Numerical Simulation Methods in Geophysics, Part 6: FE in higher dimensions

1. MGPY+MGIN

thomas.guenther@geophysik.tu-freiberg.de

TU BERGAKADEMIE FREIBERG

TUBAF

Die Ressourcenuniversität.
Seit 1765.

Finite Elements - Recap

- weak formulation of PDF together with shape & test functions
- integration over modellling domain (proper choice)
- hat functions in 1D, localized elements

Difference of FE to FD

u is described on the whole space and approximates the solution, not the PDE!

The stiffness matrix in 1D

Matrix integrating gradients of base functions for neighbors with a

$$\mathbf{A}_{i,i+1} = -\frac{a_i}{\Delta x_i^2} \cdot \Delta x_i = -\frac{a_i}{\Delta x_i}$$

$$A_{i,i} = \int_\Omega a \boldsymbol{\nabla} v_i \cdot \boldsymbol{\nabla} v_i \mathrm{d}\Omega = -A_{i,i+1} - A_{i+1,i}$$

$\Rightarrow$ matrix-vector equation $\mathbf{A}\mathbf{u} = \mathbf{b}$ with bending&shear stiffness in $\mathbf{A}$

Right-hand side vector

The right-hand-side vector $b = \int v_i f \mathrm{d}\Gamma$ also scales with Δx

e.g. $f = \boldsymbol{\nabla} \cdot \mathbf{j}_s \Rightarrow b = \int v_i \boldsymbol{\nabla} \cdot \mathbf{j}_s \mathrm{d}\Omega = \int_\Gamma v_i \mathbf{j}_S \cdot \mathbf{n}$

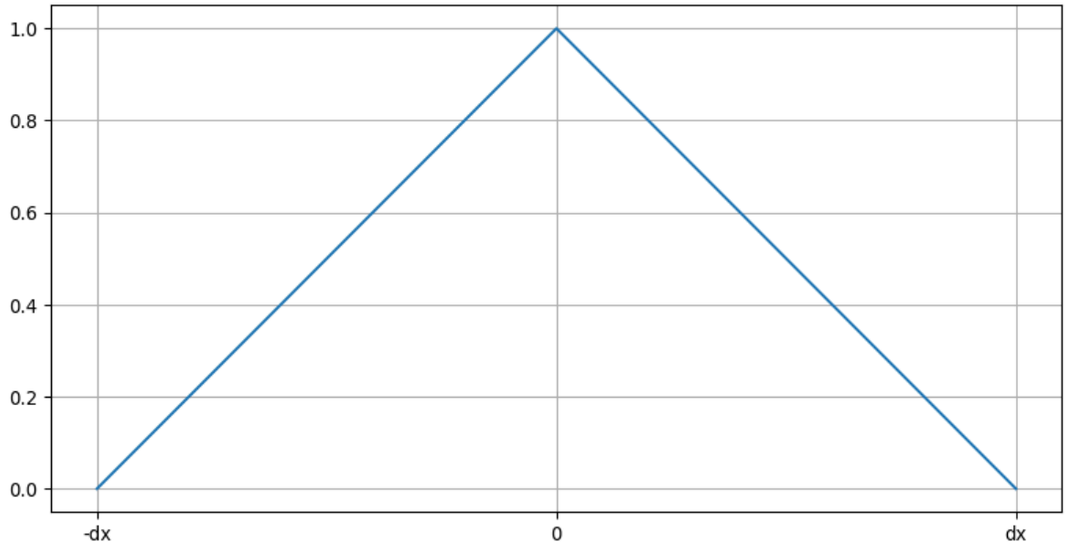
RHS = integrated source function (includes Δx)

(both $\mathbf{A}$ and $\mathbf{b}$ identical to FD for Δx=1)

Difference of FE to FD

Any source function $f(x)$ can be integrated on the whole space!

Right-hand side vector



$$b_i = \int v_i f dx$$

Area: $\Delta x_{i-1}/2 + \Delta x_i/2$

$$b_i = \int_{-\Delta x}^{0} \frac{x + \Delta x}{\Delta x} + \int_{0}^{\Delta x} 1 - \frac{x}{\Delta x} = \Delta x_{i-1}/2 + \Delta x_i/2$$

Boundary conditions

second term

$$-\int_{\Gamma} a v_j \boldsymbol{\nabla} v_i \cdot \mathbf{n} \mathrm{d}\Gamma$$

reads in 1D as

$$\left[a v_i v_j'\right]_{x_0}^{x_N} = a_{N-1} u_N v_N' - a_0 u_0 v_0'$$

$\Rightarrow$ Homogeneous Neumann BC ($v_0' = 0$) are automatically implemented

Higher order

Quadratic elements

$$u(\xi) = c_1 + c_2\xi + c_3\xi^2$$

nodes at x_0, $x_{1/2}$, x_1

$$u_i = u(0) = c_1, u_1 = c_1 + c_2 + c_3$$

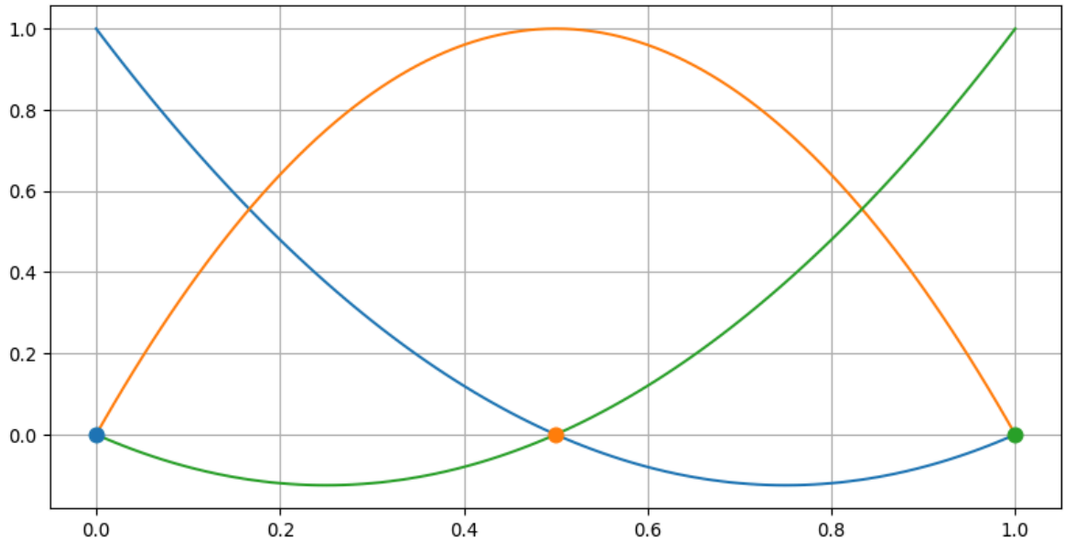
$$u_{1/2} = c_1 + c_2/2 + c_3/4$$

$$u(\xi) = u_0(1 - 3\xi + 2\xi^2) + u_{1/2}(4\xi - 4\xi^2) + u_1(-\xi + 2\xi^2)$$

Quadratic elements

$$u(\xi) = u_0(1 - 3\xi + 2\xi^2) + u_{1/2}(4\xi - 4\xi^2) + u_1(-\xi + 2\xi^2)$$

```
 1 import numpy as np
 2 import matplotlib.pyplot as plt
 3 x=np.linspace(0, 1, 101)
 4
 5 # Plot velocity distribution.
 6 plt.plot(x, 1-3*x+2*x**2)
 7 plt.plot(x, 4*x-4*x**2)
 8 plt.plot(x, -x+2*x**2)
 9 plt.plot(0, 0, "o", color="C0", ms=8)
10 plt.plot(0.5, 0, "o", color="C1", ms=8)
11 plt.plot(1, 0, "o", color="C2", ms=8)
12 plt.grid()
```



Elements of n-th order

```python
n = 4
A = np.zeros([n, n])
xis = np.arange(n)/(n-1)
for i, xi in enumerate(xis):
    A[i, :] = xis[i]**np.arange(n)

print(A)
print(np.linalg.inv(A.T))
```

```
[[1.         0.         0.         0.        ]
 [1.         0.33333333 0.11111111 0.03703704]
 [1.         0.66666667 0.44444444 0.2962963 ]
 [1.         1.         1.         1.        ]
]]
[[  1.   -5.5   9.   -4.5]
 [  0.    9.  -22.5  13.5]
 [  0.   -4.5  18.  -13.5]
 [  0.    1.   -4.5   4.5]]
```

```python
n = 5
A = np.zeros([n, n])
xis = np.arange(n)/(n-1)
for i, xi in enumerate(xis):
    A[i, :] = xis[i]**np.arange(n)

AI = np.linalg.inv(A.T)
print(np.round(AI, 2))
```

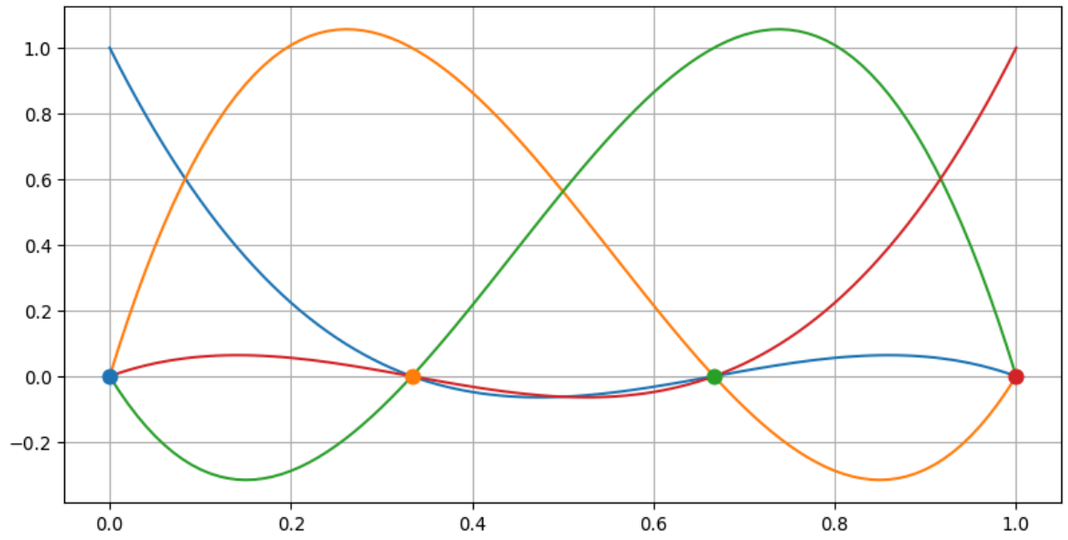
```
[[   1.     -8.33   23.33  -26.67   10.67]
 [   0.     16.    -69.33   96.    -42.67]
 [   0.    -12.     76.    -128.     64.  ]
 [   0.      5.33  -37.33   74.67  -42.67]
 [  -0.     -1.      7.33  -16.     10.67]]
```

Cubic elements

$$u(\xi) = u_0(1 - 5.5\xi + 9\xi^2 - 4.5\xi^3) + u_{1/3}(9\xi - 22.5\xi^2 + 13.5\xi^3) +$$

$$+u_{2/3}(-4.5\xi + 18\xi^2 - 13.5\xi^3) + u_1(\xi - 4.5\xi^2 + 4.5\xi^3)$$

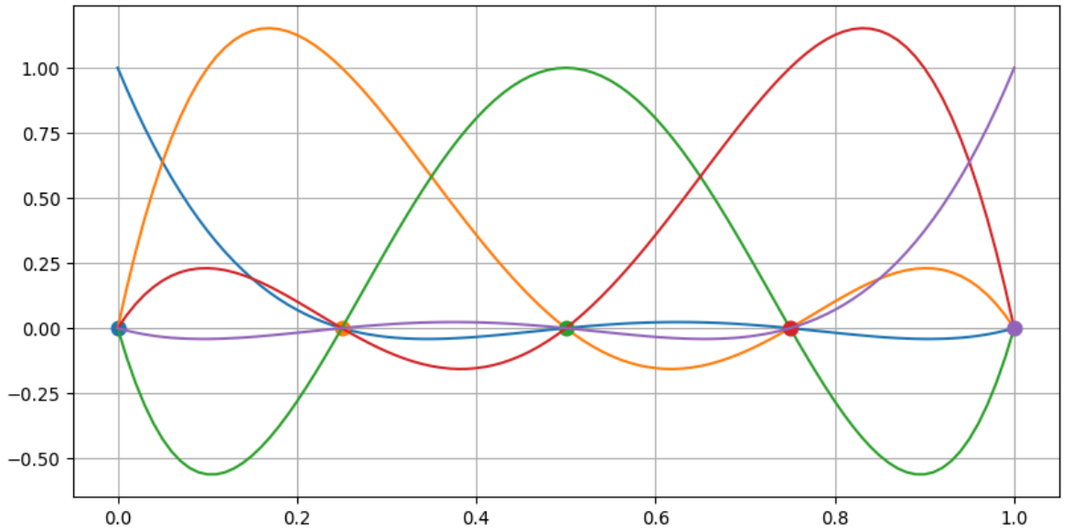
```
plt.plot(x, 1-5.5*x+9*x**2-4.5*x**3)
plt.plot(x, 9*x-22.5*x**2+13.5*x**3)
plt.plot(x, -4.5*x+18*x**2-13.5*x**3)
plt.plot(x, x-4.5*x**2+4.5*x**3)
for i in range(4):
    plt.plot(i/(3), 0, "o",
             color=f"C{i}", ms=8)
plt.grid()
```



Fourth-order elements

```
for i in range(n):
    y = np.zeros_like(x)
    for j in range(n):
        y += AI[i, j]*x**j

    plt.plot(x, y)
    plt.plot(i/(n-1), 0, "o",
             color=f"C{i}", ms=8)
plt.grid()
```

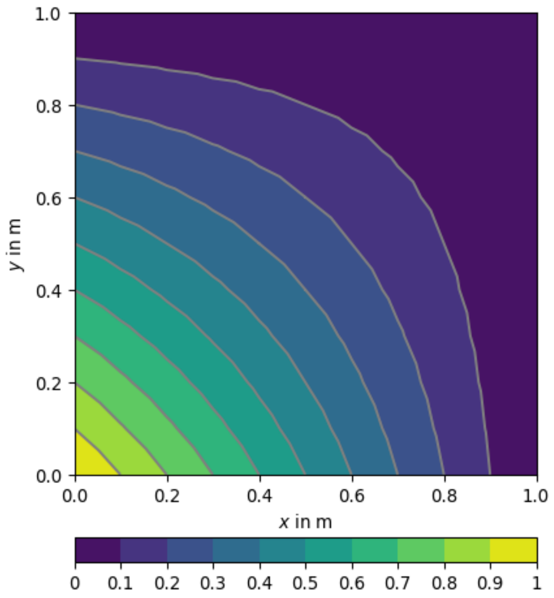


Higher dimensions

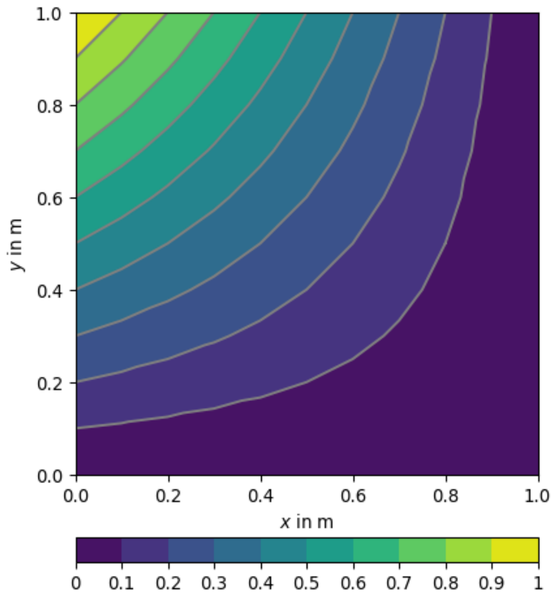
2D: Rectangular grid

- constructing shape functions from *crossing* 1D shape functions

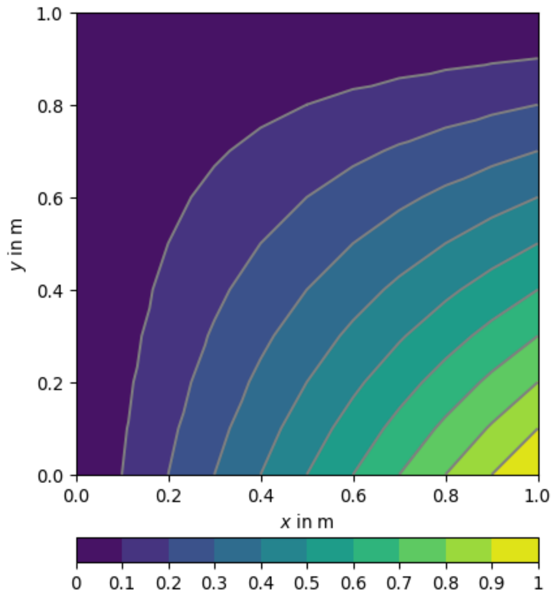
$$(1-\xi)\cdot(1-\eta)$$



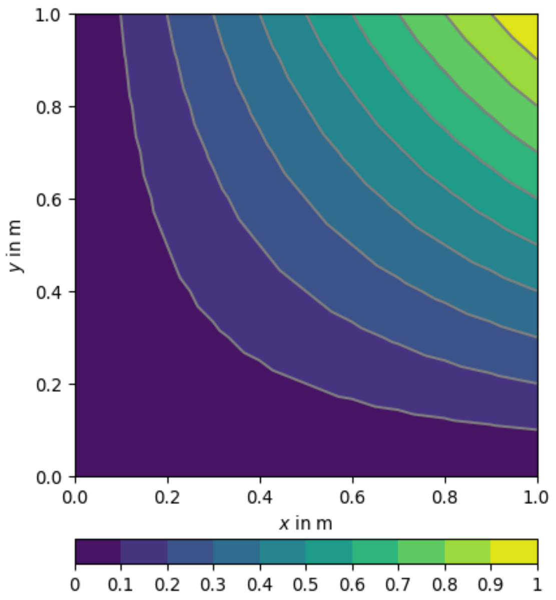
$$(1-\xi)\cdot\eta$$



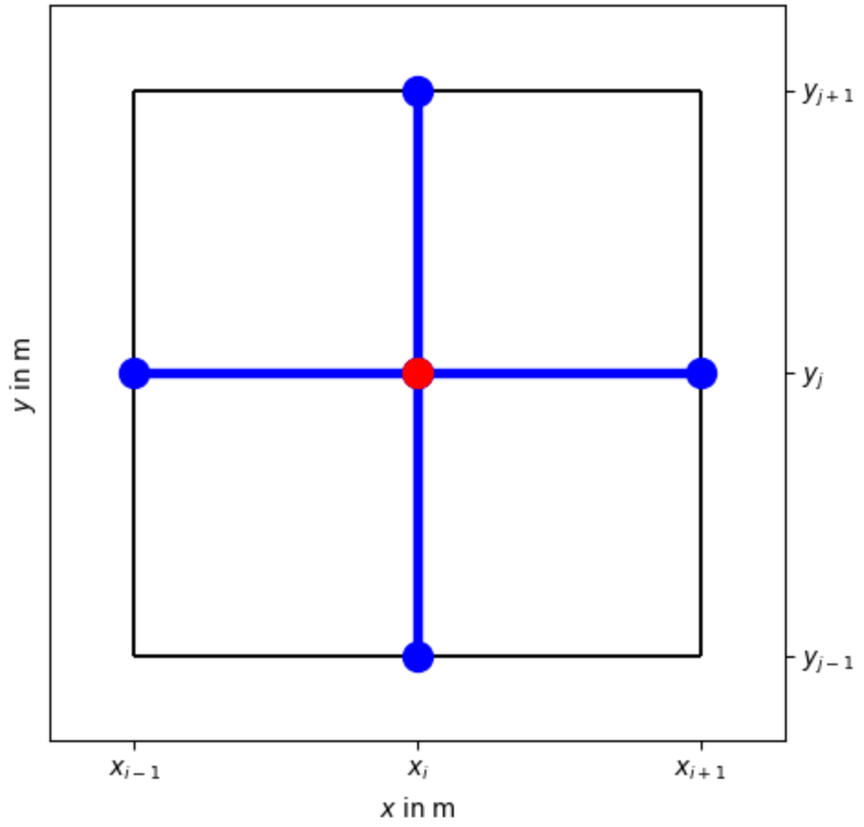
$$\xi\cdot(1-\eta)$$

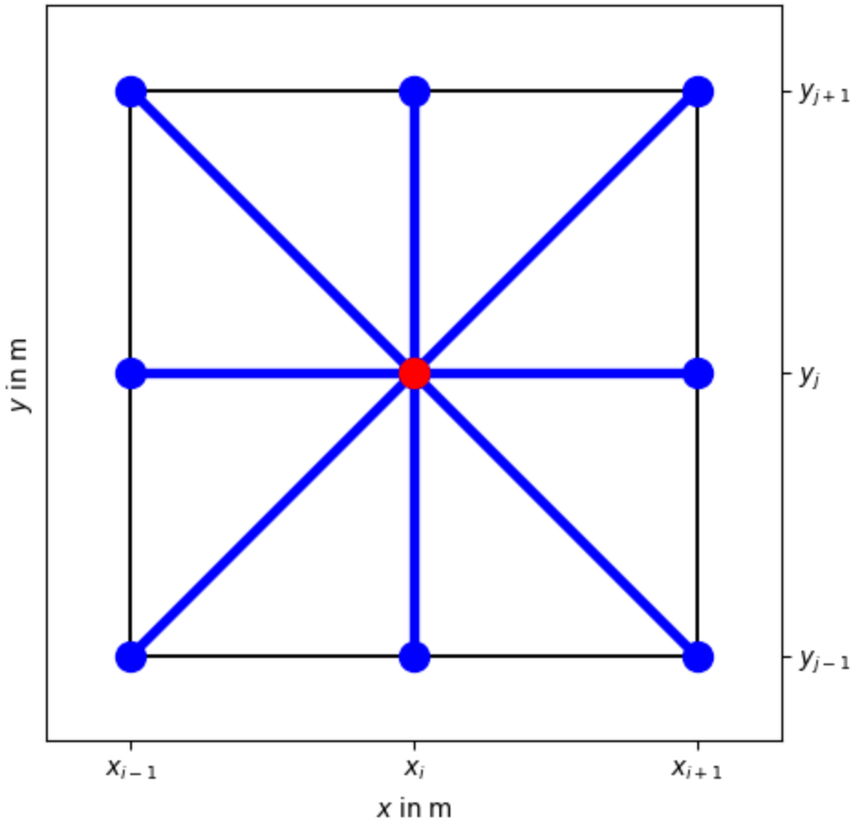


$$\xi\cdot\eta$$



Difference stencils FD vs. FE

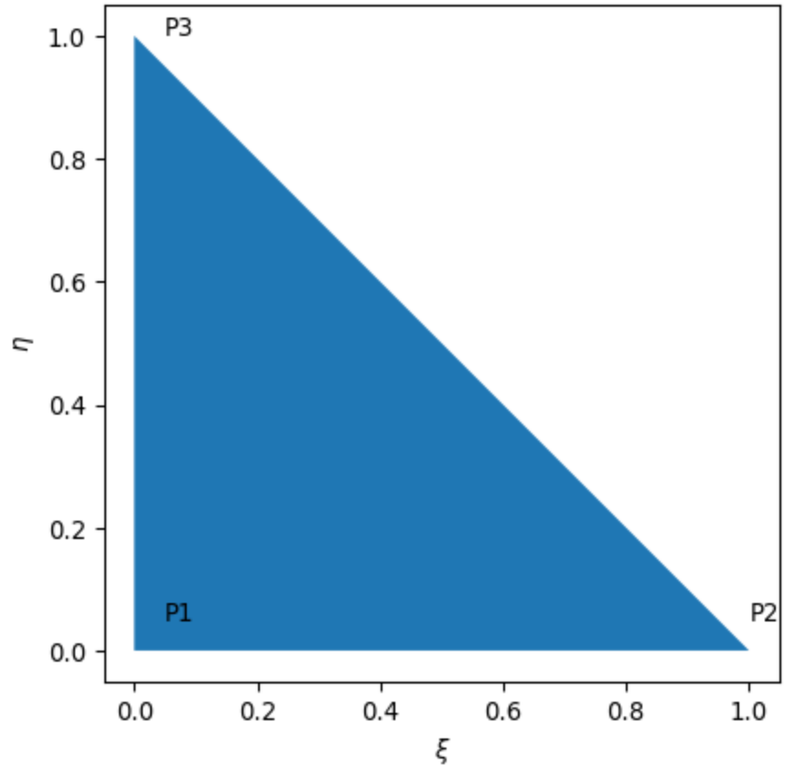




Triangles with linear shape functions

$$x = x_1 + (x_2 - x_1)\xi + (x_3 - x_1)\eta$$

$$y = y_1 + (y_2 - y_1)\xi + (y_3 - y_1)\eta$$

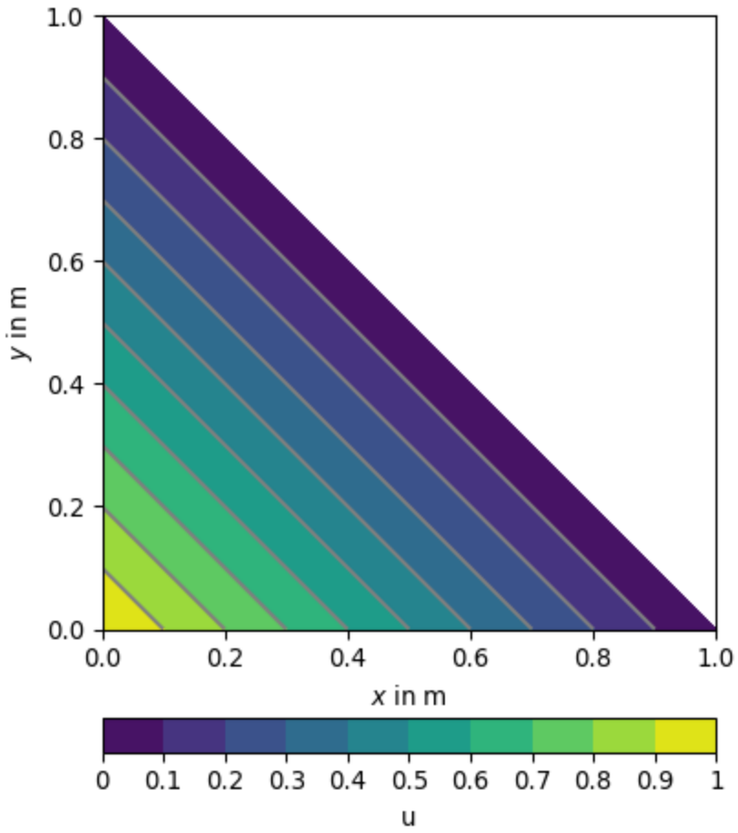


Triangle

$$u(\xi) = u_1(1 - \xi - \eta) + u_2\xi + u_3\eta$$

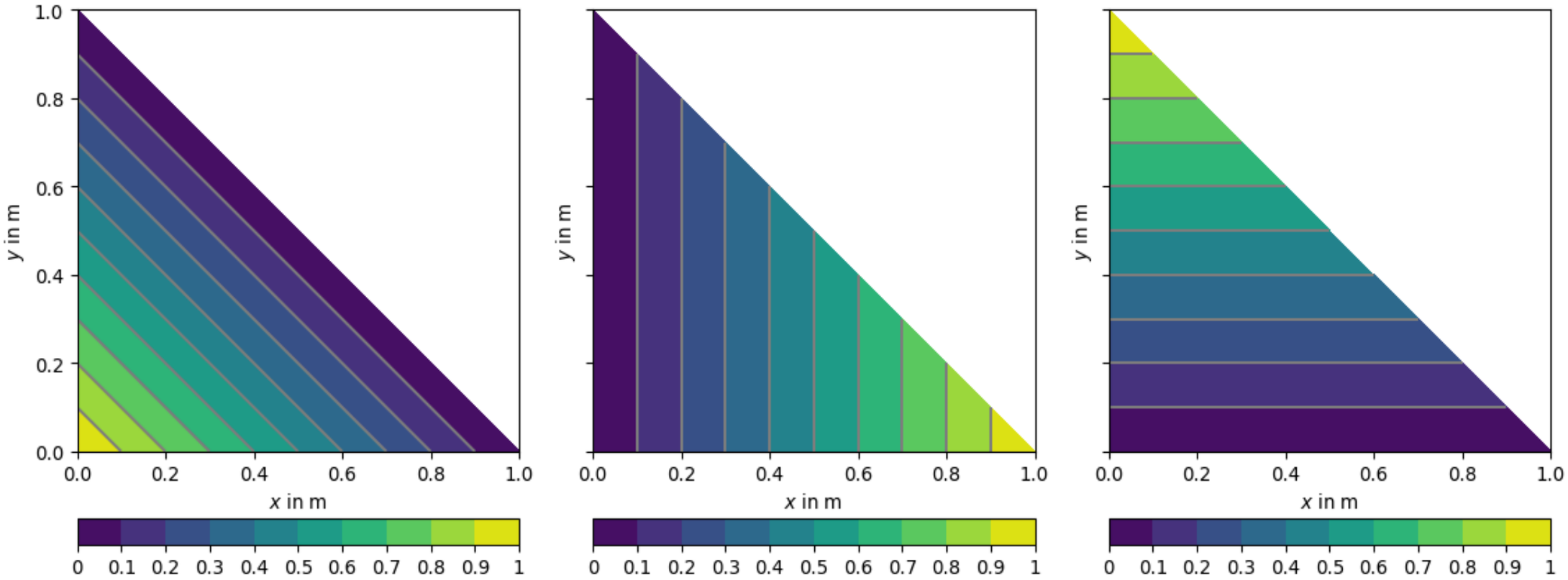
```python
import pygimli as pg
import pygimli.meshtools as mt

shape = mt.createPolygon(
    [[0, 0], [1, 0],[0, 1]],
    isClosed=True)
mesh = mt.createMesh(shape, area=0.01)
mx = pg.x(mesh)
my = pg.y(mesh)
# Plot velocity distribution.
fig, ax = plt.subplots()
pg.show(mesh, 1-mx-my, ax=ax,
        nLevs=11, label="u");
```



Triangle linear shape functions

$$u(\xi) = u_1(1 - \xi - \eta) + u_2\xi + u_3\eta$$



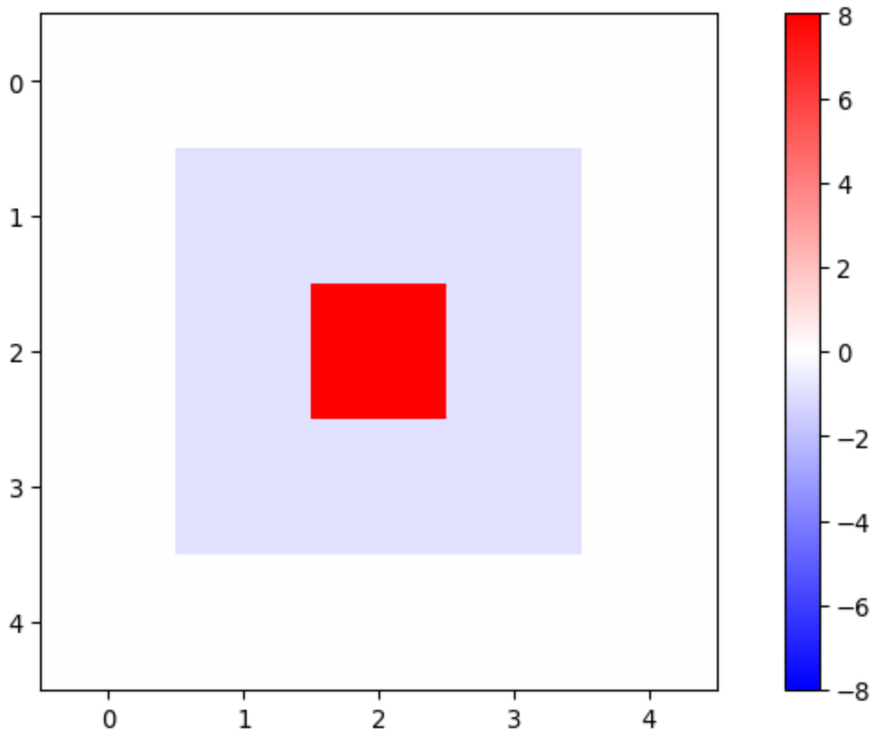
The pyGIMLi solver module

- `div`, `grad` operators
- `createStiffnessMatrix`, `createMassMatrix` assembling
- `createLoadVector`
- `assembleDirichletBC`, `assembleNeumannBC`
- `solveFiniteElements`, `solveFiniteElements` as ready equation solvers

2D rectangular grid coupling coefficients

```
1  grid5 = pg.createGrid(5, 5)
2  A = pg.solver.createStiffnessMatrix(grid5)
3  one = np.zeros(grid5.nodeCount())
4  one[12] = 1
5  row = np.reshape(A.dot(one), [5, 5])
6  print(np.round(row, 2))
7  plt.imshow(row*3, cmap="bwr", vmin=-8, vma
8  plt.colorbar();
```

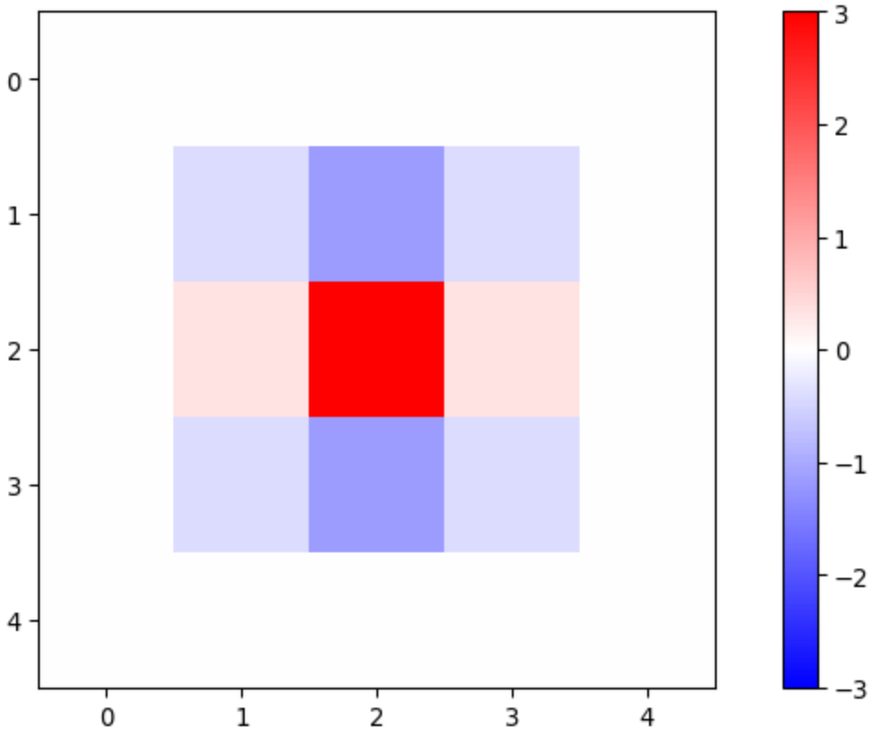
```
[[ 0.    0.    0.    0.    0.  ]
 [ 0.   -0.33 -0.33 -0.33  0.  ]
 [ 0.   -0.33  2.67 -0.33  0.  ]
 [ 0.   -0.33 -0.33 -0.33  0.  ]
 [ 0.    0.    0.    0.    0.  ]]
```



2D rectangular grid scaling

```
1  grid5.scale([1, 0.5])
2  A = pg.solver.createStiffnessMatrix(grid5)
3  row = A.dot(one)
4  rowMat = np.reshape(row, [5, 5])
5  print(np.round(rowMat, 2))
6  plt.imshow(rowMat, cmap="bwr", vmin=-3, vma
7  plt.colorbar();
```

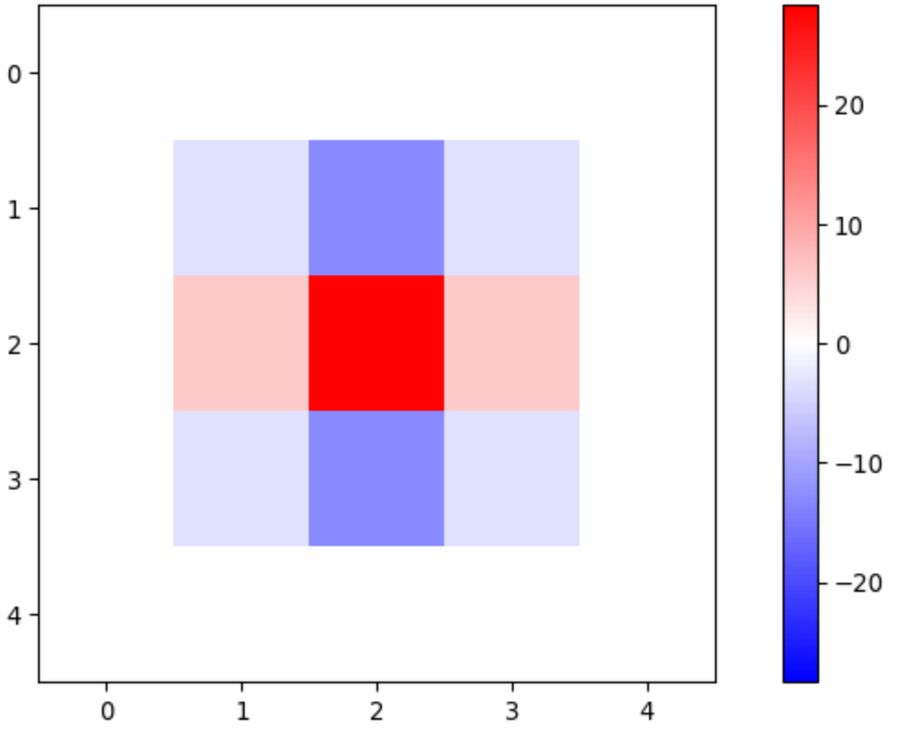
```
[[ 0.    0.    0.    0.    0.  ]
 [ 0.   -0.42 -1.17 -0.42  0.  ]
 [ 0.    0.33  3.33  0.33  0.  ]
 [ 0.   -0.42 -1.17 -0.42  0.  ]
 [ 0.    0.    0.    0.    0.  ]]
```



2D rectangular grid conductivity

```
1  grid5.scale([1, 0.5])
2  A = pg.solver.createStiffnessMatrix(grid5,
3  row = A.dot(one)
4  rowMat = np.reshape(row, [5, 5])
5  print(np.round(rowMat, 2))
6  plt.imshow(rowMat, vmin=-max(row),
7             cmap="bwr", vmax=max(row))
8  plt.colorbar();
```

```
[[  0.     0.     0.     0.     0.  ]
 [  0.    -3.54 -12.92  -3.54   0.  ]
 [  0.     5.83  28.33   5.83   0.  ]
 [  0.    -3.54 -12.92  -3.54   0.  ]
 [  0.     0.     0.     0.     0.  ]]
```



The general solution

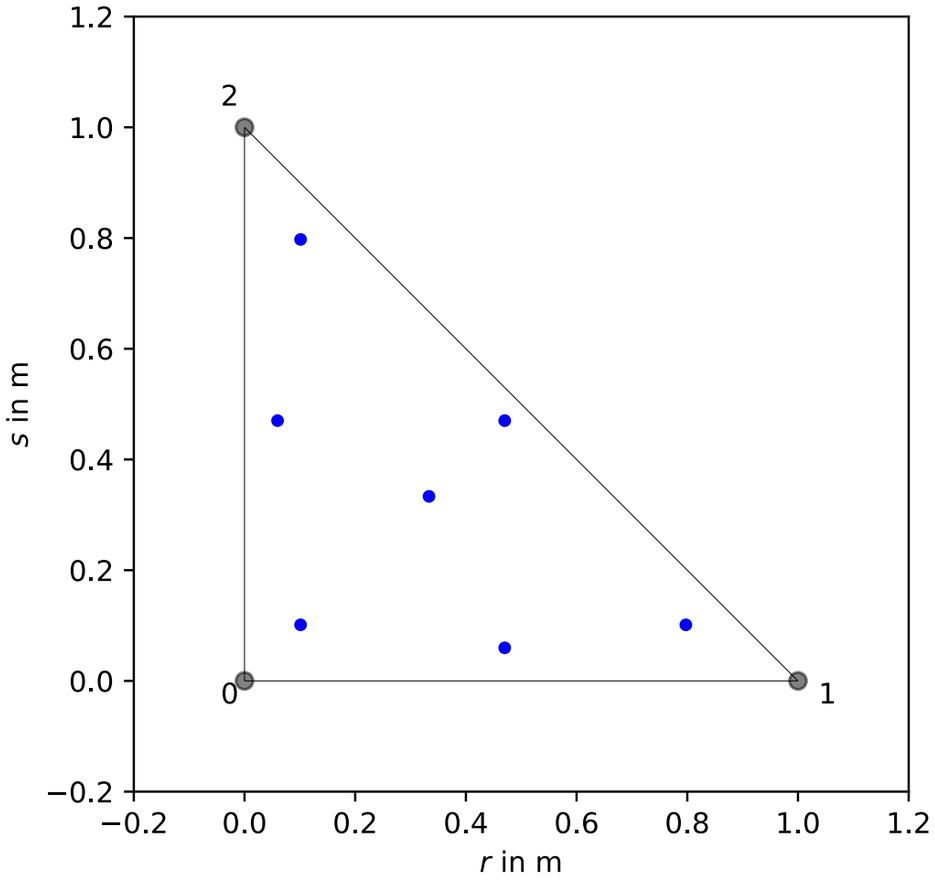
Solving any integral using (Gaussian) quadrature

$$\int g(x)\mathrm{d}x \approx \sum_q g(x_q)w_q$$

$$f_i^c = \int_{\Omega_c} v_i f \mathrm{d}x \approx \sum_q v(x_q^c) f(x_q^c) w_q^c$$

$$a_{ij}^c = \int_c a_c \boldsymbol{\nabla} v_i \cdot \boldsymbol{\nabla} v_j = \sum a_c \boldsymbol{\nabla} v_i(x_q^c) \cdot \boldsymbol{\nabla} v_j(x_q^c) w_q^c$$

Gaussian quadrature

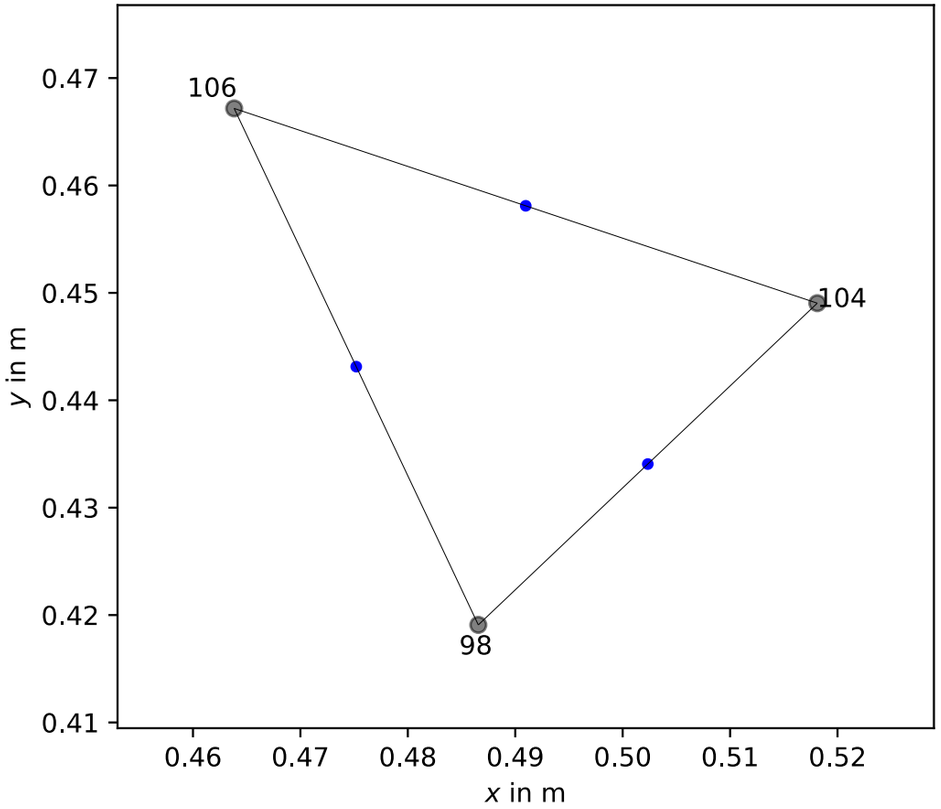


Quadrature points

```
quadratureRules(c.shape(), 5)
```

- optimum quadrature on reference triangle for a given order (5)

Gaussian quadrature



Quadrature points

```
quadratureRules(c, 2)
```

- optimum quadrature on arbitrary triangle for order 2

Verification

1. Method of Manufactured Solutions (MMS)

- manufacture a smooth u
- generate f matching approximation of u

2. Method of Exact Solutions (MES)

- find parameters for which an analytic solution exists

3. Perform convergence tests for increasingly smaller h

- approximation error $E(h) < Ch^n$ test for some h

Green’s functions

The Green’s function G is the solution for a Dirac source δ

$$\mathfrak{L}G = \delta(\mathbf{r})$$

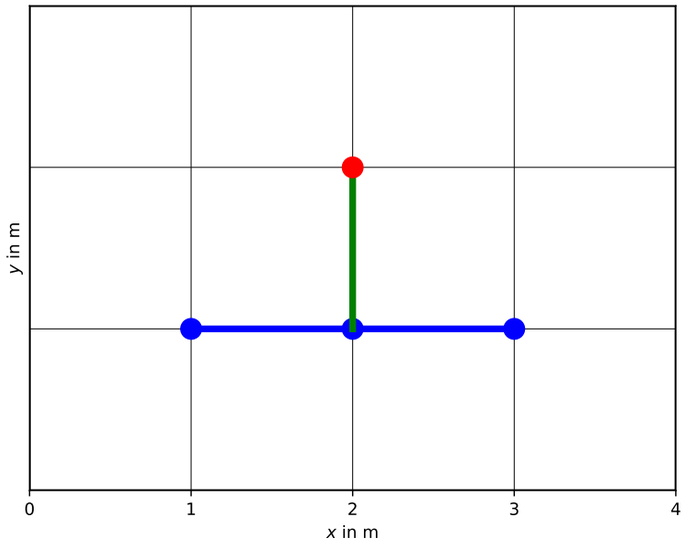
The solution can then be obtained by convolution

$$u(\mathbf{r}) = G(\mathbf{r} - \mathbf{r}') * f(\mathbf{r}') =$$

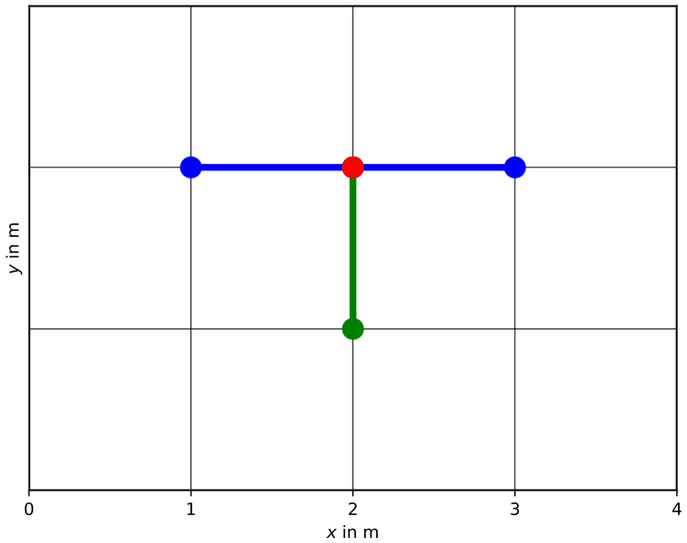
Time-stepping in FE

Recap time-stepping in FD

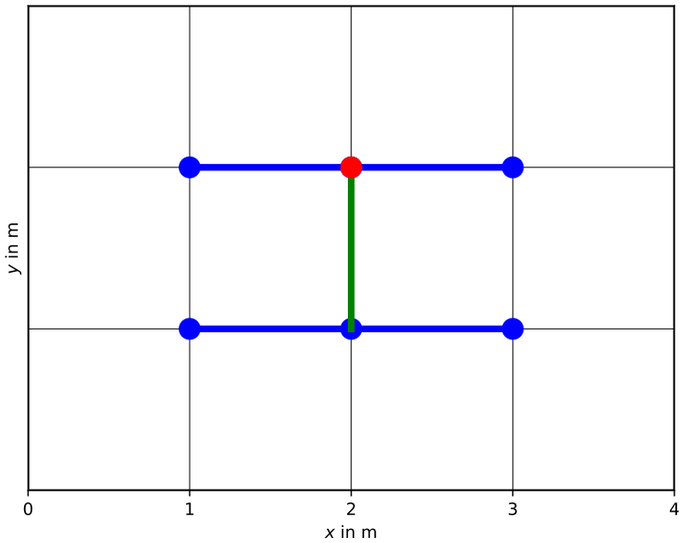
Explicit:		$\mathbf{u}^{n+1}$	$=$	$(\mathbf{I} - \Delta t \mathbf{A})$ $\mathbf{u}^n$
Implicit:	$(\mathbf{I} + \Delta t \mathbf{A})$	$\mathbf{u}^{n+1}$	$=$	$\mathbf{u}^n$
Mixed:	$(2\mathbf{I} + \Delta t \mathbf{A})$	$\mathbf{u}^{n+1}$	$=$	$(2\mathbf{I} - \Delta t \mathbf{A})$ $\mathbf{u}^n$



Explicit



Implicit



Mixed

Variational formulation of Diffusion equation

$$\frac{\partial u}{\partial t} - \boldsymbol{\nabla} \cdot a \boldsymbol{\nabla} u = f$$

Finite Difference in Time (NOT in space)

$$\frac{u^{n+1} - u^n}{\Delta t} - \boldsymbol{\nabla} \cdot a \boldsymbol{\nabla} u = f$$

Variational formulation

$$\frac{u^{n+1} - u^n}{\Delta t} - \boldsymbol{\nabla} \cdot a \boldsymbol{\nabla} u = f$$

Multiplication with test function w and integration $\Rightarrow$ weak form

$$1/\Delta t(\int_\Omega w u^{n+1} \mathrm{d}\Omega - \int_\Omega w u^n \mathrm{d}\Omega) - \int_\Omega w \boldsymbol{\nabla} \cdot a \boldsymbol{\nabla} u \mathrm{d}\Omega = \int_\Omega w f \mathrm{d}\Omega$$

$$1/\Delta t(\int_\Omega w u^{n+1} \mathrm{d}\Omega - \int_\Omega w u^n \mathrm{d}\Omega) - \int_\Omega a \boldsymbol{\nabla} w \cdot \boldsymbol{\nabla} u \mathrm{d}\Omega = \int_\Omega w f \mathrm{d}\Omega$$

Variational formulation of diffusion equation

u is constructed of shape functions $\mathbf{v}_i$ that are identical to w

The integral over the Poisson term

$$-\int_\Omega a\boldsymbol{\nabla} w \cdot \boldsymbol{\nabla} u \mathrm{d}\Omega$$

is represented by $\mathbf{A}\mathbf{v}$ using the stiffness matrix

$$\mathbf{A}_{i,j} = \int_\Omega \sigma\boldsymbol{\nabla} v_i \cdot \boldsymbol{\nabla} v_j \mathrm{d}\Omega$$

The mass matrix

Weighted integrals over both u are represented by the mass matrix $\mathbf{Mv}$

$$\mathbf{M}_{i,j} = \int_{\Omega} v_i \cdot v_j \mathrm{d}\Omega$$

explicit method (use u^n): $\mathbf{Mu}^{n+1} = (\mathbf{M} - \Delta t \mathbf{A})\mathbf{u}^n$

implicit method (use u^{n+1}): $(\mathbf{M} + \Delta t \mathbf{A})\mathbf{u}^{n+1} = \mathbf{Mu}^n$

mixed method (mix u^n/u^{n+1}):
$(2\mathbf{M} + \Delta t \mathbf{A})\mathbf{u}^{n+1} = (2\mathbf{M} - \Delta t \mathbf{A})\mathbf{u}^n$

Time-stepping in FE

Explicit:	$\mathbf{M}$	$\mathbf{u}^{n+1}$	$=$	$(\mathbf{M} - \Delta t \mathbf{A})$ $\mathbf{u}^{n}$
Implicit:	$(\mathbf{M} + \Delta t \mathbf{A})$	$\mathbf{u}^{n+1}$	$=$	$\mathbf{M}$ $\mathbf{u}^{n}$
Mixed:	$(2\mathbf{M} + \Delta t \mathbf{A})$	$\mathbf{u}^{n+1}$	$=$	$(2\mathbf{M} - \Delta t \mathbf{A})$ $\mathbf{u}^{n}$

Tip

same as in FD but with FE mass matrix $\mathbf{M}$ instead of $\mathbf{I}$

The mass matrix in 1D

$$\mathbf{M}_{i,j} = \int_{\Omega} v_i \cdot v_j \mathrm{d}\Omega$$

$$\mathbf{M}_{i,i+1} = \int_{x_i}^{x_{i+1}} \frac{x_{i+1} - x}{\Delta x_i} \frac{x - x_i}{\Delta x_i} \mathrm{d}x = \int_0^1 (1-\xi)\xi \Delta x_i \mathrm{d}\xi$$

$$\Rightarrow \mathbf{M}_{i,i+1} = \Delta x_i \int_0^1 (\xi - \xi^2) = \Delta x_i \left| \frac{1}{2}\xi^2 - \frac{1}{3}\xi^3 \right|_0^1 = \frac{\Delta x_i}{6}$$

The mass matrix in 1D

$$\mathbf{M}_{i,i} = \Delta x_{i-1} \int_0^1 \xi^2 \mathrm{d}\xi + \Delta x_i \int_0^1 (1-\xi)^2 \mathrm{d}\xi$$

$$\mathbf{M}_{i,i} = \Delta x_{i-1} \left| \frac{1}{3}\xi^3 \right|_0^1 - \Delta x_i \left| \frac{1}{3}\xi^3 \right|_1^0$$

$$\Rightarrow \mathbf{M}_{i,i} = \frac{\Delta x_{i-1}}{3} + \frac{\Delta x_i}{3} = 2(\mathbf{M}_{i,i-1} + \mathbf{M}_{i,i+1})$$

$\Delta x = 1 \quad \Rightarrow \quad [1, 4, 1]$ (stiffness was [-1, 2, -1])

Inner vs. outer nodes

distinguish dofs into inner and outer $[\mathbf{u}_i, \mathbf{u}_o]^T$

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{ii} & \mathbf{A}_{io} \\ \mathbf{A}_{oi} & \mathbf{A}_{oo} \end{pmatrix}$$

$$\mathbf{A} \begin{bmatrix} \mathbf{u}_i \\ \mathbf{u}_o \end{bmatrix} = \begin{bmatrix} \mathbf{f}_i \\ \mathbf{f}_o \end{bmatrix}$$

$$\Rightarrow \mathbf{A}_{ii}\mathbf{u}_i = \mathbf{f}_i - \mathbf{A}_{oi}\mathbf{u}_o$$

Tasks

1. Write a function computing the FE stiffness matrix for 1D discretization
2. Test it by solving the Poisson equation with $f = 1$ (analytical solution)
3. Compare with analytical and FD solutions
4. Write a function computing the FE mass matrix for 1D discretization
5. Repeat the time-stepping tasks from FD with FE