Numerical Simulation Methods in Geophysics, Part 7: Finite Elements

1. MGPY+MGIN

thomas.guenther@geophysik.tu-freiberg.de

Recap Finite Elements

- weak form of PDE: integral over product with test function $\mathbf{w}$
- approximate u with shape functions $u = \sum u_i \mathbf{v}_i$
- Galerkin's method: same function space for $\mathbf{w}$ and $\mathbf{v}$

Difference of FE to FD

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

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

Recap (cont)

Generality of FE

Arbitrary base functions v_i can be used to describe u

- started with piece-wise linear (hat) functions
- system identical to FD for $\Delta x =$const and a=const

Method of weighted residuals

PDE $\mathfrak{L}(u) = f \Rightarrow$ approximated by u_h

residual $R = L_h(u) - f$ to be minimized, integrating over modelling domain

$$\int_\Omega wR\mathrm{d}\Omega = \int_\Omega w\mathfrak{L}(u_h)\mathrm{d}\Omega - \int_\Omega wf\mathrm{d}\Omega = 0$$

with approximation $u_h(\mathbf{r}) = \sum_j^M u_j \mathbf{v}_j(\mathbf{r})$

($\mathbf{v}$ basis / shape functions, $\mathbf{w}$ test / trial functions)

Bilinear form for Poisson equation

Solve $\mathbf{A}\mathbf{x} = \mathbf{b}$ with $A_{ij} = (\boldsymbol{\nabla} v_i, \boldsymbol{\nabla} v_j)$ and $b_i = (\mathbf{v}_i, f)$, where

$$(\mathbf{a}, \mathbf{b}) = \int_\Omega \mathbf{a} \cdot \mathbf{b} \, \mathrm{d}\Omega = \sum_{c=i}^{M} \int_{\Omega_c} \mathbf{a} \cdot \mathbf{b} \, \mathrm{d}\Omega_c$$

Solve the integrals either analytically or numerically

The general solution

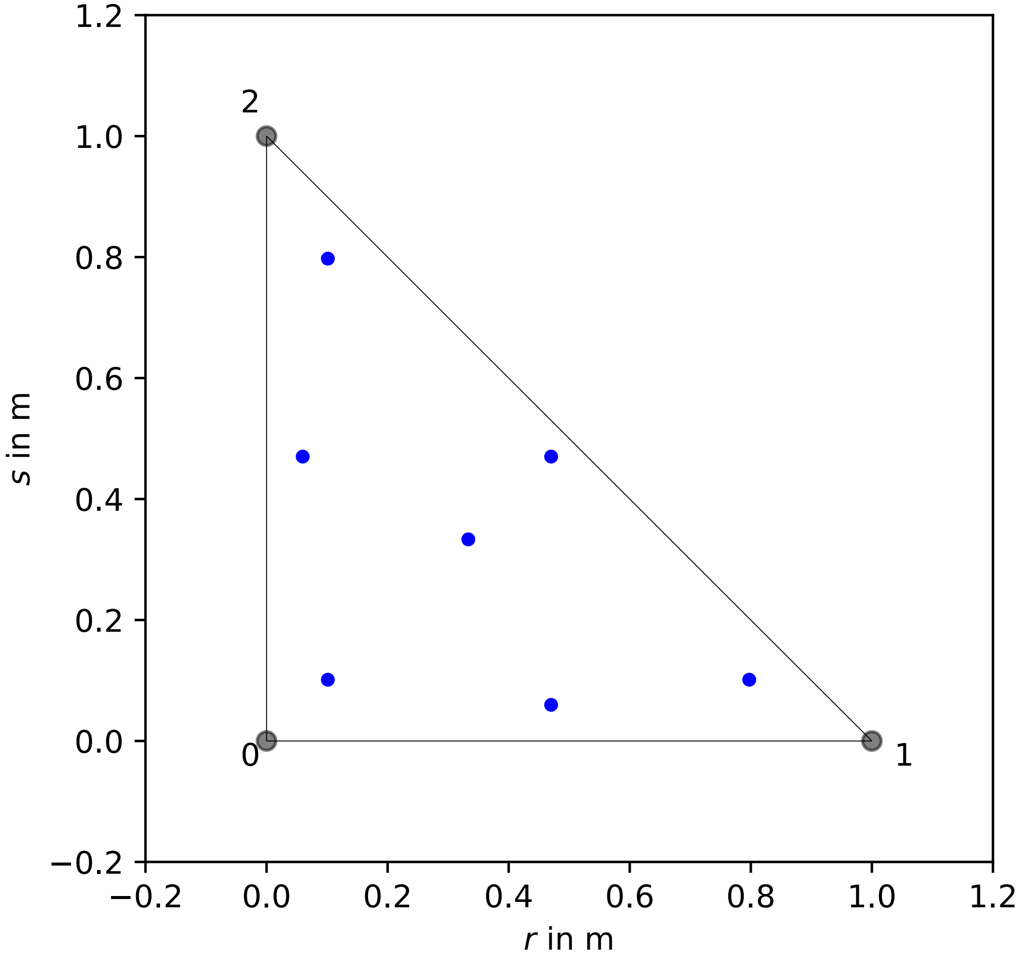
Solving any integral using (Gaussian) quadrature

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

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

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

Gaussian quadrature

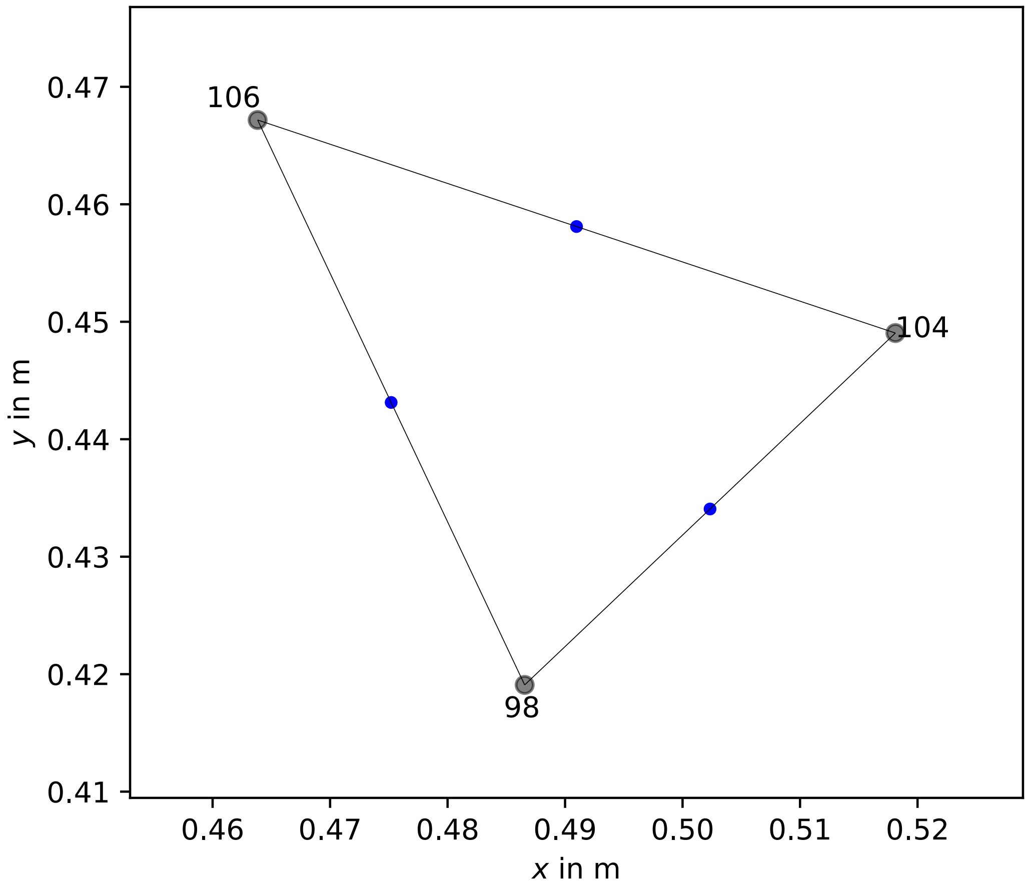


Quadrature points

`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

Coordinate transformation

1D: local coordinate $\xi = \frac{x - x_i}{x_{i+1} - x_i}$ (0..1)

$$u(\xi) = c_1 + c_2 \xi$$

$$u_0 = u(0) = c_1, u_1 = u(1) = c_1 + c_2 \; c_2 = u_1 - u_0$$

$$\Rightarrow u(\xi) = u_0 + \xi(u_1 - u_0) = u_0(1 - \xi) + u_1 \xi = u_i v_i + u_1 v_1$$

Quadratic elements

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

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

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

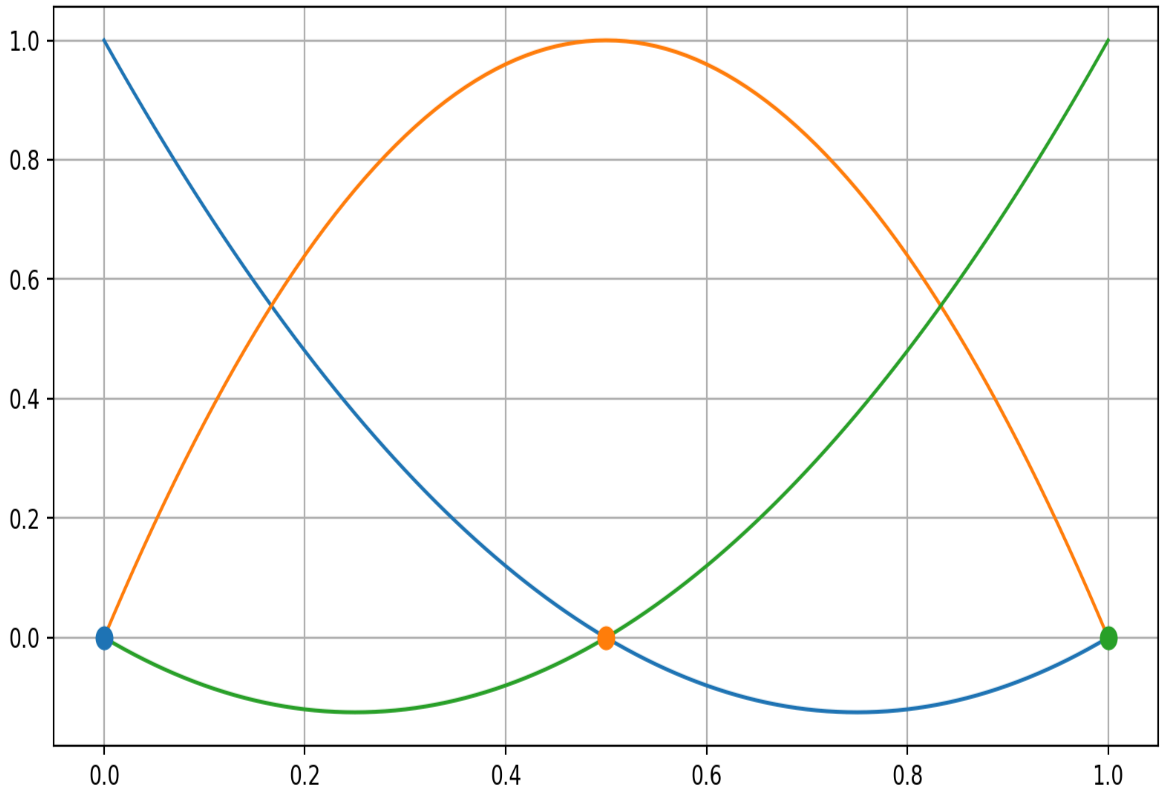
$$u_{1/2} = c_1 + c_2/2 + c_3/4$$

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

Quadratic elements

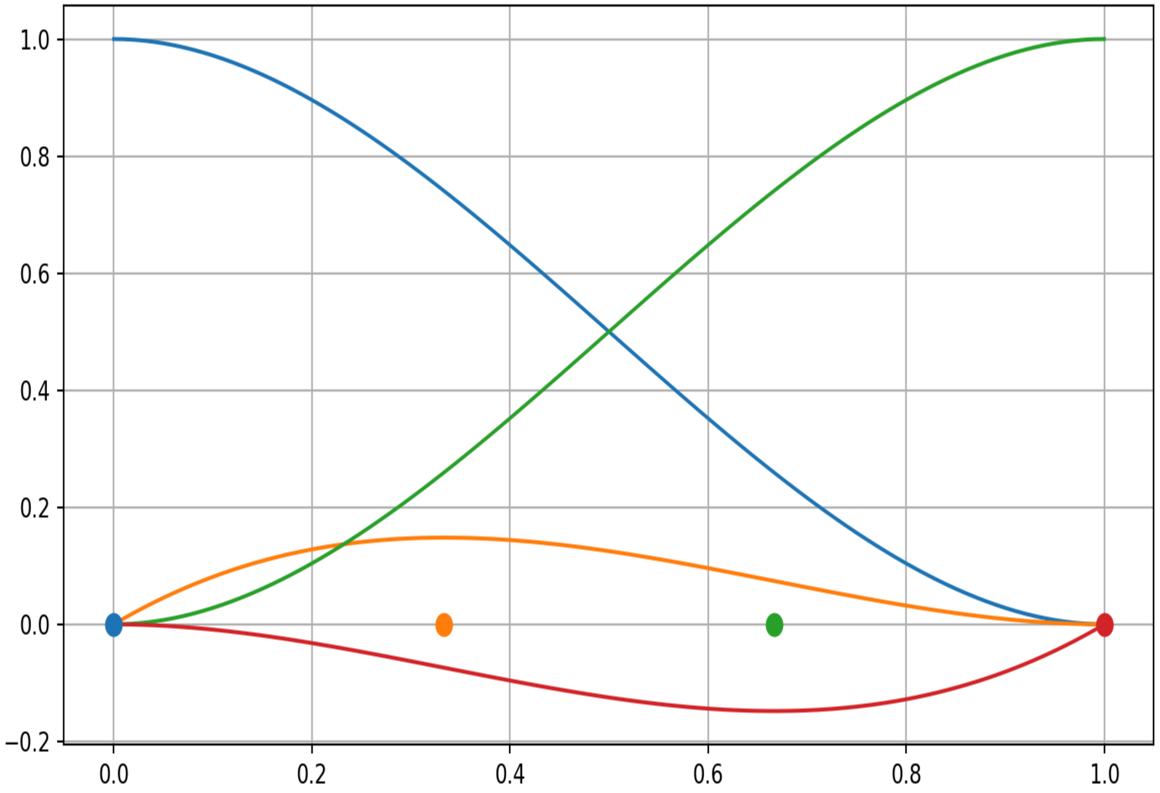
$$u(\xi) = u_0(3\xi + 2\xi^2) + u_{1/2}(4\xi - 4\xi^2) + u_1(-\xi + 2\xi^2)$$

```python
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(0, 1, 101)

# Plot velocity distribution.
plt.plot(x, 1-3*x+2*x**2)
plt.plot(x, 4*x-4*x**2)
plt.plot(x, -x+2*x**2)
plt.plot(0, 0, "o", color="C0", ms=8)
plt.plot(0.5, 0, "o", color="C1", ms=8)
plt.plot(1, 0, "o", color="C2", ms=8)
plt.grid()
```



Cubic elements

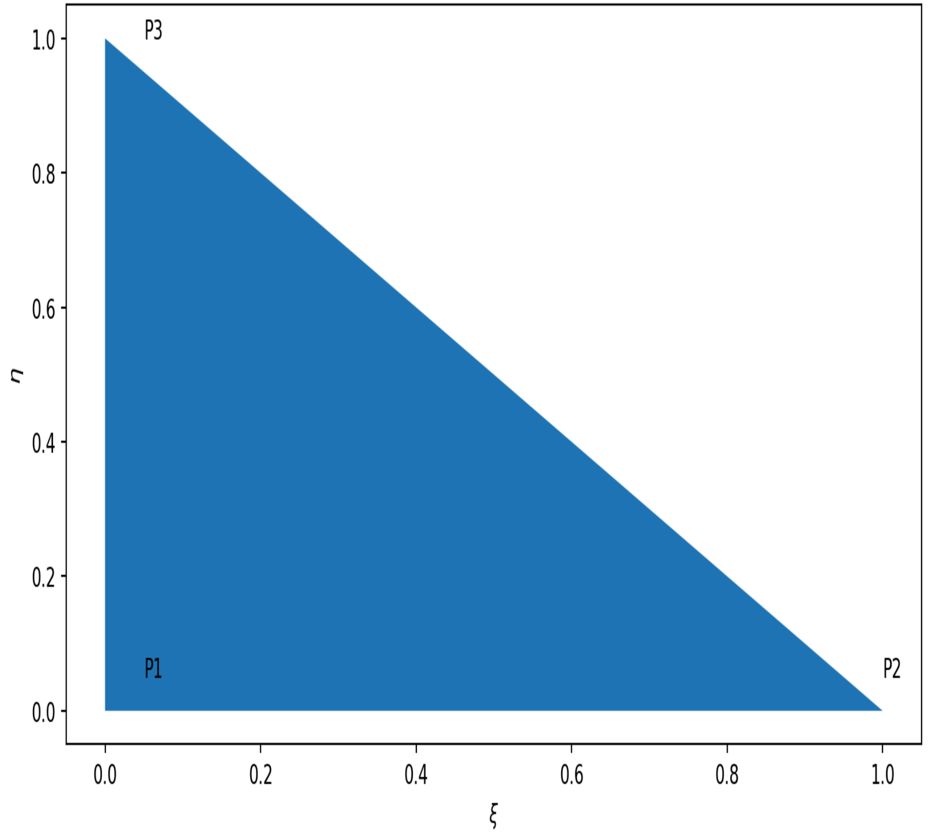
```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(0, 1, 101)

# Plot velocity distribution.
plt.plot(x, 1-3*x**2+2*x**3)
plt.plot(x, x-2*x**2+x**3)
plt.plot(x, 3*x**2-2*x**3)
plt.plot(x, -x**2+x**3)
for i in range(4):
    plt.plot(i/(3), 0, "o",
             color=f"C{i}", ms=8)
plt.grid()
```



Triangles with linear shape functions

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

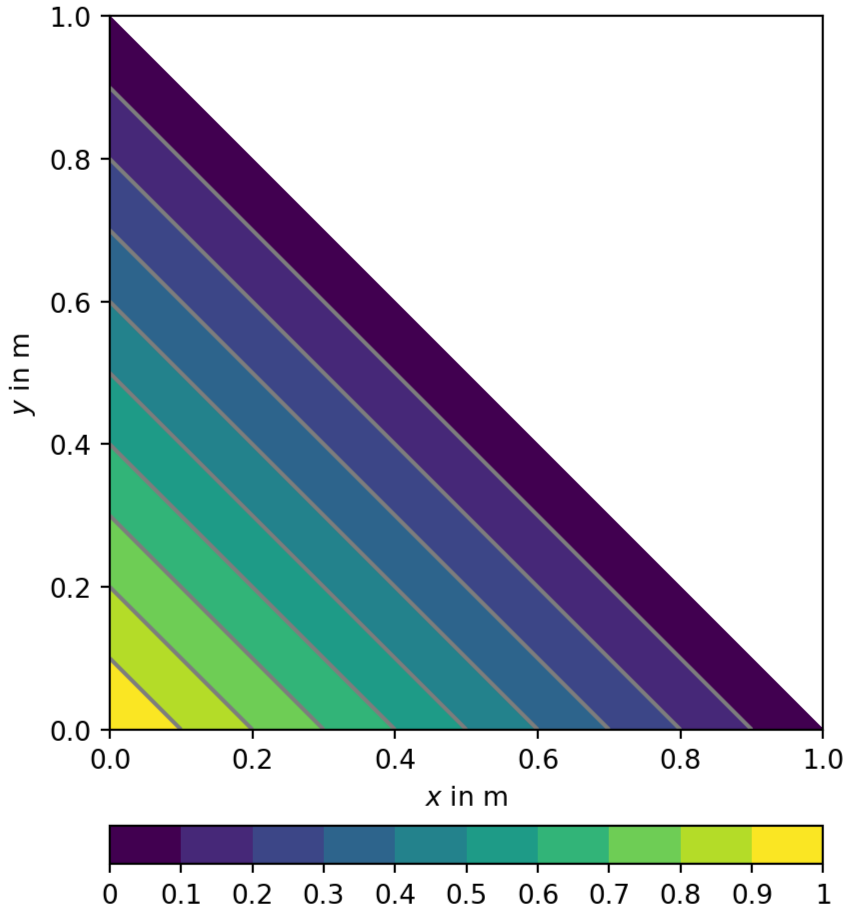
$$y = y_1 + (y_2 - y_1)\xi + (y_3 - y_1)\eta$$



Triangle

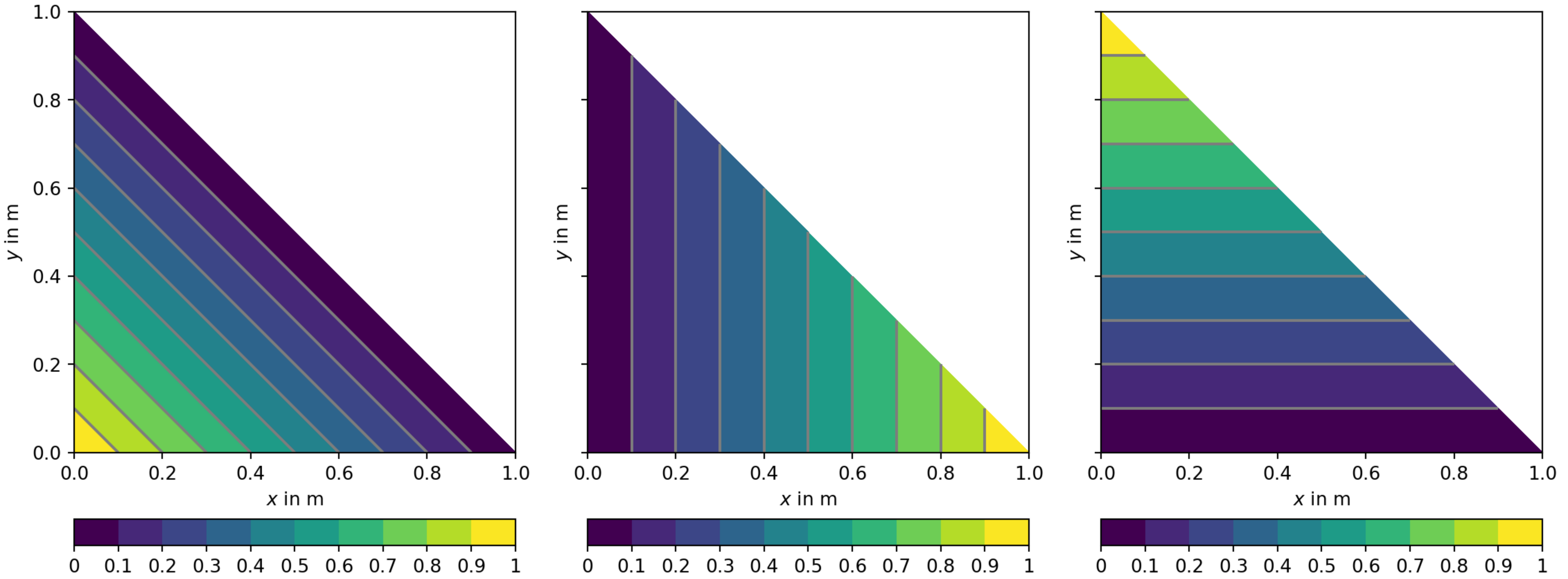
$$u(\xi) = u_1(1 - \xi - \eta) + u_2\xi + u_3\eta$$

```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);
```



Triangle linear shape functions

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



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

$$\mathfrak{L}G = \delta$$

The solution can then be obtained by convolution

$$u = G * f$$