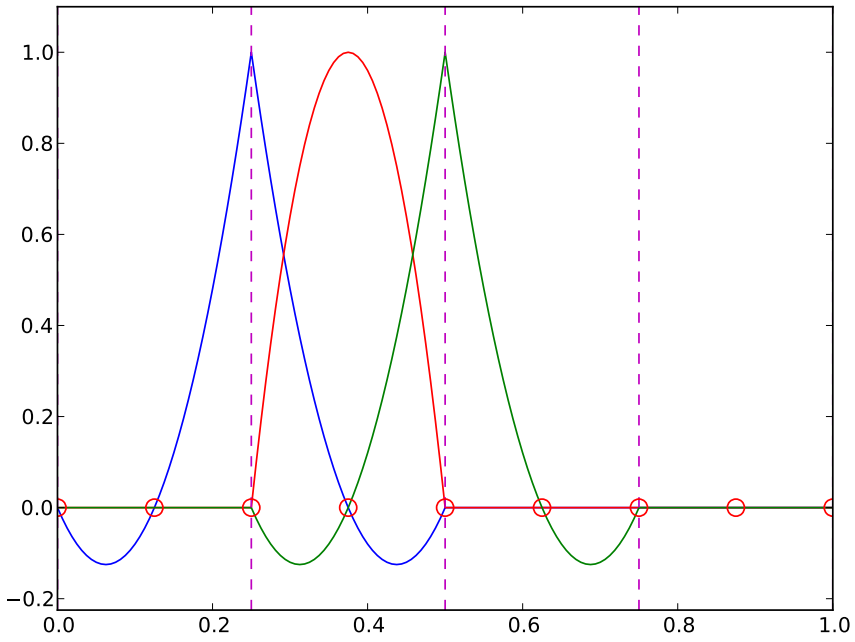


3.7 Example on elements with three nodes (P2 elements)

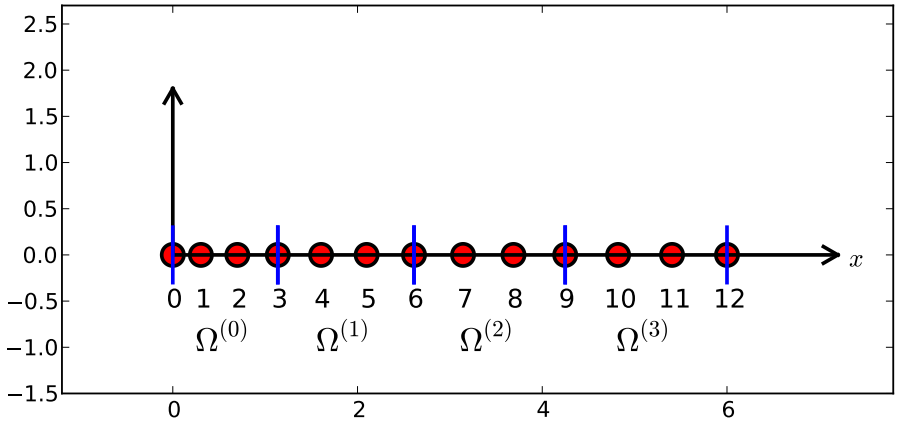



```
nodes = [0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0]
elements = [[0, 1, 2], [2, 3, 4], [4, 5, 6], [6, 7, 8]]
```

3.8 Some corresponding basis functions (P2 elements)

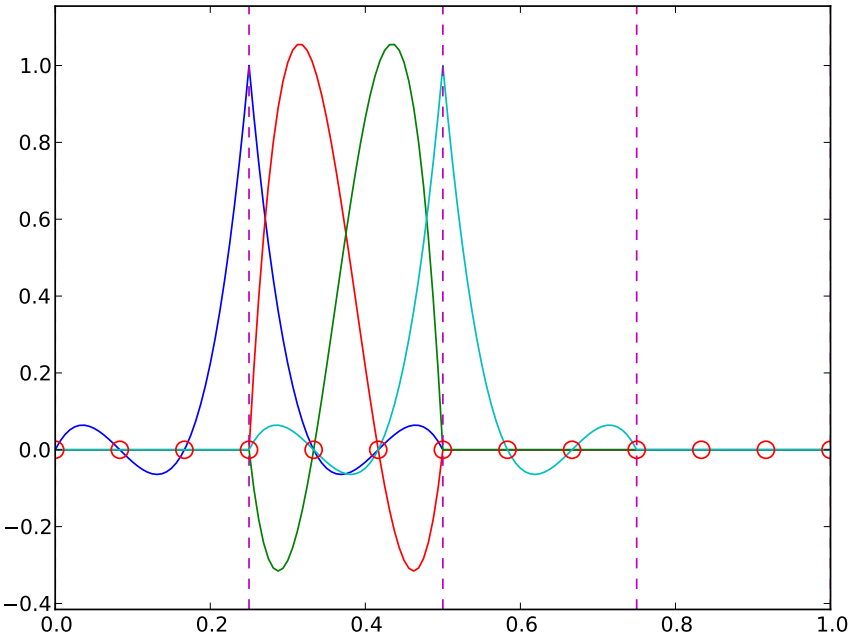


3.9 Examples on elements with four nodes per element (P3 elements)

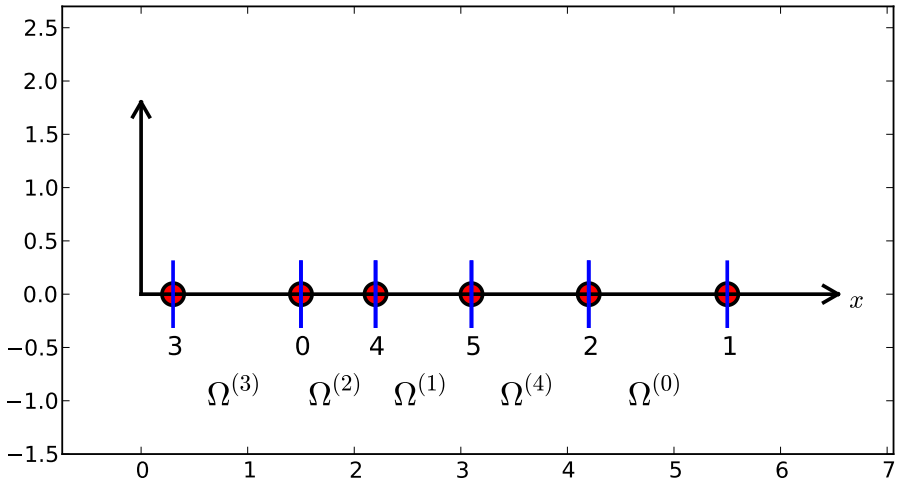



```
d = 3  # d+1 nodes per element
num_elements = 4
num_nodes = num_elements*d + 1
nodes = [i*0.5 for i in range(num_nodes)]
elements = [[i*d+j for j in range(d+1)] for i in range(num_elements)]
```

3.10 Some corresponding basis functions (P3 elements)



3.11 The numbering does not need to be regular from left to right




```
nodes = [1.5, 5.5, 4.2, 0.3, 2.2, 3.1]
elements = [[2, 1], [4, 5], [0, 4], [3, 0], [5, 2]]
```

3.12 Interpretation of the coefficients c_i

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

$$u(x_i) = \sum_{j \in I} c_j \varphi_j(x_i) = c_i \varphi_i(x_i) = c_i \tag{36}$$

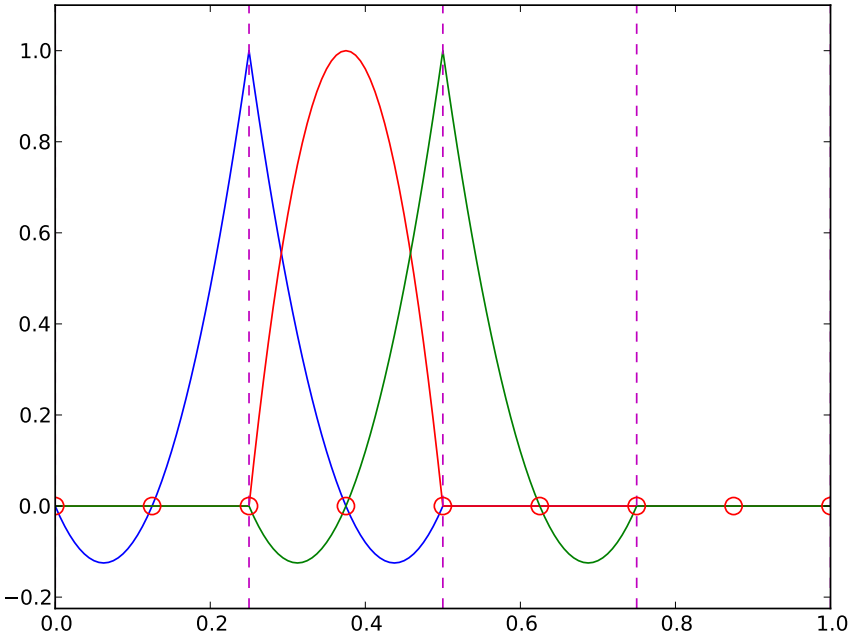
3.13 Properties of the basis functions

$\varphi_i(x)$ 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!).

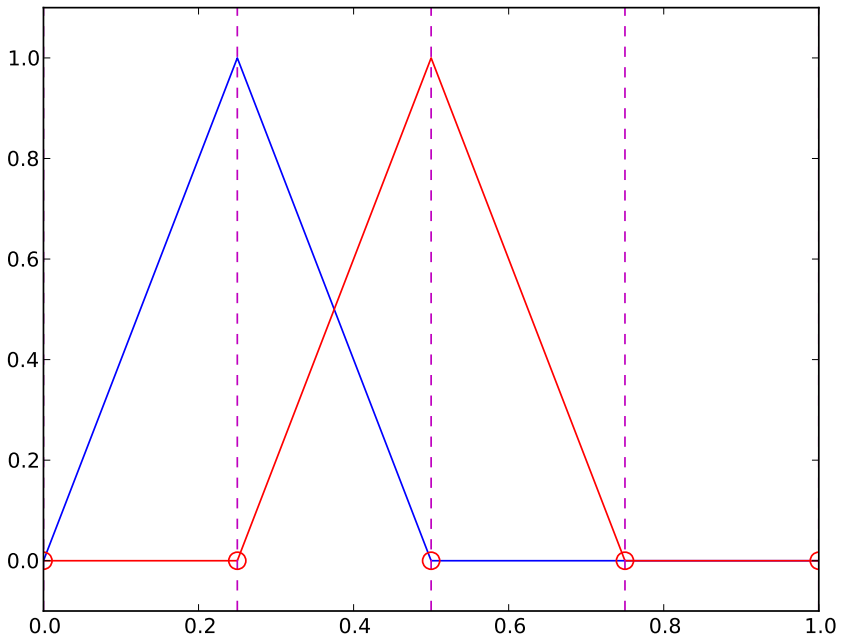
3.14 How to construct quadratic φ_i (P2 elements)



1. Associate Lagrange polynomials with the nodes in an element

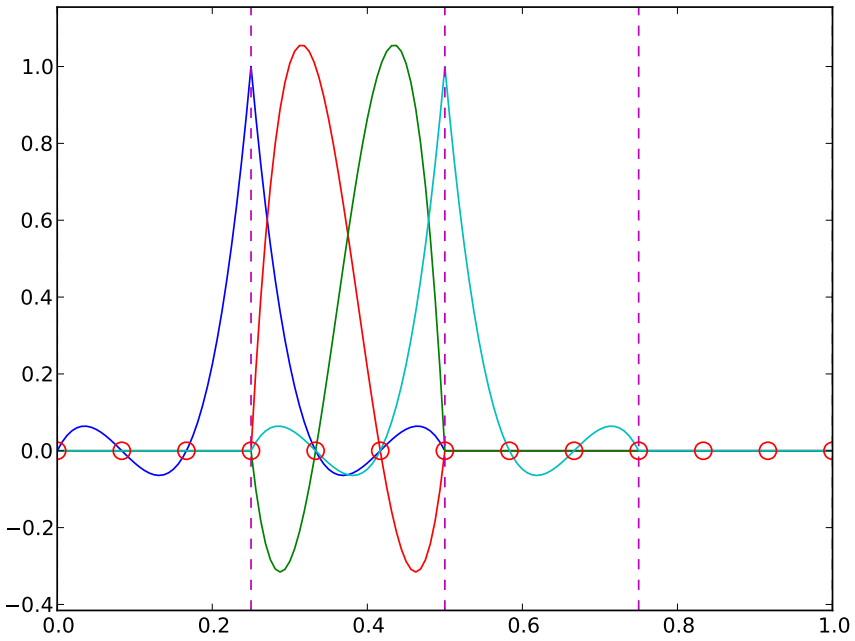
2. When the polynomial is 1 on the element boundary, combine it with the polynomial in the neighboring element

3.15 Example on linear φ_i (P1 elements)



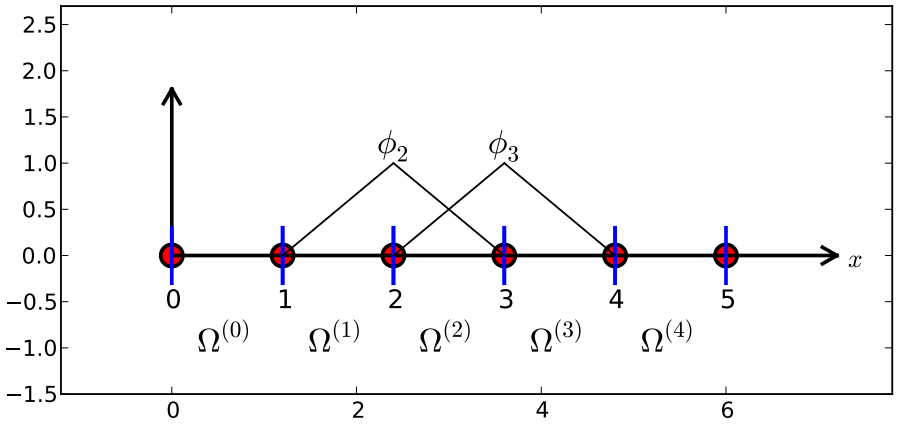
$$\varphi_i(x) = \begin{cases} 0, & x < x_{i-1}, \\ (x - x_{i-1})/h, & x_{i-1} \le x < x_i, \\ 1 - (x - x_i)/h, & x_i \le x < x_{i+1}, \\ 0, & x \ge x_{i+1} \end{cases} \tag{37}$$

3.16 Example on cubic φ_i (P3 elements)



4 Calculating the linear system for c_i

4.1 Computing a specific matrix entry (1)

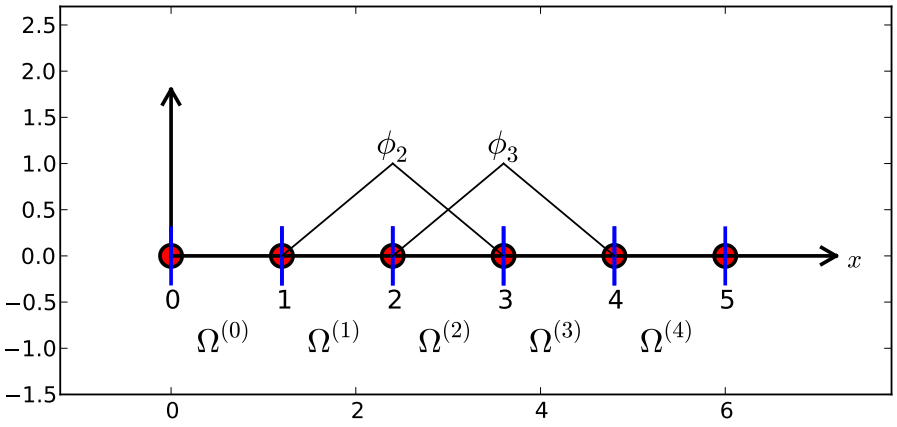


$A_{2,3} = \int_\Omega \varphi_2\varphi_3 dx$: $\varphi_2\varphi_3 \neq 0$ only over element 2. There,

$$\varphi_3(x) = (x - x_2)/h, \quad \varphi_2(x) = 1 - (x - x_2)/h$$

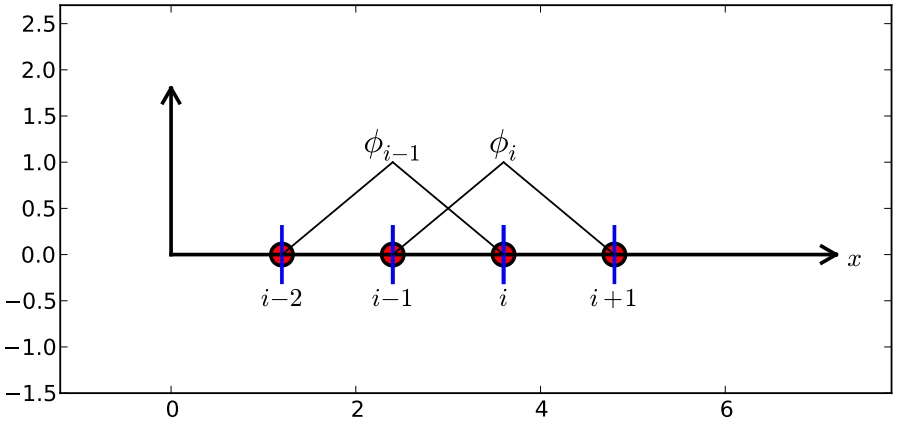
$$A_{2,3} = \int_\Omega \varphi_2 \varphi_3 \,\mathrm{d}x = \int_{x_2}^{x_3} \left(1 - \frac{x - x_2}{h}\right) \frac{x - x_2}{h} \,\mathrm{d}x = \frac{h}{6}$$

4.2 Computing a specific matrix entry (2)



$$A_{2,2} = \int_{x_1}^{x_2} \left(\frac{x - x_1}{h}\right)^2 \mathrm{d}x + \int_{x_2}^{x_3} \left(1 - \frac{x - x_2}{h}\right)^2 \mathrm{d}x = \frac{h}{3}$$

4.3 Calculating a general row in the matrix; figure



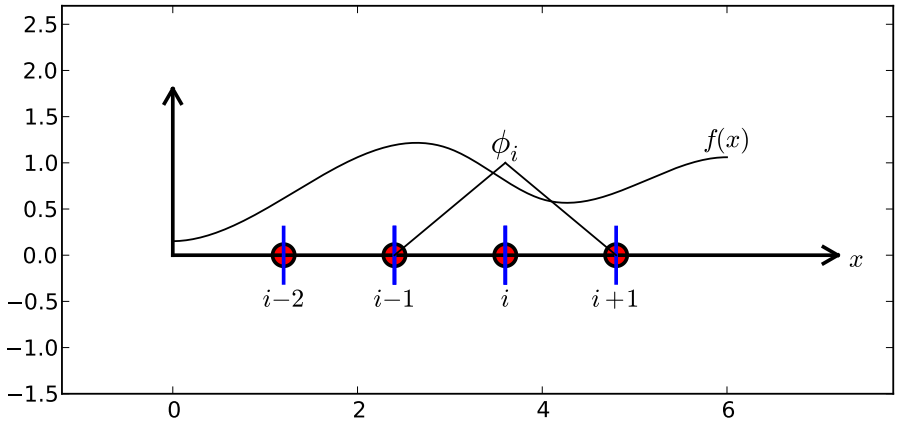
$$A_{i,i-1} = \int_\Omega \varphi_i \varphi_{i-1} \,\mathrm{d}x = ?$$

4.4 Calculating a general row in the matrix; details

$$
\begin{aligned}
A_{i,i-1} &= \int_\Omega \varphi_i \varphi_{i-1}\,\mathrm{d}x \\
&= \underbrace{\int_{x_{i-2}}^{x_{i-1}} \varphi_i \varphi_{i-1}\,\mathrm{d}x}_{\varphi_i = 0} + \int_{x_i}^{x_i} \varphi_i \varphi_{i-1}\,\mathrm{d}x + \underbrace{\int_{x_i}^{x_{i+1}} \varphi_i \varphi_{i-1}\,\mathrm{d}x}_{\varphi_{i-1}=0} \\
&= \int_{x_{i-1}}^{x_i} \underbrace{\frac{x - x_i}{h}}_{\varphi_i(x)} \underbrace{\left(1 - \frac{x - x_{i-1}}{h}\right)}_{\varphi_{i-1}(x)} \mathrm{d}x = \frac{h}{6}
\end{aligned}
$$

- $A_{i,i+1} = A_{i,i-1}$ due to symmetry
- $A_{i,i} = h/3$ (same calculation as for $A_{2,2}$)
- $A_{0,0} = A_{N,N} = h/3$ (only one element)

4.5 Calculation of the right-hand side



$$
b_i = \int_\Omega \varphi_i(x) f(x)\,\mathrm{d}x = \int_{x_{i-1}}^{x_i} \frac{x - x_{i-1}}{h} f(x)\,\mathrm{d}x + \int_{x_i}^{x_{i+1}} \left(1 - \frac{x - x_i}{h}\right) f(x)\,\mathrm{d}x \tag{38}
$$

Need a specific $f(x)$ to do more...

4.6 Specific example: two elements; linear system and solution

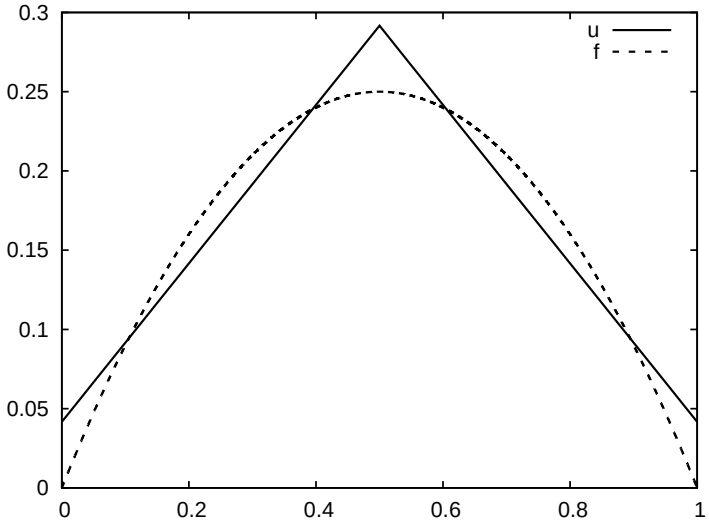
- $f(x) = x(1-x)$ on $\Omega = [0,1]$
- Two equal-sized elements $[0, 0.5]$ and $[0.5, 1]$

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

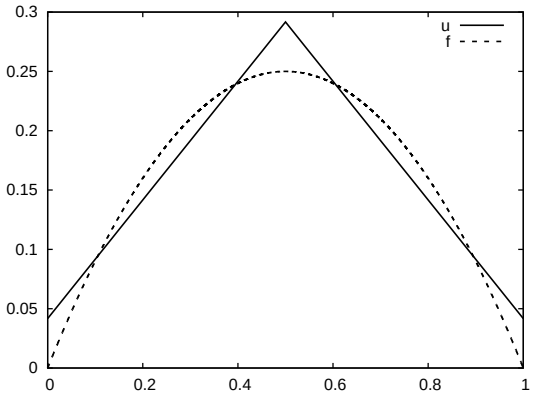
$$
c_0 = \frac{h^2}{6}, \quad c_1 = h - \frac{5}{6}h^2, \quad c_2 = 2h - \frac{23}{6}h^2
$$

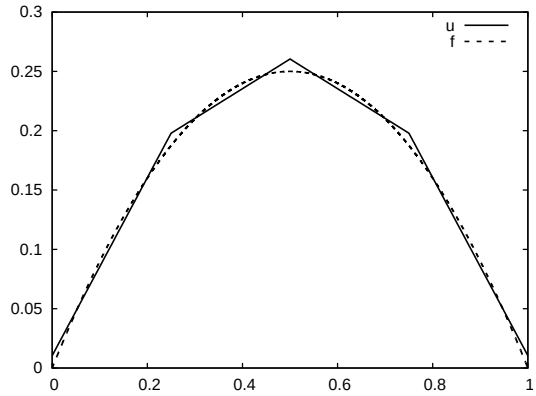
4.7 Specific example: two elements; plot

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



4.8 Specific example: what about four elements?





5 Assembly of elementwise computations

5.1 Split the integrals into elementwise integrals

$$A_{i,j} = \int_\Omega \varphi_i\varphi_j dx = \sum_e A^{(e)}_{i,j}, \quad A^{(e)}_{i,j} = \int_{\Omega^{(e)}} \varphi_i\varphi_j dx \tag{39}$$

Important:

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

5.2 The element matrix

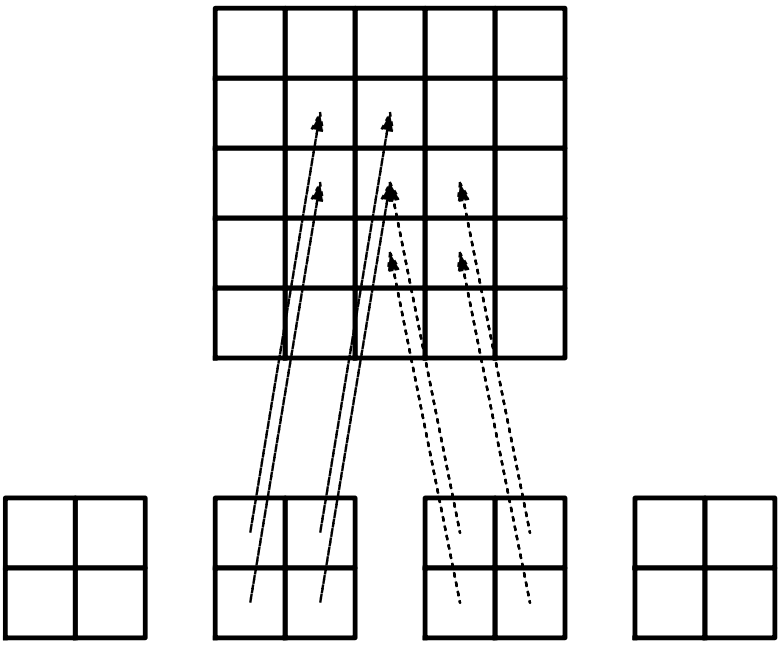
$$\tilde{A}^{(e)} = \{\tilde{A}^{(e)}_{r,s}\}, \quad r,s \in I_d = \{0,\ldots,d\}$$

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

- 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*)

$$A_{q(e,r),q(e,s)} := A_{q(e,r),q(e,s)} + \tilde{A}^{(e)}_{r,s}, \quad r,s \in I_d \tag{40}$$

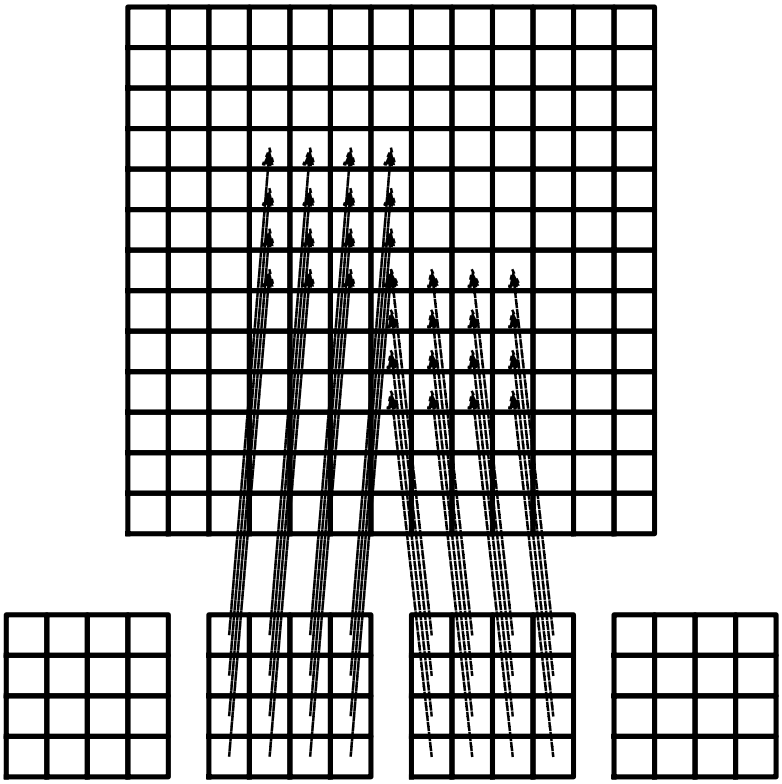
5.3 Illustration of the matrix assembly: regularly numbered P1 elements



Animation[2]

[2] `http://tinyurl.com/k3sdbuv/pub/mov-fem/fe_assembly.html`

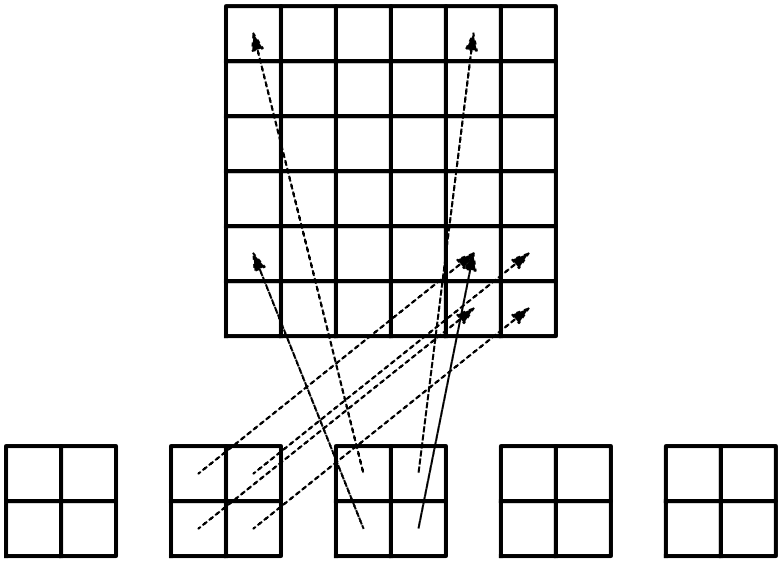
5.4 Illustration of the matrix assembly: regularly numbered P3 elements



Animation[3]

[3] http://tinyurl.com/k3sdbuv/pub/mov-fem/fe_assembly.html

5.5 Illustration of the matrix assembly: irregularly numbered P1 elements



Animation[4]

5.6 Assembly of the right-hand side

Split in elementwise contributions:

$$b_i = \int_\Omega \varphi_i \varphi_j dx = \sum_e b_i^{(e)}, \quad b_i^{(e)} = \int_{\Omega^{(e)}} f(x)\varphi_i(x)dx \tag{41}$$

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*

$$\tilde{b}_r^{(e)} = \{\tilde{b}_r^{(e)}\}, \quad r \in I_d$$

Assembly:

$$b_{q(e,r)} := b_{q(e,r)} + \tilde{b}_r^{(e)}, \quad r, s \in I_d \tag{42}$$

6 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.

[4] `http://tinyurl.com/k3sdbuv/pub/mov-fem/fe_assembly.html`

6.1 Affine mapping

$$x = \frac{1}{2}(x_L + x_R) + \frac{1}{2}(x_R - x_L)X \tag{43}$$

or rewritten as

$$x = x_m + \frac{1}{2}hX, \qquad x_m = (x_L + x_R)/2 \tag{44}$$

6.2 Integral transformation

Integrating on the reference element is a matter of just changing the integration variable from x to X. Introduce local basis function

$$\tilde{\varphi}_r(X) = \varphi_{q(e,r)}(x(X)) \tag{45}$$

The integral transformation reads

$$\tilde{A}^{(e)}_{r,s} = \int_{\Omega^{(e)}} \varphi_{q(e,r)}(x)\varphi_{q(e,s)}(x)dx = \int_{-1}^{1} \tilde{\varphi}_r(X)\tilde{\varphi}_s(X)\frac{dx}{dX}dX \tag{46}$$

Introduce the notation $\det J = dx/dX = h/2$ (2D/3D must use $\det J$)

$$\tilde{A}^{(e)}_{r,s} = \int_{-1}^{1} \tilde{\varphi}_r(X)\tilde{\varphi}_s(X)\det J\, dX \tag{47}$$

$$\tilde{b}^{(e)}_{r} = \int_{\Omega^{(e)}} f(x)\varphi_{q(e,r)}(x)dx = \int_{-1}^{1} f(x(X))\tilde{\varphi}_r(X)\det J\, dX \tag{48}$$

6.3 Advantages of the reference element

- 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)
- All geometric information (length and location) is "factored out" in the mapping and $\det J$

6.4 Standardized basis functions for P1 elements

$$\tilde{\varphi}_0(X) = \frac{1}{2}(1 - X) \tag{49}$$

$$\tilde{\varphi}_1(X) = \frac{1}{2}(1 + X) \tag{50}$$

6.5 Standardized basis functions for P2 elements

P2 elements:

$$\tilde{\varphi}_0(X) = \frac{1}{2}(X - 1)X \tag{51}$$

$$\tilde{\varphi}_1(X) = 1 - X^2 \tag{52}$$

$$\tilde{\varphi}_2(X) = \frac{1}{2}(X + 1)X \tag{53}$$

Easy to generalize to arbitrary order!

6.6 Integration over a reference element; element matrix

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

$$\begin{aligned}
\tilde{A}^{(e)}_{0,0} &= \int_{-1}^{1} \tilde{\varphi}_0(X)\tilde{\varphi}_0(X)\frac{h}{2}dX \\
&= \int_{-1}^{1} \frac{1}{2}(1-X)\frac{1}{2}(1-X)\frac{h}{2}dX = \frac{h}{8}\int_{-1}^{1}(1-X)^2dX = \frac{h}{3}, \qquad (54)\\
\tilde{A}^{(e)}_{1,0} &= \int_{-1}^{1} \tilde{\varphi}_1(X)\tilde{\varphi}_0(X)\frac{h}{2}dX \\
&= \int_{-1}^{1} \frac{1}{2}(1+X)\frac{1}{2}(1-X)\frac{h}{2}dX = \frac{h}{8}\int_{-1}^{1}(1-X^2)dX = \frac{h}{6}, \qquad (55)\\
\tilde{A}^{(e)}_{0,1} &= \tilde{A}^{(e)}_{1,0}, \qquad (56)\\
\tilde{A}^{(e)}_{1,1} &= \int_{-1}^{1} \tilde{\varphi}_1(X)\tilde{\varphi}_1(X)\frac{h}{2}dX \\
&= \int_{-1}^{1} \frac{1}{2}(1+X)\frac{1}{2}(1+X)\frac{h}{2}dX = \frac{h}{8}\int_{-1}^{1}(1+X)^2dX = \frac{h}{3} \qquad (57)
\end{aligned}$$

6.7 Integration over a reference element; element vector

$$\begin{aligned}
\tilde{b}^{(e)}_0 &= \int_{-1}^{1} f(x(X))\tilde{\varphi}_0(X)\frac{h}{2}dX \\
&= \int_{-1}^{1} (x_m + \frac{1}{2}hX)(1-(x_m+\frac{1}{2}hX))\frac{1}{2}(1-X)\frac{h}{2}dX \\
&= -\frac{1}{24}h^3 + \frac{1}{6}h^2x_m - \frac{1}{12}h^2 - \frac{1}{2}hx_m^2 + \frac{1}{2}hx_m \qquad (58)\\
\tilde{b}^{(e)}_1 &= \int_{-1}^{1} f(x(X))\tilde{\varphi}_1(X)\frac{h}{2}dX \\
&= \int_{-1}^{1} (x_m + \frac{1}{2}hX)(1-(x_m+\frac{1}{2}hX))\frac{1}{2}(1+X)\frac{h}{2}dX \\
&= -\frac{1}{24}h^3 - \frac{1}{6}h^2x_m + \frac{1}{12}h^2 - \frac{1}{2}hx_m^2 + \frac{1}{2}hx_m \qquad (59)
\end{aligned}$$

x_m: element midpoint.

6.8 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}
```

7 Implementation

- Coming functions appear in `fe_approx1D.py`[5]
- Functions can operate in symbolic or numeric mode
- The code documents all steps in finite element calculations!

7.1 Compute finite element basis functions

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

```
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

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

7.2 Compute the 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):
```

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

```
            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
```

7.3 Example on symbolic and numeric element matrix

```
>>> 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]
```

7.4 Compute the 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 `f.subs('x', x)`: replace `x` by $x(X)$ such that `f` contains `X`

7.5 Fallback on numerical integration if symbolic integration fails

- Element matrix: only polynomials and `sympy` always succeeds
- Element vector: $\int f\tilde{\varphi}\,\mathrm{d}x$ can fail (`sympy` then returns an `Integral` object instead of a number)

```
def element_vector(f, phi, Omega_e, symbolic=True):
        ...
        I = sm.integrate(f*phi[r]*detJ, (X, -1, 1))  # try...
        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
        ...
```

7.6 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
```

7.7 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
```

Note: the symbolic computation of `A` and `b` and the symbolic solution can be very tedious.

7.8 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]
```

7.9 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!

7.10 General result: the coefficient matrix is sparse

- Sparse = most of the entries are zeros
- Below: P1 elements

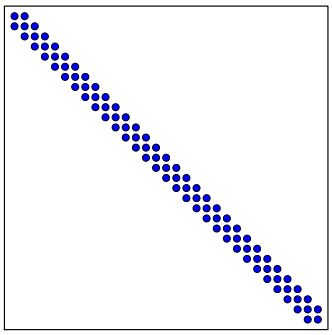
$$A = \frac{h}{6}\begin{pmatrix} 2 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 1 & 4 & 1 & \ddots & & & & & \vdots \\ 0 & 1 & 4 & 1 & \ddots & & & & \vdots \\ \vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & 0 & 1 & 4 & 1 & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & & \ddots & 1 & 4 & 1 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 & 2 \end{pmatrix} \tag{60}$$

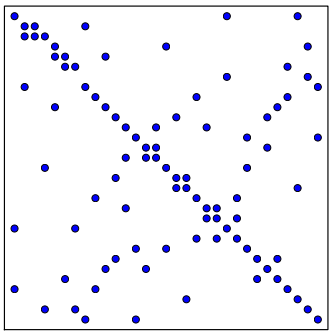
7.11 Exemplifying the sparsity for P2 elements

$$A = \frac{h}{30}\begin{pmatrix} 4 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 16 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & 8 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 16 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & 8 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 16 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & 8 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 16 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & 4 \end{pmatrix} \tag{61}$$

7.12 Matrix sparsity pattern for regular/random numbering of P1 elements

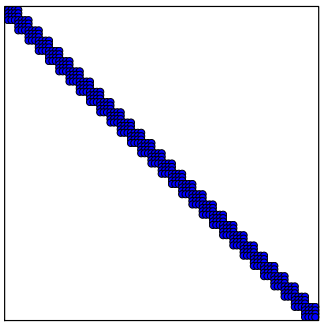
- Left: number nodes and elements from left to right
- Right: number nodes and elements arbitrarily

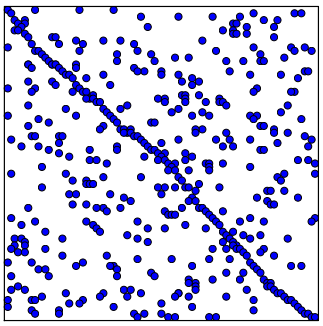




7.13 Matrix sparsity pattern for regular/random numbering of P3 elements

- Left: number nodes and elements from left to right
- Right: number nodes and elements arbitrarily





7.14 Sparse matrix storage and solution

The minimum storage requirements for the coefficient matrix $A_{i,j}$:

- P1 elements: only 3 nonzero entires per row
- P2 elements: only 5 nonzero entires per row
- P2 elements: only 7 nonzero entires per row
- It is important to utilize sparse storage and sparse solvers
- In Python: `scipy.sparse` package

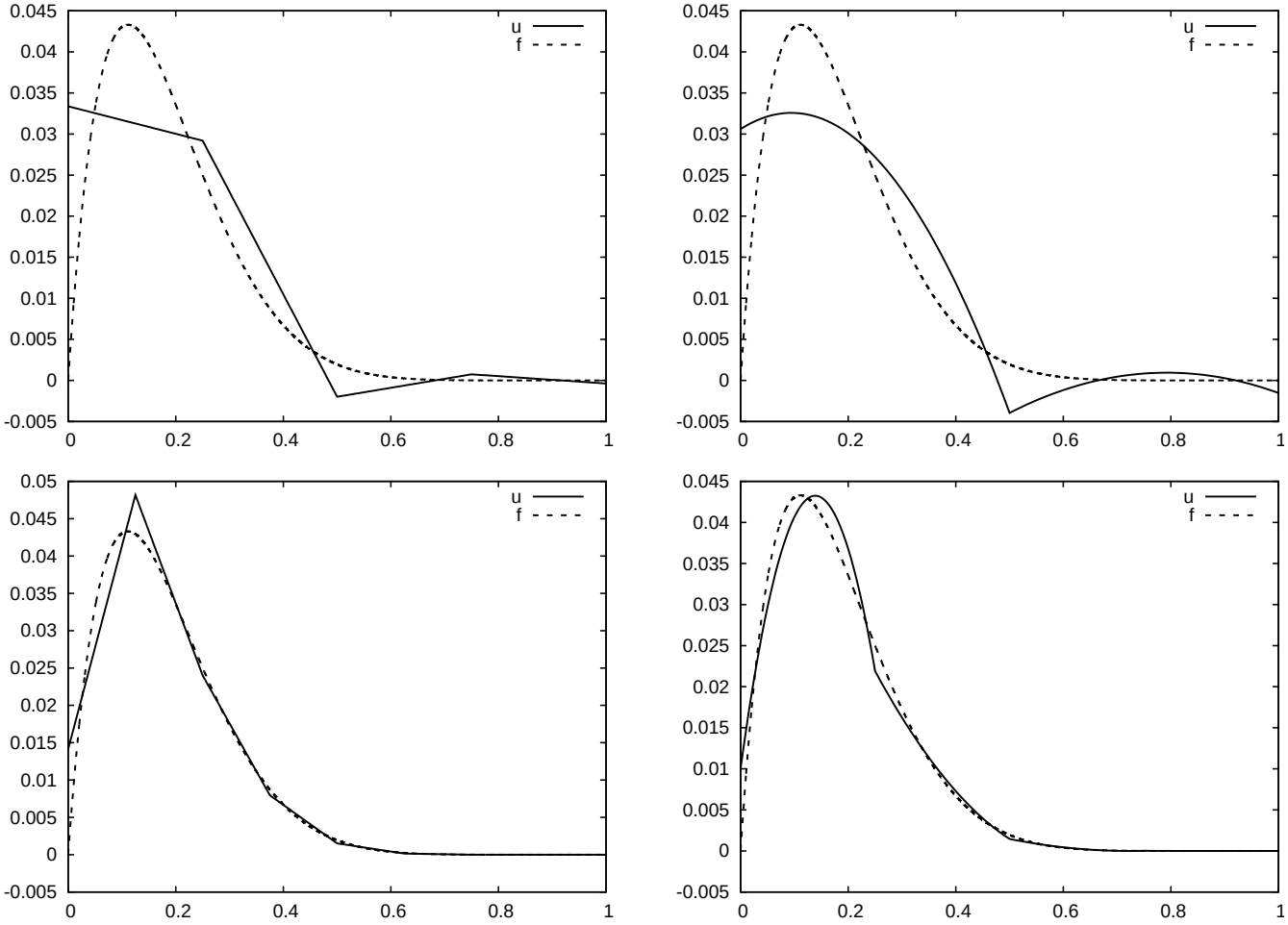
7.15 Approximate $f \sim x^9$ by various elements; code

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

```
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)
```

7.16 Approximate $f \sim x^9$ by various elements; plot



8 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?

8.1 Interpolation/collocation with finite elements

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

$$u(x_i) = f(x_i), \quad i \in I, \tag{62}$$

which translates to

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

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

$$c_i = f(x_i) \tag{63}$$

- 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

8.2 Differential equation models

Abstract differential equation:

$$\mathcal{L}(u) = 0, \quad x \in \Omega\,. \tag{64}$$

Examples:

$$\mathcal{L}(u) = \frac{d^2u}{dx^2} - f(x), \tag{65}$$

$$\mathcal{L}(u) = \frac{d}{dx}\left(a(x)\frac{du}{dx}\right) + f(x), \tag{66}$$

$$\mathcal{L}(u) = \frac{d}{dx}\left(a(u)\frac{du}{dx}\right) - \alpha u + f(x), \tag{67}$$

$$\mathcal{L}(u) = \frac{d}{dx}\left(a(u)\frac{du}{dx}\right) + f(u,x)\,. \tag{68}$$

$$\mathcal{B}_0(u) = 0,\ x = 0, \quad \mathcal{B}_1(u) = 0,\ x = L \tag{69}$$

There are three common choices of boundary conditions:

$$\mathcal{B}_i(u) = u - g, \qquad \text{Dirichlet condition}, \tag{70}$$

$$\mathcal{B}_i(u) = -a\frac{du}{dx} - g, \qquad \text{Neumann condition}, \tag{71}$$

$$\mathcal{B}_i(u) = -a\frac{du}{dx} - a(u-g), \qquad \text{Robin condition}\,. \tag{72}$$

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

$$\mathcal{L}(u_{\mathrm{e}}) = 0, \quad x \in \Omega, \tag{73}$$

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

8.3 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

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

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

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

$$R = \mathcal{L}(u) = \mathcal{L}(\sum_j c_j \psi_j), \tag{74}$$

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

The least squares method. Idea: minimize

$$\int_\Omega R^2 dx \tag{75}$$

With the inner product

$$(f, g) = \int_\Omega f(x) g(x) dx, \tag{76}$$

the least-squares method can be defined as

$$\min_{c_0, \ldots, c_N} E = (R, R)\,. \tag{77}$$

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

$$\int_\Omega 2R \frac{\partial R}{\partial c_i} dx = 0 \quad \Leftrightarrow \quad (R, \frac{\partial R}{\partial c_i}) = 0, \quad i \in I\,. \tag{78}$$

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

$$(R, v) = 0, \quad \forall v \in V\,. \tag{79}$$

Equivalent statement:

$$(R, \psi_i) = 0, \quad i \in I, \tag{80}$$

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 $W \neq V$:

$$(R, v) = 0, \quad \forall v \in W\,. \tag{81}$$

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

$$(R, w_i) = 0, \quad i \in I\,. \tag{82}$$

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.

$$R(x_i; c_0, \ldots, c_N) = 0, \quad i \in I\,. \tag{83}$$

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

$$\int_\Omega f(x)\delta(x - x_i)dx = f(x_i), \quad x_i \in \Omega\,. \tag{84}$$

8.4 Examples on using the principles

The model problem.

$$-u''(x) = f(x), \quad x \in \Omega = [0, L], \quad u(0) = 0,\ u(L) = 0\,. \tag{85}$$

Basis functions.

$$\psi_i(x) = \sin\left((i+1)\pi\frac{x}{L}\right), \quad i \in I\,. \tag{86}$$

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:

$$\int_0^L \sin\left((i+1)\pi\frac{x}{L}\right)\sin\left((j+1)\pi\frac{x}{L}\right)\,dx = \begin{cases} \frac{1}{2}L & i = j \\ 0, & i \neq j \end{cases} \tag{87}$$

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

The residual.

$$\begin{aligned} R(x; c_0, \ldots, c_N) &= u''(x) + f(x), \\ &= \frac{d^2}{dx^2}\left(\sum_{j\in I} c_j\psi_j(x)\right) + f(x), \\ &= -\sum_{j\in I} c_j\psi_j''(x) + f(x)\,. \end{aligned} \tag{88}$$

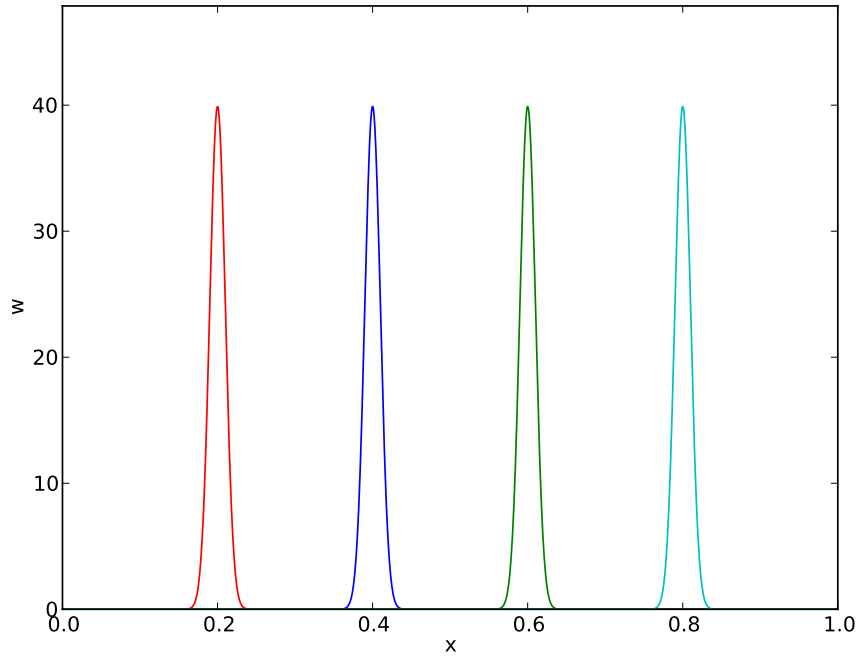


Figure 1: Approximation of delta functions by narrow Gaussian functions.

The least squares method.

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

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

$$\frac{\partial R}{\partial c_i} = \frac{\partial}{\partial c_i}\left(\sum_{j\in I} c_j\psi_j''(x) + f(x)\right) = \psi_i''(x)\,. \tag{89}$$

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

$$(\sum_j c_j\psi_j'' + f, \psi_i'') = 0, \quad i \in I, \tag{90}$$

which can be rearranged as

$$\sum_{j\in I}(\psi_i'', \psi_j'')c_j = -(f, \psi_i''), \quad i \in I\,. \tag{91}$$

This is nothing but a linear system

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

with

$$\begin{aligned} A_{i,j} &= (\psi_i'', \psi_j'') \\ &= \pi^4(i+1)^2(j+1)^2L^{-4}\int_0^L \sin\left((i+1)\pi\frac{x}{L}\right)\sin\left((j+1)\pi\frac{x}{L}\right)\,dx \\ &= \begin{cases} \frac{1}{2}L^{-3}\pi^4(i+1)^4 & i=j \\ 0, & i\neq j \end{cases} \end{aligned} \tag{92}$$

$$b_i = -(f,\psi_i'') = (i+1)^2\pi^2L^{-2}\int_0^L f(x)\sin\left((i+1)\pi\frac{x}{L}\right)\,dx \tag{93}$$

Since the coefficient matrix is diagonal we can easily solve for

$$c_i = \frac{2L}{\pi^2(i+1)^2}\int_0^L f(x)\sin\left((i+1)\pi\frac{x}{L}\right)\,dx\,. \tag{94}$$

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

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

The solution becomes:

$$u(x) = \sum_{k=0}^{N/2}\frac{8L^2}{\pi^3(2k+1)^3}\sin\left((2k+1)\pi\frac{x}{L}\right)\,. \tag{95}$$

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

$$(u'', v) = -(f, v), \quad \forall v \in V\,. \tag{96}$$

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

$$(\sum_{j\in I} c_j\psi_j'', \psi_i) = -(f, \psi_i), \quad i \in I\,. \tag{97}$$

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:

$$-\sum_{j\in I} c_j \psi_j''(x_i) = f(x_i), \quad i \in I\,. \tag{98}$$

This is a linear system with entries

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

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: $0.047L^2$

8.5 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:

$$\begin{aligned}\int_0^L u''(x)v(x)dx &= -\int_0^L u'(x)v'(x)dx + [vu']_0^L \\ &= -\int_0^L u'(x)v'(x)dx + u'(L)v(L) - u'(0)v(0)\,.\end{aligned} \tag{99}$$

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)

8.6 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

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

$$u(x) = \frac{x}{L}D + \sum_{j\in I} c_j \psi_j(x), \tag{100}$$

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

8.7 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:

$$a(\alpha_1 u_1 + \alpha_2 u_2, v) = \alpha_1 a(u_1, v) + \alpha_2 a(u_2, v),$$
$$a(u, \alpha_1 v_1 + \alpha_2 v_2) = \alpha_1 a(u, v_1) + \alpha_2 a(u, v_2)\,.$$

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

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

with

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

8.8 More examples on variational formulations

Variable coefficient. Consider the problem

$$-\frac{d}{dx}\left(a(x)\frac{du}{dx}\right) = f(x), \quad x \in \Omega = [0,L],\ u(0) = C,\ u(L) = D\,. \tag{101}$$

Two new features:

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

A boundary function handles nonzero Dirichlet conditions:

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

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:

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

Must have $v = 0$ where we have Dirichlet conditions: boundary terms vanish.

The final variational formulation:

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

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 $v = \psi_i$:

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

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:

$$-u''(x) + bu'(x) = f(x), \quad x \in \Omega = [0, L],\ u(0) = C,\ u'(L) = E\,. \tag{102}$$

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 $x = L$)

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

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 $v = \psi_i$,

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

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

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

8.9 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: $\psi_i(x) = (1 - x)^{i+1}$, $i \in I$
- $B(x) = Dx$

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 $N = 1$:

$$\begin{pmatrix} 1 & 1 \\ 1 & 4/3 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} -C + D + 1 \\ 2/3 - C + D \end{pmatrix}$$

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

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

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

$$-(a(u)u')' = f(u), \quad x \in [0, L],\ u(0) = 0,\ u'(L) = E\,. \tag{103}$$

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.

8.10 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

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

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

9 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:

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

9.1 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:

$$\varphi_i'(x) = \begin{cases} 0, & x < x_{i-1}, \\ h^{-1}, & x_{i-1} \leq x < x_i, \\ -h^{-1}, & x_i \leq x < x_{i+1}, \\ 0, & x \geq x_{i+1} \end{cases} \tag{104}$$

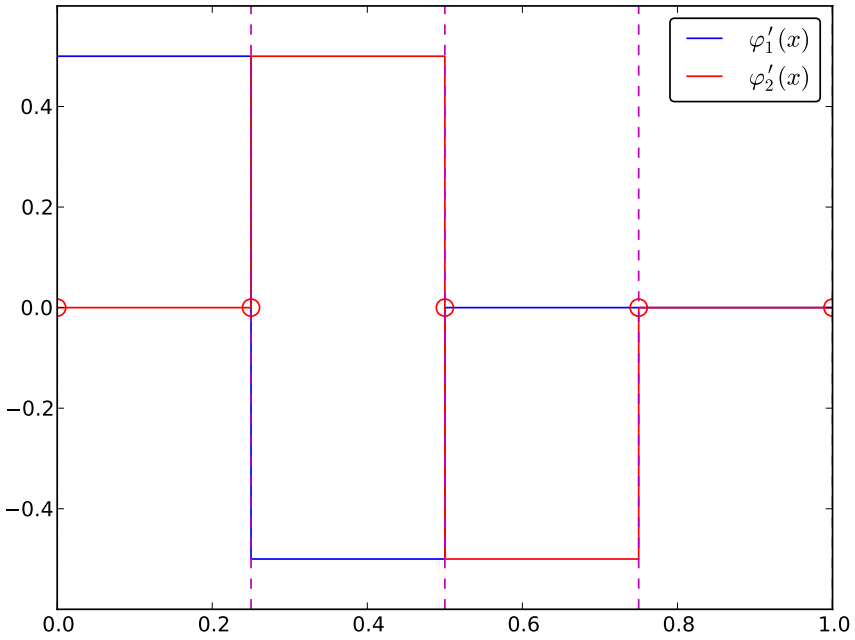


Figure 2: Illustration of the derivative of piecewise linear basis functions associated with nodes in cell 1.

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

$$\frac{1}{h}\begin{pmatrix} 2 & -1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ -1 & 2 & -1 & \ddots & & & & & \vdots \\ 0 & -1 & 2 & -1 & \ddots & & & & \vdots \\ \vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & 0 & -1 & 2 & -1 & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & -1 & 2 \end{pmatrix}\begin{pmatrix} c_1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_{N-1} \end{pmatrix} = \begin{pmatrix} 2h \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 2h \end{pmatrix} \tag{105}$$

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

$$-\frac{1}{h}u_{i-1}+\frac{2}{h}u_i-\frac{1}{h}u_{i+1}=2h\,. \tag{106}$$

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

$$-\frac{u_{i-1}-2u_i+u_{i+1}}{h^2}=2, \tag{107}$$

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

9.2 Elementwise computations

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

Present element matrix:

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

$\tilde{\varphi}_r(X)$ 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,

$$\frac{d\tilde{\varphi}_r}{dx}=\frac{d\tilde{\varphi}_r}{dX}\frac{dX}{dx}=\frac{2}{h}\frac{d\tilde{\varphi}_r}{dX}\,. \tag{108}$$

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

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

In matrix/vector notation:

$$\tilde{A}^{(e)} = \frac{1}{h}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \quad \tilde{b}^{(e)} = h\begin{pmatrix} 1 \\ 1 \end{pmatrix}. \tag{109}$$

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

10 Boundary conditions: specified value

10.1 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

$$B(x) = \sum_{j\in D} U_j\varphi_j(x), \tag{110}$$

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

In 1D

$$B(x) = U_0\varphi_0(x) + U_N\varphi_N(x)\,. \tag{111}$$

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

$$u(x) = U_0\varphi_0(x) + U_N\varphi_N(x) + \sum_{j=1}^{N-1} c_j\varphi_j(x)\,. \tag{112}$$

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

10.2 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 ($u = \sum_{j\in I} c_j\varphi_j$):

$$\frac{1}{h}\begin{pmatrix} 1 & -1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ -1 & 2 & -1 & \ddots & & & & & \vdots \\ 0 & -1 & 2 & -1 & \ddots & & & & \vdots \\ \vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & 0 & -1 & 2 & -1 & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & -1 & 1 \end{pmatrix}\begin{pmatrix} c_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_N \end{pmatrix} = \begin{pmatrix} h \\ 2h \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 2h \\ h \end{pmatrix} \tag{113}$$

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 $c_N = D$

$$\frac{1}{h}\begin{pmatrix} 1 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ -1 & 2 & -1 & \ddots & & & & & \vdots \\ 0 & -1 & 2 & -1 & \ddots & & & & \vdots \\ \vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & 0 & -1 & 2 & -1 & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} c_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_N \end{pmatrix} = \begin{pmatrix} 0 \\ 2h \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 2h \\ D \end{pmatrix} \tag{114}$$

10.3 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 $b_i = K$

$$\frac{1}{h}\begin{pmatrix} 1 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 2 & -1 & \ddots & & & & & \vdots \\ 0 & -1 & 2 & -1 & \ddots & & & & \vdots \\ \vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & 0 & -1 & 2 & -1 & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_N \end{pmatrix} = \begin{pmatrix} 0 \\ 2h \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 2h + D/h \\ D \end{pmatrix} \tag{115}$$

10.4 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:

$$\tilde{A}^{(N-1)} = A = \frac{1}{h}\begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}, \quad \tilde{b}^{(N-1)} = \begin{pmatrix} h \\ D \end{pmatrix}. \tag{116}$$

Or symmetric modification:

$$\tilde{A}^{(N-1)} = A = \frac{1}{h}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \tilde{b}^{(N-1)} = \begin{pmatrix} h + D/h \\ D \end{pmatrix}. \tag{117}$$

11 Boundary conditions: specified derivative

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

11.1 The variational formulation

Start with the Galerkin method:

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

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 $u'(0) = C$

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

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

$$\sum_{j=0}^{N-1}\left(\int_0^L \varphi_i'(x)\varphi_j'(x)dx\right)c_j = \int_0^L \left(f(x)\varphi_i(x) - D\varphi_N'(x)\varphi_i(x)\right)dx - C\varphi_i(0), \tag{118}$$

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:

$$\tilde{A}^{(0)} = A = \frac{1}{h}\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}, \quad \tilde{b}^{(0)} = \begin{pmatrix} h - C \\ h \end{pmatrix}. \tag{119}$$

12 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>
```

13 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

$$-\int_\Omega \nabla\cdot(a(\boldsymbol{x})\nabla u)v\,\mathrm{d}x = \int_\Omega a(\boldsymbol{x})\nabla u\cdot\nabla v\,\mathrm{d}x - \int_{\partial\Omega} a\frac{\partial u}{\partial n}v\,\mathrm{d}s, \tag{120}$$

- $\int_\Omega()\,\mathrm{d}x$: 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:

$$\boldsymbol{v}\cdot\nabla u + \alpha u = \nabla\cdot(a\nabla u) + f, \quad \boldsymbol{x}\in\Omega, \tag{121}$$

$$u = u_0, \quad \boldsymbol{x}\in\partial\Omega_D, \tag{122}$$

$$-a\frac{\partial u}{\partial n} = g, \quad \boldsymbol{x}\in\partial\Omega_N\,. \tag{123}$$

- 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

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

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

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

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

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

We consider an integral of the type

$$\int_{\Omega^{(e)}} a(\boldsymbol{x})\nabla\varphi_i \cdot \nabla\varphi_j \,\mathrm{d}x \tag{124}$$

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

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

Integral transformation from physical to reference coordinates:

$$\int_{\Omega}^{(e)} a(\boldsymbol{x})\nabla_{\boldsymbol{x}}\varphi_i \cdot \nabla_{\boldsymbol{x}}\varphi_j \,\mathrm{d}x \int_{\tilde{\Omega}^r} a(\boldsymbol{x}(\boldsymbol{X}))(J^{-1} \cdot \nabla_{\boldsymbol{X}}\tilde{\varphi}_r) \cdot (J^{-1} \cdot \nabla\tilde{\varphi}_s) \det J \,\mathrm{d}X \tag{125}$$

14 Systems of differential equations

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

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

14.1 Variational forms

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

$$\int_\Omega \mathcal{L}^{(0)}(u^{(0)}, \ldots, u^{(m)}) v^{(0)} \,\mathrm{d}x = 0, \tag{126}$$

$$\vdots \tag{127}$$

$$\int_\Omega \mathcal{L}^{(m)}(u^{(0)}, \ldots, u^{(m)}) v^{(m)} \,\mathrm{d}x = 0\,. \tag{128}$$

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

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

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

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 $\boldsymbol{v}$:

$$\int_\Omega \boldsymbol{\mathcal{L}}(\boldsymbol{u}) \cdot \boldsymbol{v} = 0 \quad \forall \boldsymbol{v} \in \boldsymbol{V}\,. \tag{129}$$

- 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 (126)
- E.g.: $\boldsymbol{v} = (0, \ldots, 0, v^{(m)}$ recovers (128)

14.2 A worked example

$$\mu\nabla^2 w = -\beta, \tag{130}$$
$$\kappa\nabla^2 T = -\mu||\nabla w||^2 \quad (= \mu\nabla w \cdot \nabla w)\,. \tag{131}$$

- 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)

14.3 Identical function spaces for the unknowns

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

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

$$w = \sum_{j=0}^{N} c_j^{(w)}\varphi_j, \quad T = T_0 + \sum_{j=0}^{N} c_j^{(T)}\varphi_j\,. \tag{132}$$

Variational form of each individual PDE. Inserting (132) in the PDEs, results in the residuals

$$R_w = \mu\nabla^2 w + \beta, \tag{133}$$
$$R_T = \kappa\nabla^2 T + \mu||\nabla w||^2\,. \tag{134}$$

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!):

$$\int_\Omega \mu\nabla w \cdot \nabla v\,\mathrm{d}x = \int_\Omega \beta v\,\mathrm{d}x \quad \forall v \in V, \tag{135}$$
$$\int_\Omega \kappa\nabla T \cdot \nabla v\,\mathrm{d}x = \int_\Omega \mu\nabla w \cdot \nabla w\, v\,\mathrm{d}x \quad \forall v \in V\,. \tag{136}$$

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

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

$$\int_\Omega (\mu \nabla w \cdot \nabla v_0 + \kappa \nabla T \cdot \nabla v_1) \, \mathrm{d}x = \int_\Omega (\beta v_0 + \mu \nabla w \cdot \nabla w \, v_1) \, \mathrm{d}x, \quad \forall \boldsymbol{v} \in \boldsymbol{V} \tag{137}$$

Choosing $v_0 = v$ and $v_1 = 0$ gives the variational form (135), while $v_0 = 0$ and $v_1 = v$ gives (136).

Alternative inner product notation:

$$\mu(\nabla w, \nabla v) = (\beta, v) \quad \forall v \in V, \tag{138}$$

$$\kappa(\nabla T, \nabla v) = \mu(\nabla w \cdot \nabla w, v) \quad \forall v \in V \, . \tag{139}$$

Decoupled linear systems.

$$\sum_{j=0}^{N} A_{i,j}^{(w)} c_j^{(w)} = b_i^{(w)}, \quad i = 0, \ldots, N, \tag{140}$$

$$\sum_{j=0}^{N} A_{i,j}^{(T)} c_j^{(T)} = b_i^{(T)}, \quad i = 0, \ldots, N, \tag{141}$$

$$A_{i,j}^{(w)} = \mu(\nabla \varphi_j, \nabla \varphi_i), \tag{142}$$

$$b_i^{(w)} = (\beta, \varphi_i), \tag{143}$$

$$A_{i,j}^{(T)} = \kappa(\nabla \varphi_j, \nabla \varphi_i), \tag{144}$$

$$b_i^{(T)} = (\mu \nabla w_- \cdot (\sum_k c_k^{(w)} \nabla \varphi_k), \varphi_i) \, . \tag{145}$$

Matrix-vector form (alternative notation):

$$\mu K c^{(w)} = b^{(w)}, \tag{146}$$

$$\kappa K c^{(T)} = b^{(T)}, \tag{147}$$

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)

$$\sum_{j=0}^{N} A_{i,j}^{(w,w)} c_j^{(w)} + \sum_{j=0}^{N} A_{i,j}^{(w,T)} c_j^{(T)} = b_i^{(w)}, \quad i = 0, \ldots, N, \tag{148}$$

$$\sum_{j=0}^{N} A_{i,j}^{(T,w)} c_j^{(w)} + \sum_{j=0}^{N} A_{i,j}^{(T,T)} c_j^{(T)} = b_i^{(T)}, \quad i = 0, \ldots, N, \tag{149}$$

$$A_{i,j}^{(w,w)} = \mu(\nabla\varphi_j, \varphi_i), \tag{150}$$

$$A_{i,j}^{(w,T)} = 0, \tag{151}$$

$$b_i^{(w)} = (\beta, \varphi_i), \tag{152}$$

$$A_{i,j}^{(w,T)} = \mu(\nabla w_- \cdot \nabla\varphi_j), \varphi_i), \tag{153}$$

$$A_{i,j}^{(T,T)} = \kappa(\nabla\varphi_j, \varphi_i), \tag{154}$$

$$b_i^{(T)} = 0\,. \tag{155}$$

Alternative notation:

$$\mu K c^{(w)} = b^{(w)}, \tag{156}$$

$$L c^{(w)} + \kappa K c^{(T)} = 0, \tag{157}$$

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

Corresponding block form:

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

14.4 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*

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

$$\int_\Omega \mu \nabla w \cdot \nabla v^{(w)} \,\mathrm{d}x = \int_\Omega \beta v^{(w)} \,\mathrm{d}x \quad \forall v^{(w)} \in V^{(w)}, \tag{158}$$
$$\int_\Omega \kappa \nabla T \cdot \nabla v^{(T)} \,\mathrm{d}x = \int_\Omega \mu \nabla w \cdot \nabla w \, v^{(T)} \,\mathrm{d}x \quad \forall v^{(T)} \in V^{(T)}. \tag{159}$$

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

$$\int_\Omega (\mu \nabla w \cdot \nabla v^{(w)} + \kappa \nabla T \cdot \nabla v^{(T)}) \,\mathrm{d}x = \int_\Omega (\beta v^{(w)} + \mu \nabla w \cdot \nabla w \, v^{(T)}) \,\mathrm{d}x, \tag{160}$$

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

Index