VoronoiFVM.jl Convergence Tests

Advanced Topics from Scientific Computing
TU Berlin Winter 2024/25
Notebook 16
 Jürgen Fuhrmann

CairoMakie

```
begin
    using PlutoUI
    using VoronoiFVM
    using SimplexGridFactory
    using Triangulate
    using LinearSolve, AMGCLWrap
    using ExtendableGrids
    import CairoMakie
    using GridVisualize
    CairoMakie.activate!(type = "png")
    default_plotter!(CairoMakie)
end
```

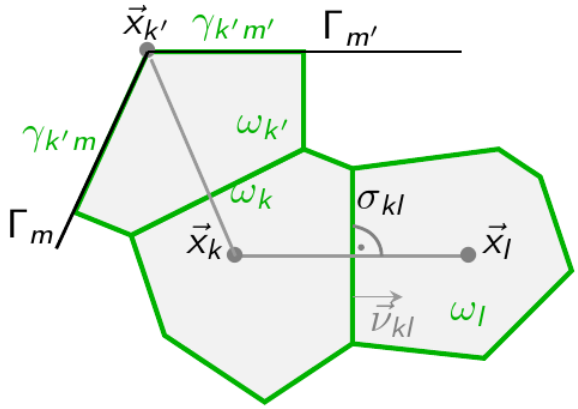
Table of Contents

Provide convergence tests for stationary solutions of linear convection-reaction-diffusion problems.

The convergence theory e.g. in R. Eymard, J. Fuhrmann, K. Gärtner. "A finite volume scheme for nonlinear parabolic equations derived from one-dimensional local Dirichlet problems." Numerische Mathematik 102 (2006): 463-495. is based on compactness arguments. While this helps to prove convergence for rather general nolinear systems, it does not provide convergence rate estimates.

Here, we provide some convergence rate tests for convection-diffusion-reaction problems with known exact solutions.

We measure the norm of the difference between the numerical solution u_h and the values of the exact solution $\hat{u}$ in the discretization points denoted by $\hat{u}_h = \Pi_h \hat{u}$ where $\Pi_h : C(\Omega) \to \mathbb{R}^n$ is the operator which evaluates a continuous function at the discretization points.



For this purpose, we use two norms:

- Discrete L^2-norm defined by

$$||u_h||_0^2 = \sum_{k \in \mathcal{N}} |\omega_k| u_k^2$$

- Discrete H^1-seminorm defined by

$$|u_h|_1^2 = \frac{1}{2} \sum_{k \in \mathcal{N}} \sum_{l \in \mathcal{N}_k} \frac{\sigma_{kl}}{h_{kl}} (u_k - u_l)^2$$

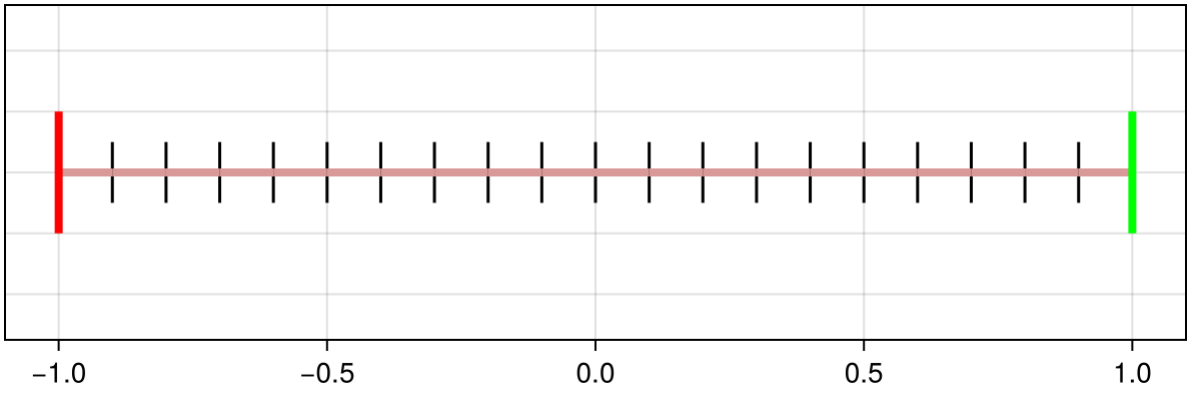
The typical converge estimates from the P1 finite element method for H^2-regular problems give

$$||u_h - \hat{u}_h||_0 = O(h^2) \quad (h \to 0)$$
$$|u_h - \hat{u}_h|_1 = O(h) \quad (h \to 0)$$

Settings

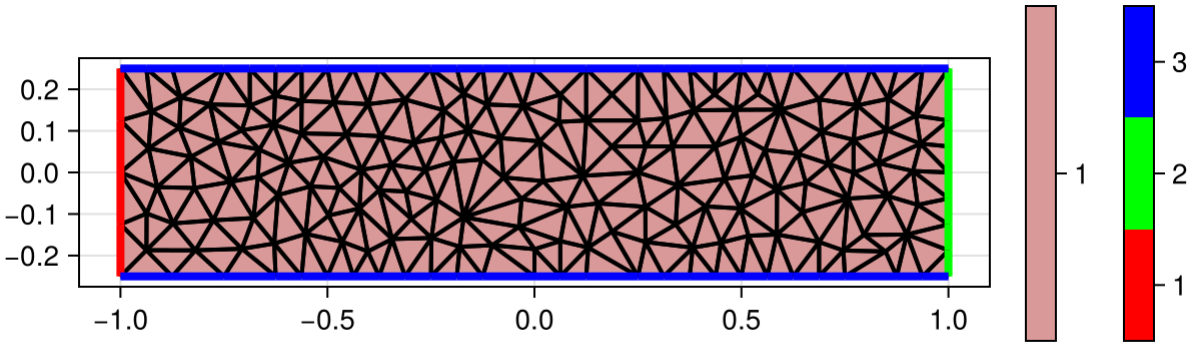
Plots of coarsest grids

1D Problems: $\Omega = (-1, 1)$




```
1 gridplot(grid1(dim = 1), resolution = (600, 200))
```

2D Problems: $\Omega = (-1, 1) \times (-0.25, 0.25)$




```
1 gridplot(grid1(dim = 2), resolution = (600, 200))
```

Parameters

```
DirectSolver(LinearSolveFunction(AMGSolverAlgorithmData(nothing, false, 1, {
                                                                    "type": "s
```

```
1 begin
2     const Nref = 7  # number of refinement levels
3     const k = 7.5   # steepness of exponential boundary layer
4     const assembly = :edgewise # instead of :cellwise, with less repeating calls
5     const strategy = DirectSolver(AMGSolverAlgorithm()) # solver strategy
6 end
```

Diffusion

$$
\begin{aligned}
-\Delta u &= 1 \\
u|_{x=-1} &= 0 \\
u|_{x=1} &= 0
\end{aligned}
$$

Exact solution:

$$u(x) = \frac{1}{2}(1 - x^2)$$

```
fexact1 (generic function with 2 methods)
```

```
begin
    fexact1(x) = (1 - x^2) / 2
    fexact1(x, y) = fexact1(x)
end
```

```
flux1 (generic function with 1 method)
```

```
function flux1(f, u, edge, data)
    f[1] = u[1, 1] - u[1, 2]
    return nothing
end
```

```
src1 (generic function with 1 method)
```

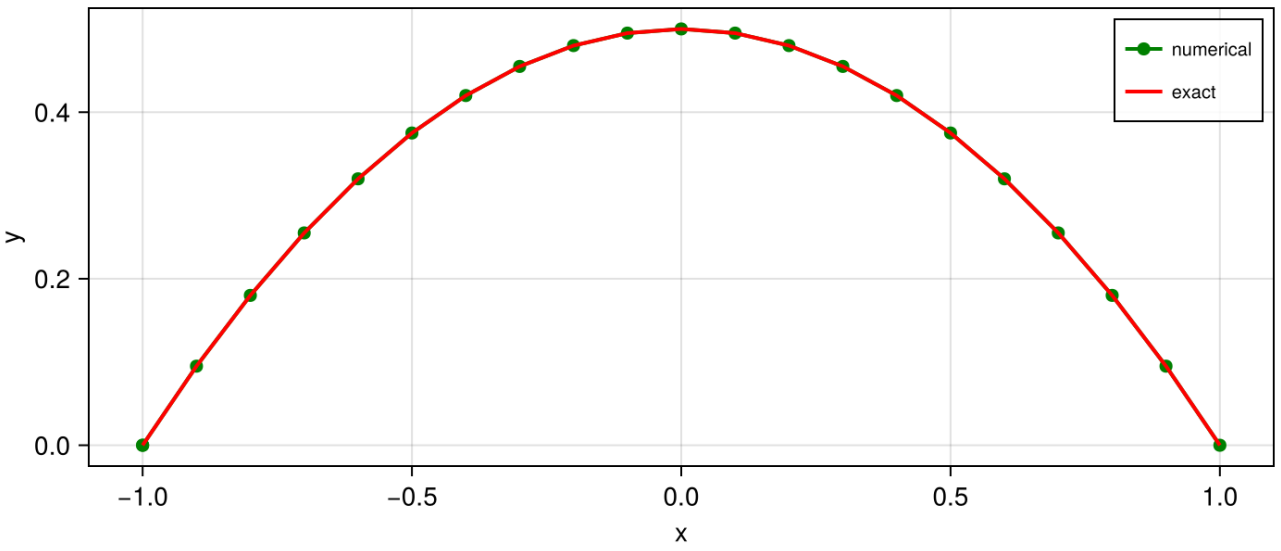
```
function src1(f, node, data)
    f[1] = 1
    return nothing
end
```

```
bc1 (generic function with 1 method)
```

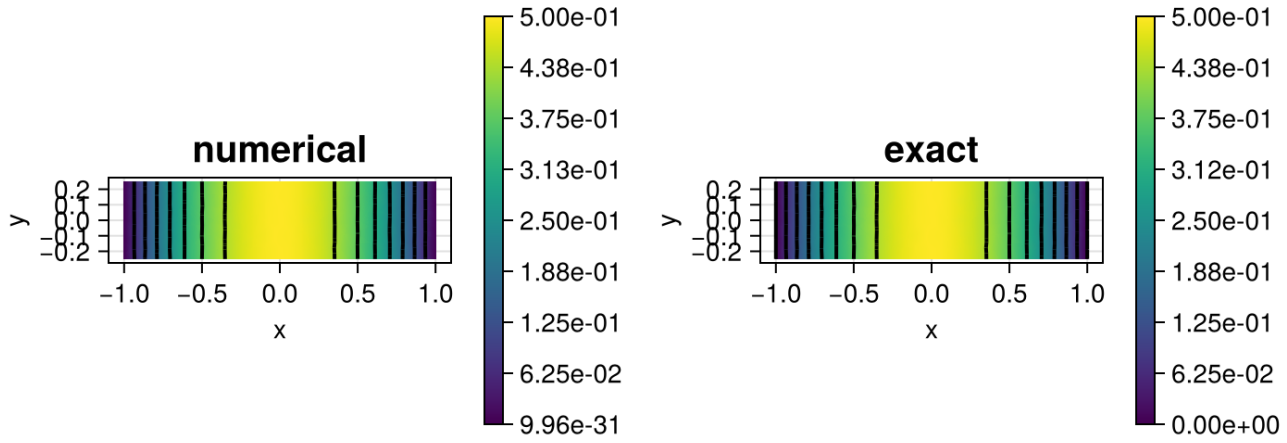
```
function bc1(f, u, bnode, data)
    boundary_dirichlet!(f, u, bnode, region = 1, value = 0)
    boundary_dirichlet!(f, u, bnode, region = 2, value = 0)
    return nothing
end
```

```
system1 (generic function with 1 method)
```

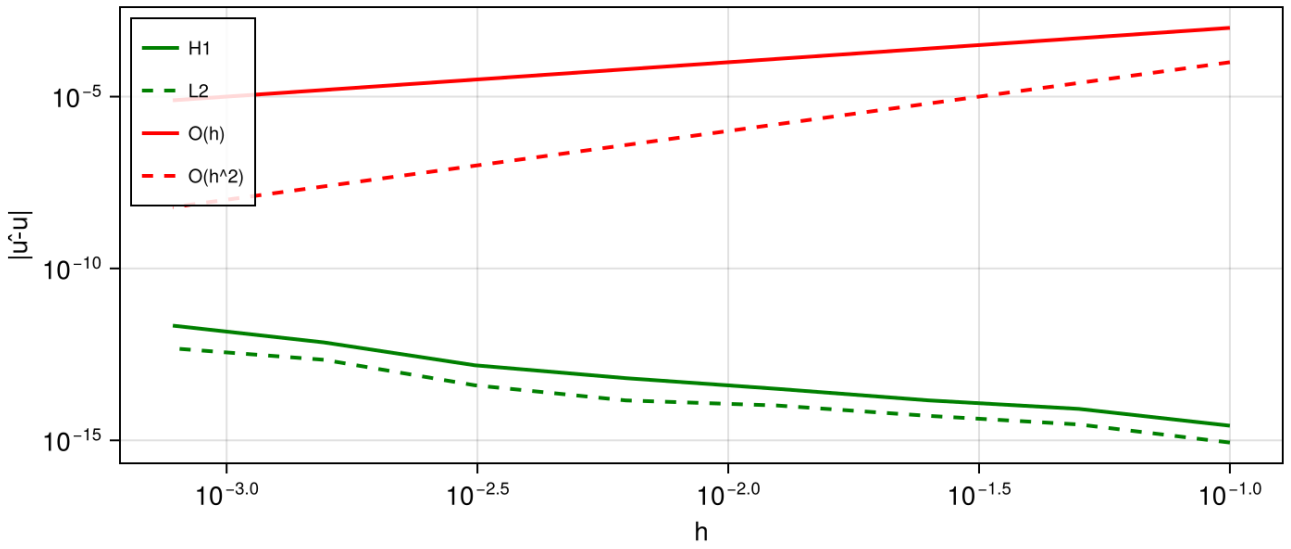
```
function system1(; h = 0.1, dim = 1)
    g = grid1(; h, dim)
    sys = VoronoiFVM.System(g; flux = flux1, source = src1, bcondition = bc1,
species = [1], assembly)
    control = SolverControl(strategy, sys)
    u = solve(sys; control)
    return sys, g, u
end
```



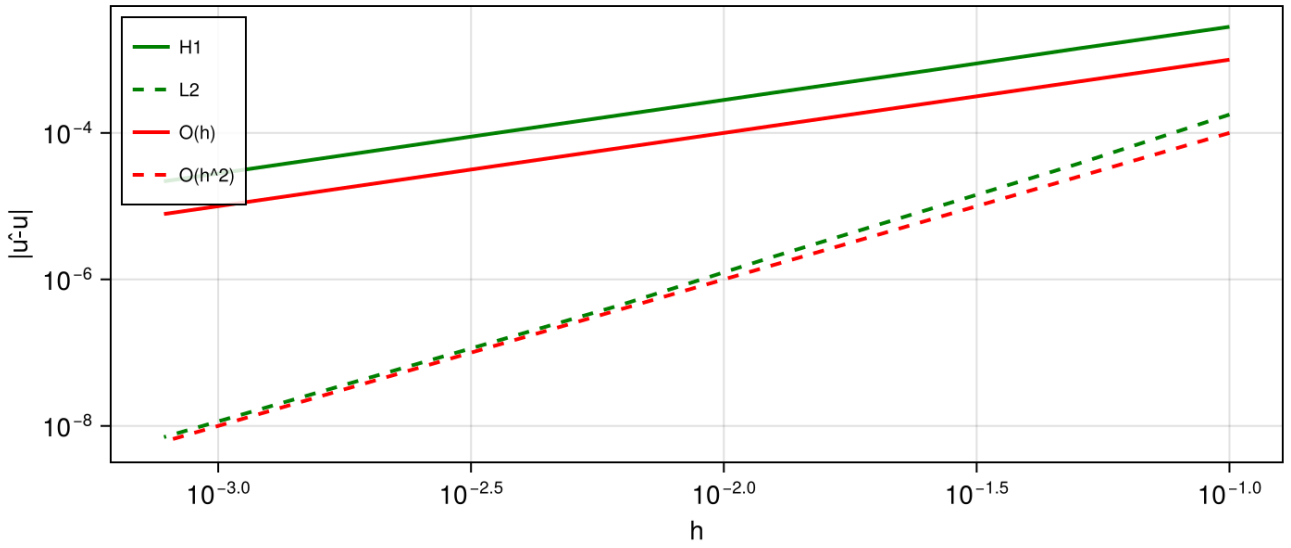

```
plot1d(system1, fexact1)
```




```
1 plot2d(system1, fexact1)
```




```
1 convtest(system1, fexact1, dim = 1)
```




```
1 convtest(system1, fexact1, dim = 2)
```

- For the 1D-Problem, the method is exact, i.e. up to roundoff errors, the discrete solution

coincides with the projection of the exact solution onto the discretization problems. This can be expected only in special cases, for general right hand sides, this would be not true. The slight increase of the error with decreasing h is due to larger floating point errors due to the increasing condition number of the matrix.
- For the 2D case, one observes first order convergence in the discrete H^1 seminorm, and second order convergence in the discrete L^2 norm.

Reaction-Diffusion

$$\begin{aligned} -\Delta u + k^2 u &= 0 \\ u|_{x=-1} &= 1 \\ u|_{x=1} &= \hat{u}(1) \end{aligned}$$

Exact solution:

$$\hat{u}(x) = \exp(-k(x+1))$$

```
fexact2 (generic function with 2 methods)
```

```
begin
    fexact2(x) = exp(-k * (x + 1))
    fexact2(x, y) = fexact2(x)
end
```

```
flux2 (generic function with 1 method)
```

```
function flux2(f, u, edge, data)
    f[1] = u[1, 1] - u[1, 2]
    return nothing
end

```

```
rea2 (generic function with 1 method)
```

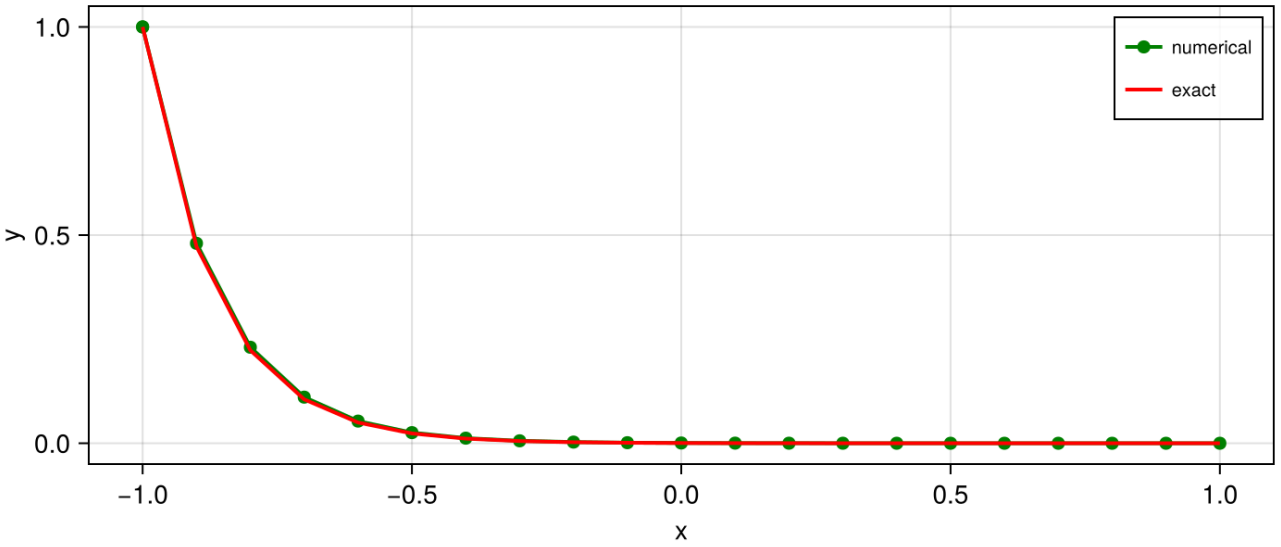
```
function rea2(f, u, node, data)
    f[1] = k^2 * u[1]
    return nothing
end

```

```
bc2 (generic function with 1 method)
```

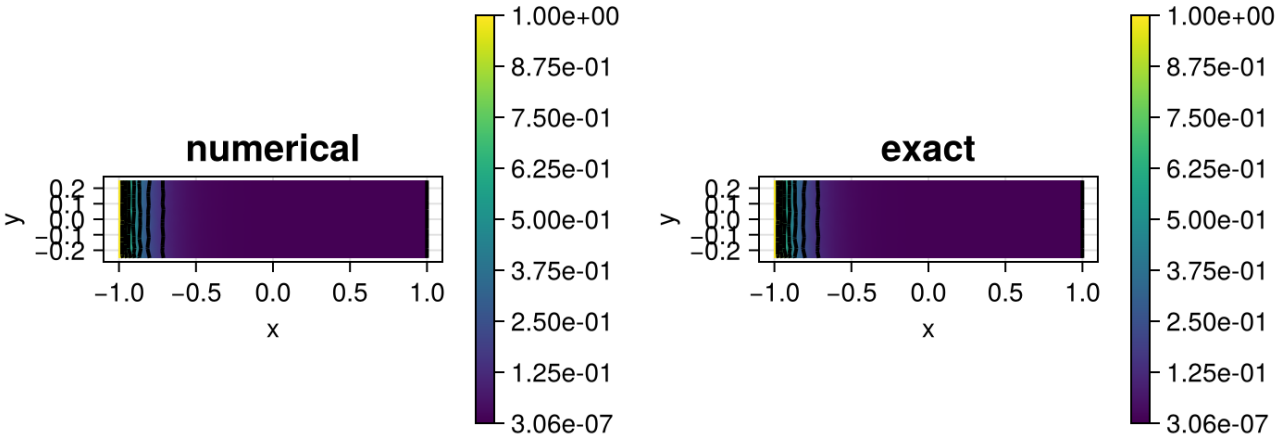
```
function bc2(f, u, bnode, data)
    boundary_dirichlet!(f, u, bnode, region = 1, value = 1)
    boundary_dirichlet!(f, u, bnode, region = 2, value = fexact2(1))
    return nothing
end

```

```
system2 (generic function with 1 method)
```

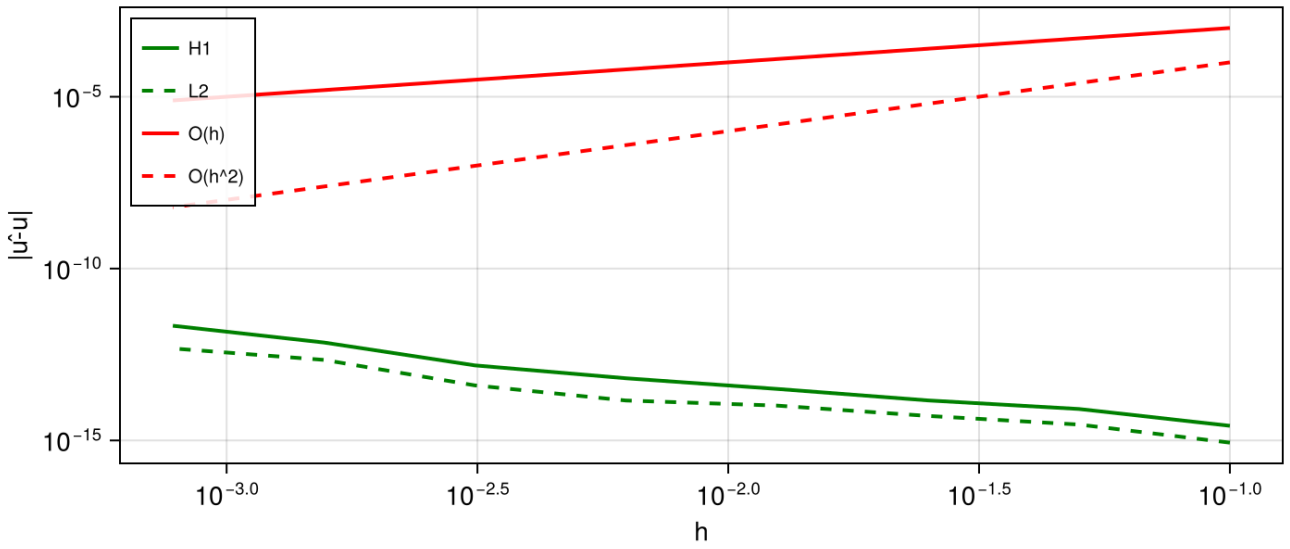
```
function system2(; h = 0.1, dim = 1)
    g = grid1(; h, dim)
    sys = VoronoiFVM.System(g; flux = flux2, reaction = rea2, bcondition = bc2,
species = [1], assembly)
    control = SolverControl(strategy, sys)
    u = solve(sys; control)
    return sys, g, u
end

```



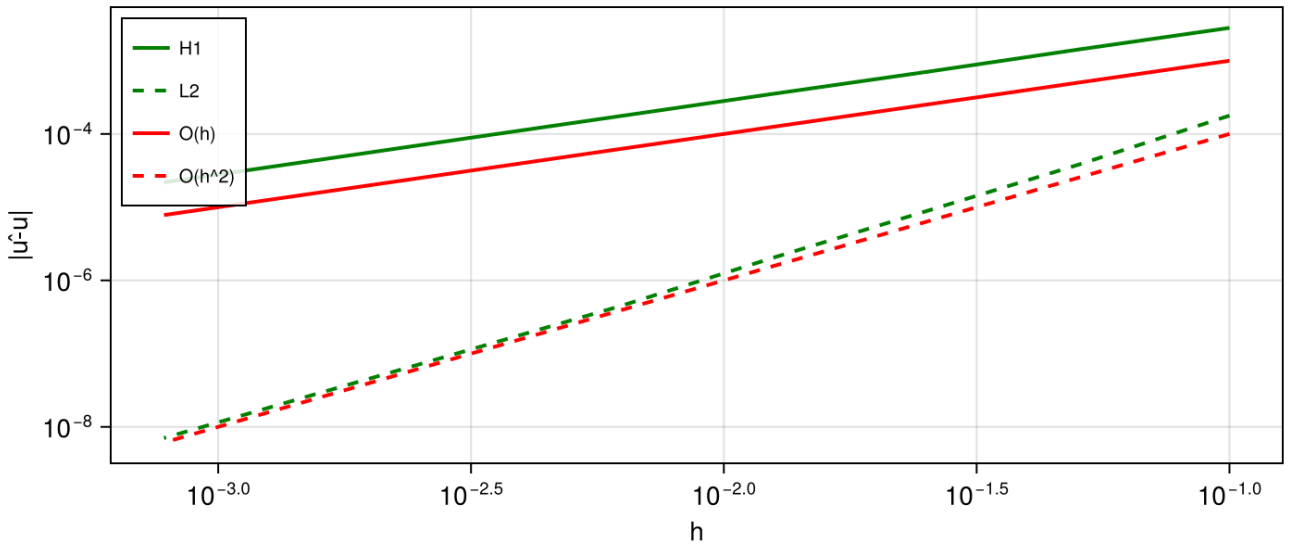

```
plot1d(system2, fexact2)
```




```
plot2d(system2, fexact2)
```




```
1 convtest(system2, fexact2, dim = 1)
```




```
1 convtest(system2, fexact2, dim = 2)
```

- Again, in 1D we have a special case where the scheme is exact.
- In 2D, the convergence order results are the same as for the diffusion case.

Convection-Diffusion

$$
\begin{aligned}
-\nabla \cdot (\nabla u + k u) &= 0 \\
u|_{x=-1} &= 1 \\
u|_{x=1} &= \hat{u}(1)
\end{aligned}
$$

Exact solution

$$\hat{u}(x) = \exp(-k(x+1))$$

flux3_upwind (generic function with 1 method)

```
function flux3_upwind(f, u, edge, data)
    vh = data.evelo[edge.index]
    fdiff = (u[1, 1] - u[1, 2])
    if vh > 0
        f[1] = fdiff + vh * u[1, 1]
    else
        f[1] = fdiff + vh * u[1, 2]
    end
    return nothing
end

```

flux3_sg (generic function with 1 method)

```
function flux3_sg(f, u, edge, data)
    vh = data.evelo[edge.index]
    Bplus = fbernoulli(vh)
    Bminus = fbernoulli(-vh)
    f[1] = Bminus * u[1, 1] - Bplus * u[1, 2]
    return nothing
end

```

flux3_central (generic function with 1 method)

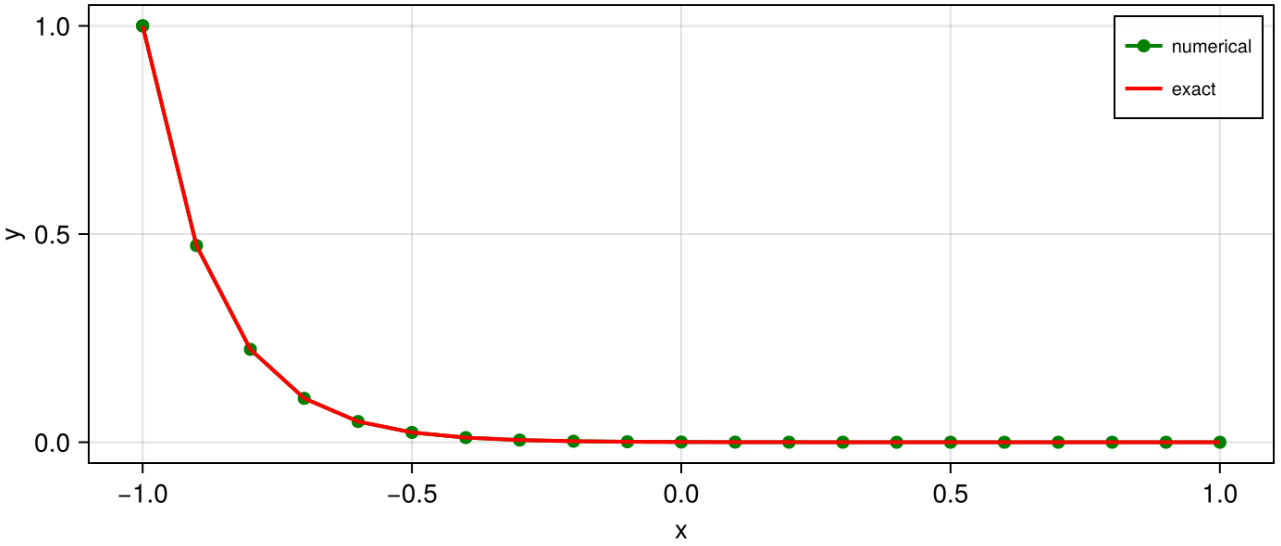
```
function flux3_central(f, u, edge, data)
    vh = data.evelo[edge.index]
    fdiff = (u[1, 1] - u[1, 2])
    f[1] = fdiff + 0.5 * vh * (u[1, 1] + u[1, 2])
    return nothing
end

```

bc3 (generic function with 1 method)

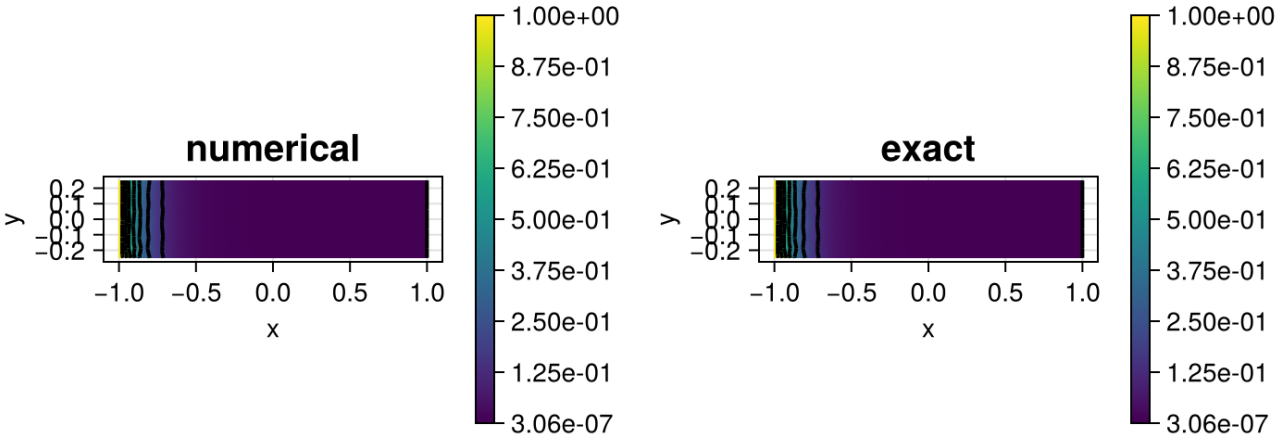
```
function bc3(f, u, bnode, data)
    boundary_dirichlet!(f, u, bnode, region = 1, value = 1)
    boundary_dirichlet!(f, u, bnode, region = 2, value = fexact2(1))
    return nothing
end

```

```
system3 (generic function with 1 method)
```

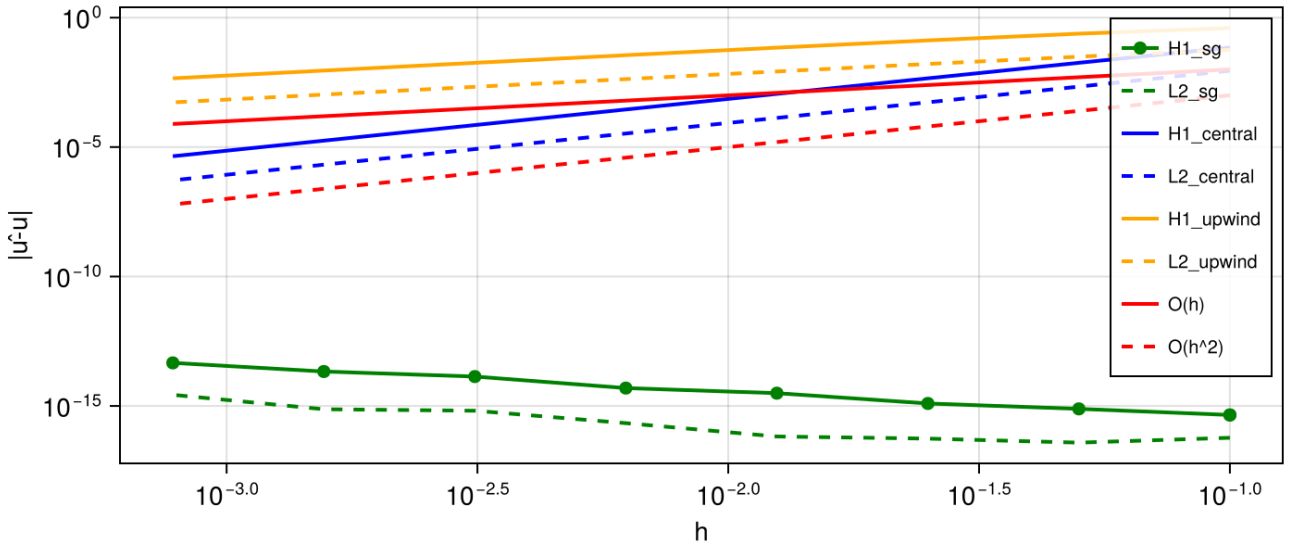
```
function system3(; h = 0.1, dim = 1, flux3 = flux3_sg)
    g = grid1(; h, dim)
    if dim == 1
        X = g[Coordinates][1, :]
        n = length(X)
        ev = [-k * (X[i + 1] - X[i]) for i in 1:(n - 1)]
    else
        ev = edgevelocities(g, (x, y) -> (k, 0))
    end
    data = (evelo = ev,)
    sys = VoronoiFVM.System(g; data, flux = flux3, bcondition = bc3, species = [1], assembly)
    control = SolverControl(strategy, sys)
    u = solve(sys)
    return sys, g, u
end
```



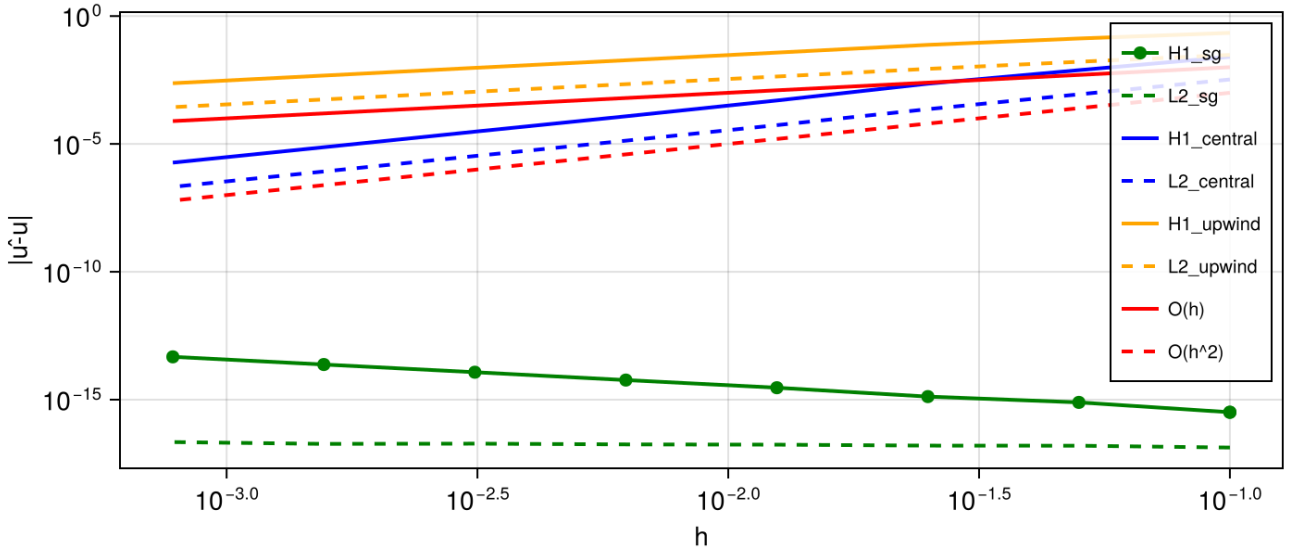

```
plot1d(system3, fexact2)
```




```
plot2d(system3, fexact2)
```




```
convtest_cd(system3, fexact2, dim = 1)
```




```
convtest_cd(system3, fexact2, dim = 2)
```

- For the 1D problems, the exponential fitting / Scharfetter-Gummel scheme is exact, this has to be expected as the flux is defined from exact solutions at the grid intervals which are glued together. The central difference scheme exhibits second order convergence for noth norms, and the upwind scheme is first order accurate for both norms.
- A similar qualitative picture is seen for the 2D case.

Convection-Diffusion-Reaction

$$
\begin{aligned}
-\nabla \cdot (\nabla u + \frac{k}{2} u) + \frac{k^2}{2} u &= 0 \\
u|_{x=-1} &= 1 \\
u|_{x=1} &= \hat{u}(1)
\end{aligned}
$$

Exact solution:

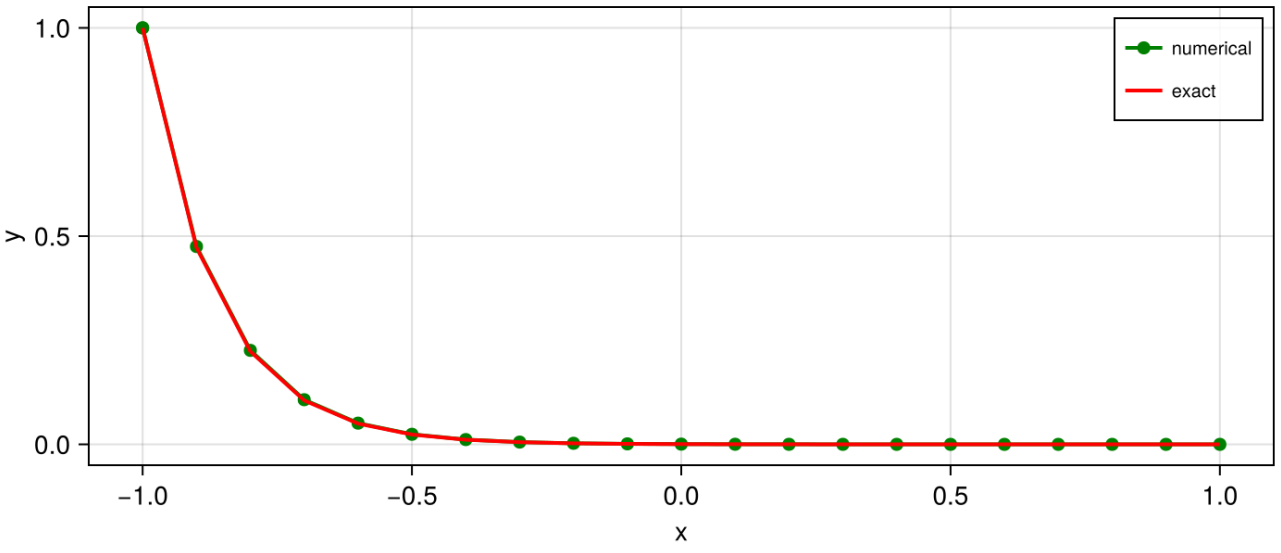
$$\hat{u}(x) = \exp(-k(x+1))$$

```
rea4 (generic function with 1 method)
```

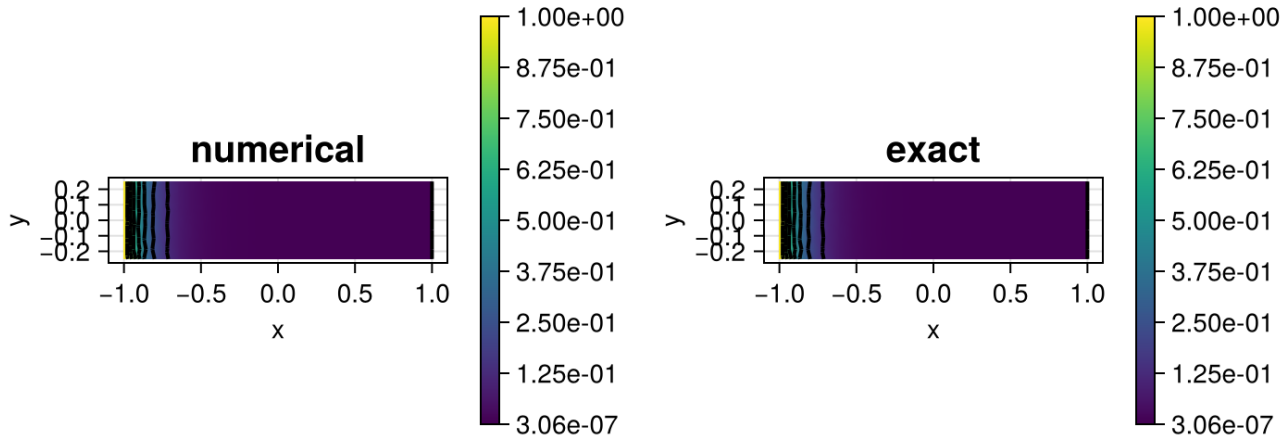
```
function rea4(f, u, node, data)
    f[1] = 0.5 * k^2 * u[1]
    return nothing
end
```

```
system4 (generic function with 1 method)
```

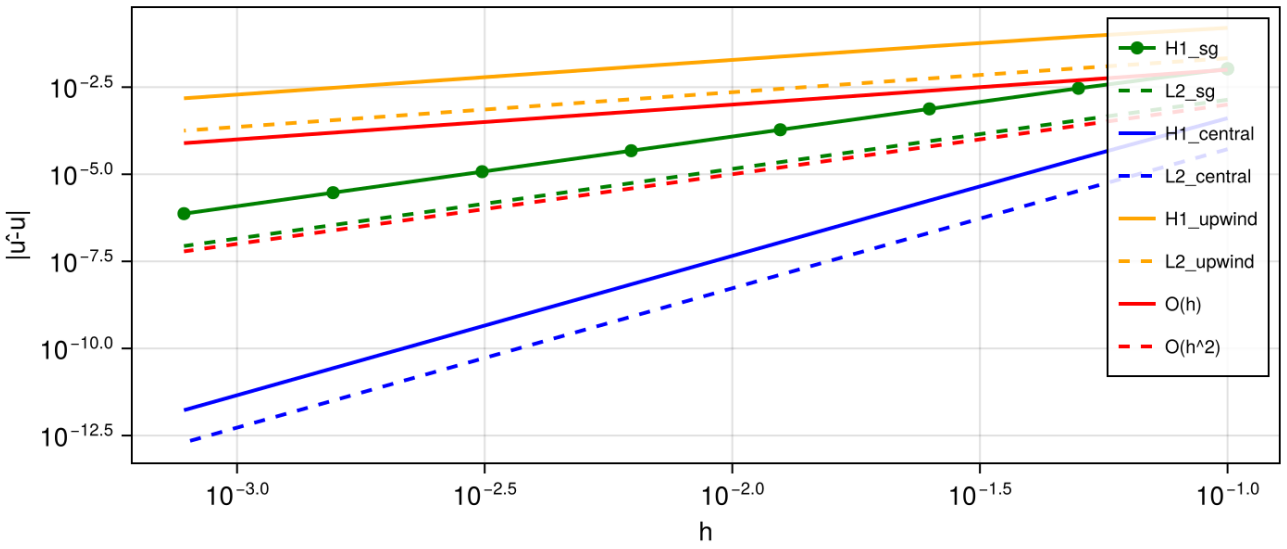
```
function system4(; h = 0.1, dim = 1, flux3 = flux3_sg)
    g = grid1(; h, dim)
    if dim == 1
        X = g[Coordinates][1, :]
        n = length(X)
        ev = [-k * (X[i + 1] - X[i]) / 2 for i in 1:(n - 1)]
    else
        ev = edgevelocities(g, (x, y) -> (k / 2, 0))
    end
    data = (evelo = ev,)
    sys = VoronoiFVM.System(g; data, flux = flux3, reaction = rea4, bcondition = bc3, species = [1], assembly)
    control = SolverControl(strategy, sys)
    u = solve(sys)
    return sys, g, u
end

```



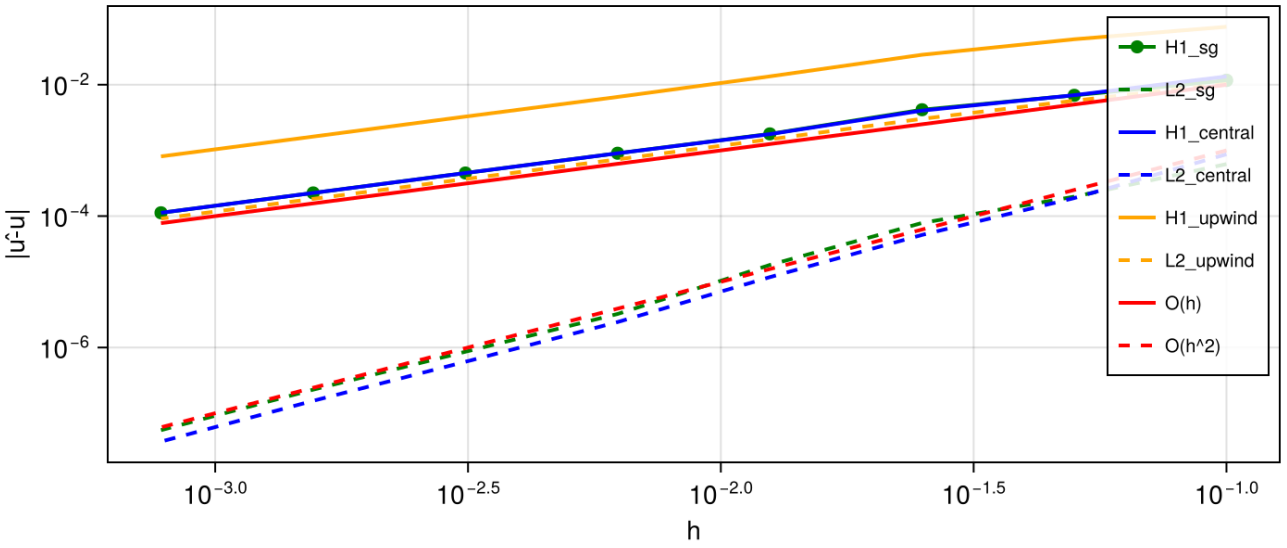

```
plot1d(system4, fexact2)
```




```
1 plot2d(system4, fexact2)
```




```
1 convtest_cd(system4, fexact2, dim = 1)
```




```
1 convtest_cd(system4, fexact2, dim = 2)
```

- For 1D problems, the Scharfetter-Gummel scheme is second order convergent in both

norms, the simple upwind scheme is first order convergent. The central scheme provides higher than second order convergence.

- In 2D, both central differences and the Scharfetter-Gummel scheme are fitst order convergent in the H^1-seminorm and second order convergent in the L^2 norm. The simple upwind scheme is first order convergent in both norms.

Helpers

grid1 (generic function with 1 method)

```
function grid1(; h = 0.1, dim = 1, x0 = -1, x1 = 1, y0 = -0.25, y1 = 0.25)
    if dim == 1
        return simplexgrid(x0:h:x1)
    else
        b = SimplexGridBuilder(Generator = Triangulate)
        p00 = point!(b, x0, y0)
        p10 = point!(b, x1, y0)
        p11 = point!(b, x1, y1)
        p01 = point!(b, x0, y1)
        facetregion!(b, 3)
        facet!(b, p00, p10)
        facet!(b, p01, p11)
        facetregion!(b, 1)
        facet!(b, p00, p01)
        facetregion!(b, 2)
        facet!(b, p10, p11)
        return simplexgrid(b, maxvolume = h^2 / 2)
    end
    return nothing
end
```

plot1d (generic function with 1 method)

```
function plot1d(system, fexact)
    s, g, u = system(dim = 1)
    uex = map(fexact, g)
    vis = GridVisualizer(size = (700, 300), legend = :rt)
    scalarplot!(vis, g, u[1, :], color = :green, markershape = :circle, markersize = 10, markevery = 1, label = "numerical")
    scalarplot!(vis, g, uex, color = :red, clear = false, markershape = :none, label = "exact")
    return reveal(vis)
end
```

```
plot2d (generic function with 1 method)
```

```
function plot2d(system, fexact)
    s, g, u = system(dim = 2)
    uex = map(fexact, g)
    vis = GridVisualizer(size = (700, 300), layout = (1, 2))
    scalarplot!(vis[1, 1], g, u[1, :], color = :green, markershape = :circle, markersize = 10, markevery = 1, title = "numerical")
    scalarplot!(vis[1, 2], g, uex, color = :red, clear = false, markershape = :none, title = "exact")
    return reveal(vis)
end
```

```
convtest (generic function with 1 method)
```

```
function convtest(system, fexact; dim = 2)
    H = [0.1 * 2.0^(-i) for i in 0:Nref]
    L2 = Float64[]
    H1 = Float64[]
    for h in H
        sys, g, u = system1(; h, dim)
        uex = unknowns(sys)
        uex[1, :] .= map(fexact1, g)
        push!(H1, h1norm(sys, u - uex))
        push!(L2, l2norm(sys, u - uex))
    end
    vis = GridVisualizer(
        size = (700, 300), xscale = :log, yscale = :log, legend = :lt,
        xlabel = "h", ylabel = "|u-û|"
    )
    scalarplot!(vis, H, H1, color = :green, label = "H1")
    scalarplot!(vis, H, L2, color = :green, label = "L2", clear = false, linestyle = :dash)
    scalarplot!(vis, H, H * 1.0e-2, color = :red, linestyle = :solid, label = "O(h)", clear = false)
    scalarplot!(vis, H, H .^ 2 * 1.0e-2, color = :red, linestyle = :dash, label = "O(h^2)", clear = false)
    return reveal(vis)
end
```

convtest_cd (generic function with 1 method)

```
function convtest_cd(system, fexact; dim = 2)
    H = [0.1 * 2.0^(-i) for i in 0:Nref]
    L2_sg = Float64[]
    H1_sg = Float64[]
    L2_upwind = Float64[]
    H1_upwind = Float64[]
    L2_central = Float64[]
    H1_central = Float64[]
    for h in H
        sys, g, u = system(; h, dim, flux3 = flux3_sg)
        uex = unknowns(sys)
        uex[1, :] .= map(fexact, g)
        push!(H1_sg, h1norm(sys, u - uex))
        push!(L2_sg, l2norm(sys, u - uex))
        sys, g, u = system(; h, dim, flux3 = flux3_upwind)
        push!(H1_upwind, h1norm(sys, u - uex))
        push!(L2_upwind, l2norm(sys, u - uex))
        sys, g, u = system(; h, dim, flux3 = flux3_central)
        push!(H1_central, h1norm(sys, u - uex))
        push!(L2_central, l2norm(sys, u - uex))
    end
    vis = GridVisualizer(
        size = (700, 300), xscale = :log, yscale = :log, legend = :rt,
        xlabel = "h", ylabel = "|u-û|"
    )
    scalarplot!(vis, H, H1_sg, color = :green, label = "H1_sg", markershape = :circle, markersize = 10, markevery = 1)
    scalarplot!(vis, H, L2_sg, color = :green, label = "L2_sg", clear = false, markershape = :none, linestyle = :dash)
    scalarplot!(vis, H, H1_central, color = :blue, label = "H1_central", clear = false, linestyle = :solid)
    scalarplot!(vis, H, L2_central, color = :blue, label = "L2_central", clear = false, linestyle = :dash)
    scalarplot!(vis, H, H1_upwind, color = :orange, label = "H1_upwind", clear = false, linestyle = :solid)
    scalarplot!(vis, H, L2_upwind, color = :orange, label = "L2_upwind", clear = false, linestyle = :dash)
    scalarplot!(vis, H, H * 1.0e-1, color = :red, linestyle = :solid, label = "O(h)", clear = false)
    scalarplot!(vis, H, H .^ 2 * 1.0e-1, color = :red, linestyle = :dash, label = "O(h^2)", clear = false)
    return reveal(vis)
end
```