

## Advanced Topics from Scientific Computing

TU Berlin Winter 2023/24

### Notebook 16

 Jürgen Fuhrmann

CairoMakie

```
1 begin
2     using PlutoUI
3     using VoronoiFVM
4     using SimplexGridFactory
5     using Triangulate
6     using LinearSolve, AMGCLWrap
7     using ExtendableGrids
8     import CairoMakie
9     using GridVisualize
10    CairoMakie.activate!(type="svg")
11    default_plotter!(CairoMakie)
12 end
```

## Table of Contents

### VoronoiFVM.jl Convergence Tests

Settings

Plots of coarsest grids

Parameters

Diffusion

Reaction-Diffusion

Convection-Diffusion

Convection-Diffusion-Reaction

Helpers

# VoronoiFVM.jl Convergence Tests

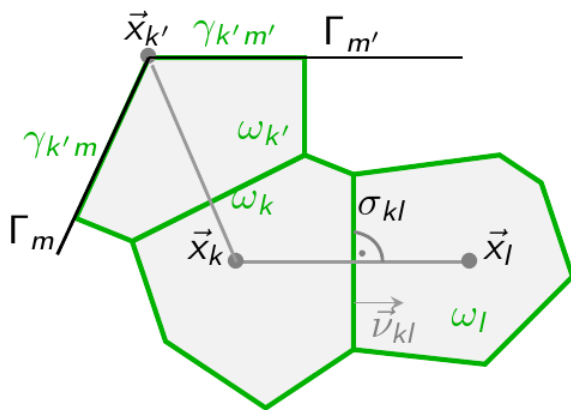
---

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 nonlinear 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  $\Pi_h : C(\Omega) \rightarrow \mathbb{R}^n$  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

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

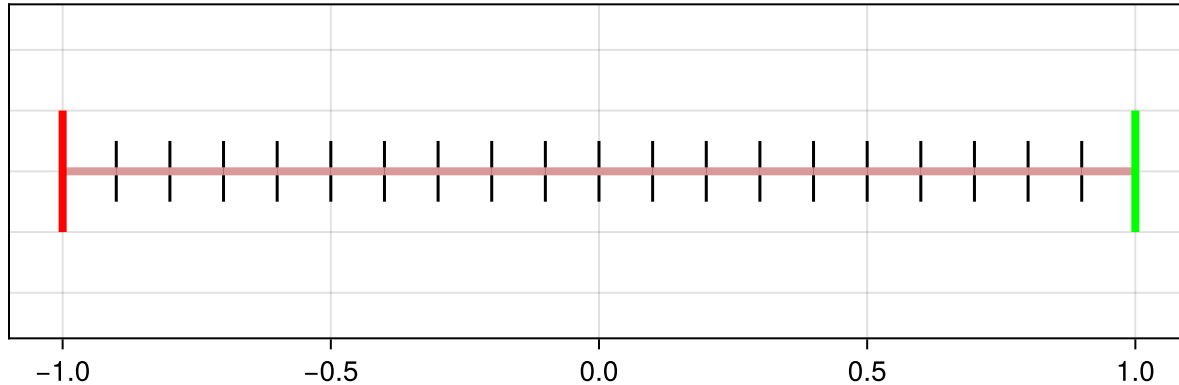
The typical converge estimates from the P1 finite element method for  $H^2$ -regular problems give

$$\begin{aligned} \|u_h - \hat{u}_h\|_0 &= O(h^2) \quad (h \rightarrow 0) \\ |u_h - \hat{u}_h|_1 &= O(h) \quad (h \rightarrow 0) \end{aligned}$$

# Settings

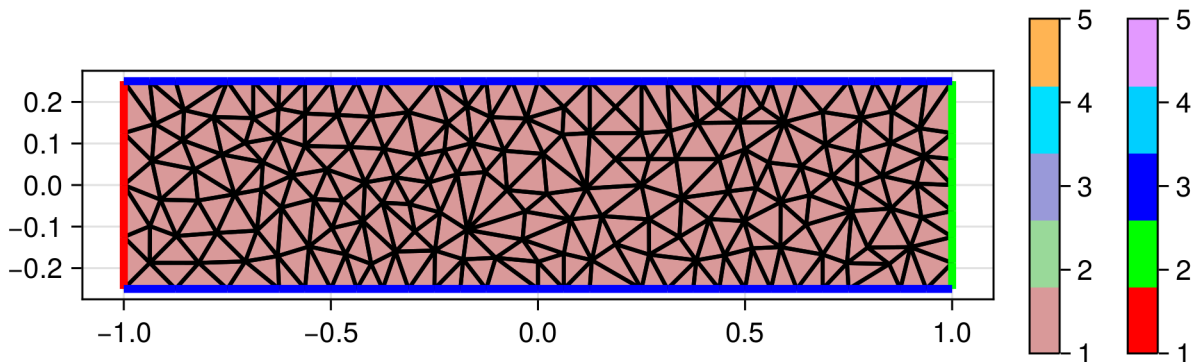
## Plots of coarsest grids

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, { "tvne": '

```

```
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)

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

flux1 (generic function with 1 method)

```
1 function flux1(f,u,enge)
2   f[1]=u[1,1]- u[1,2]
3 end
```

src1 (generic function with 1 method)

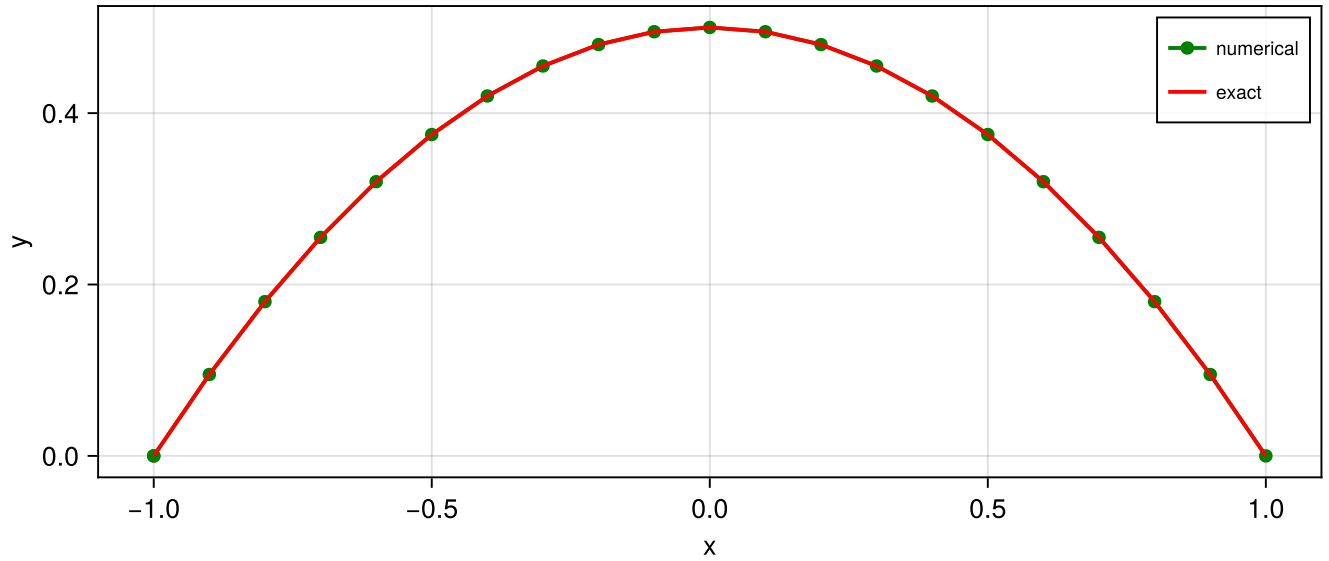
```
1 function src1(f,node)
2   f[1]=1
3 end
```

bc1 (generic function with 1 method)

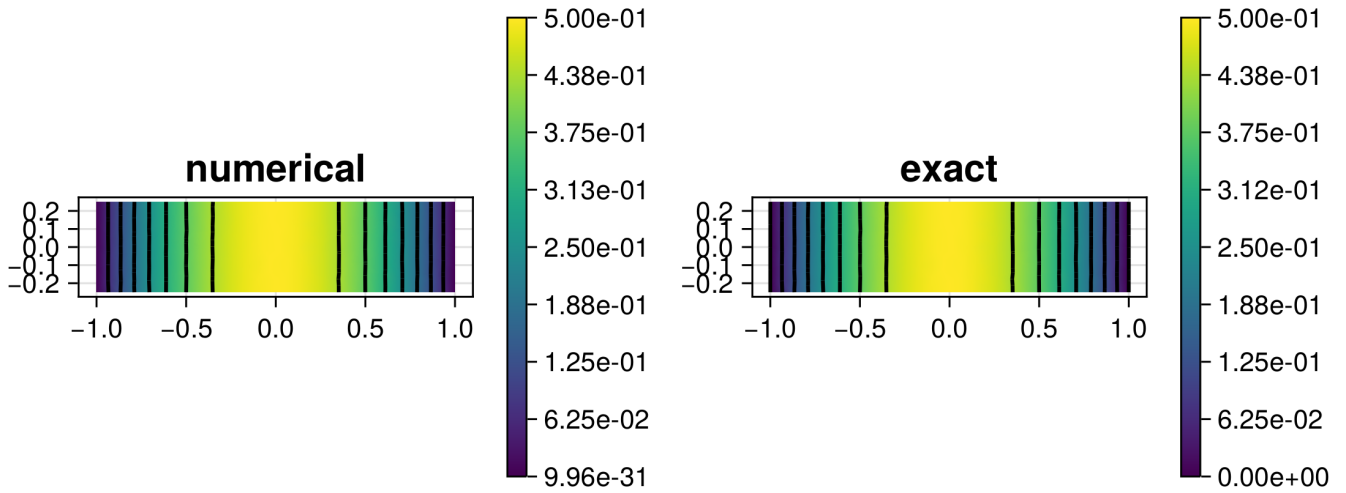
```
1 function bc1(f,u,bnode)
2   boundary_dirichlet!(f,u,bnode,region=1,value=0)
3   boundary_dirichlet!(f,u,bnode,region=2,value=0)
4 end
```

system1 (generic function with 1 method)

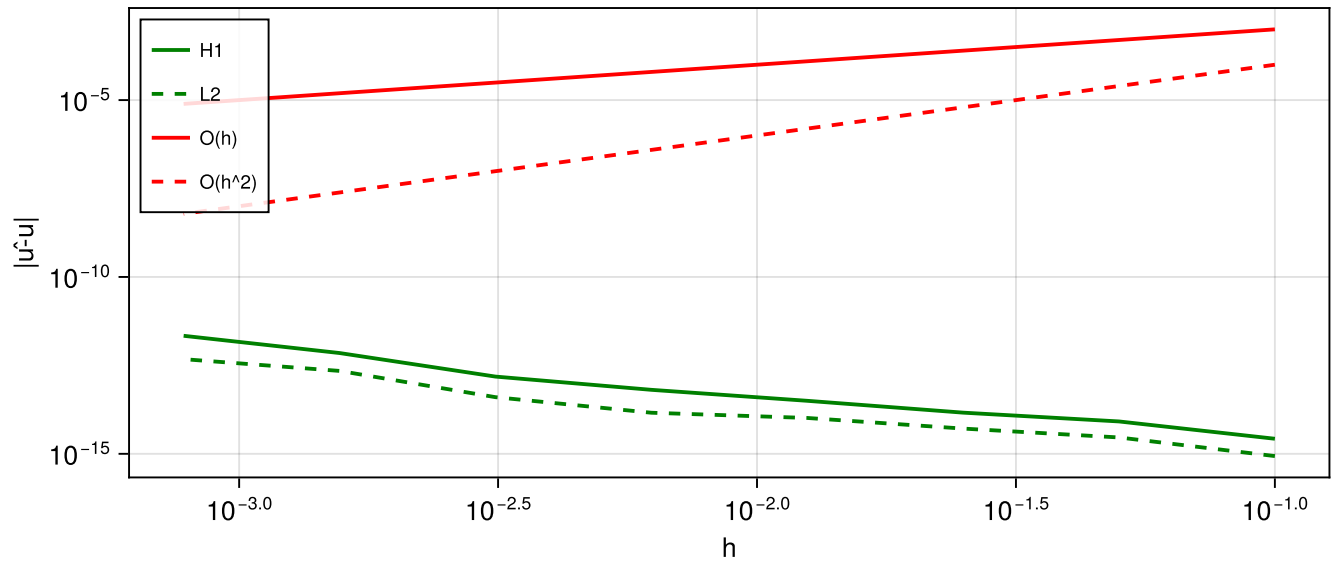
```
1 function system1(;h=0.1,dim=1)
2   g=grid1(;h,dim)
3   sys=VoronoiFVM.System(g;flux=flux1,source=src1,bcondition=bc1,species=
4   [1],assembly)
5   control=SolverControl(strategy,sys)
6   u=solve(sys; control)
7   sys,g,u
end
```



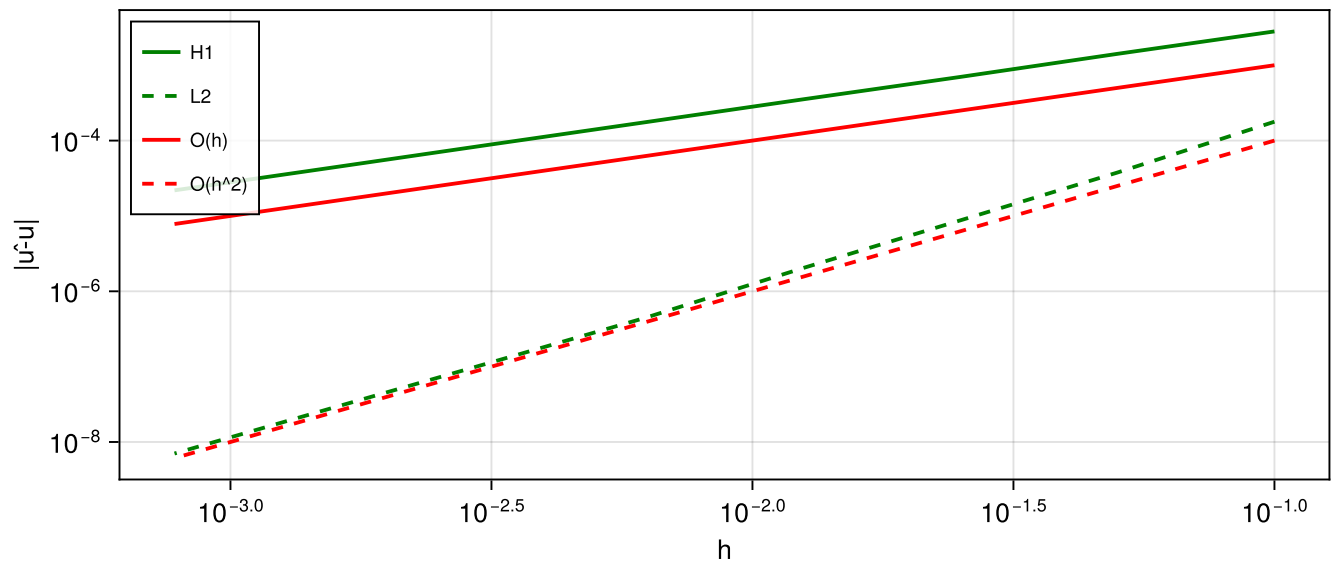
```
1 plot1d(system1, fexact1)
```



```
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)

```

1 begin
2   fexact2(x)= exp(-k*(x+1))
3   fexact2(x,y)= fexact2(x)
4 end

```

flux2 (generic function with 1 method)

```

1 function flux2(f,u,edge)
2   f[1]=u[1,1]- u[1,2]
3 end
4

```

rea2 (generic function with 1 method)

```

1 function rea2(f,u,node)
2   f[1]=k^2*u[1]
3 end
4

```

bc2 (generic function with 1 method)

```

1 function bc2(f,u,bnode)
2   boundary_dirichlet!(f,u,bnode,region=1,value=1)
3   boundary_dirichlet!(f,u,bnode,region=2,value=fexact2(1))
4 end
5

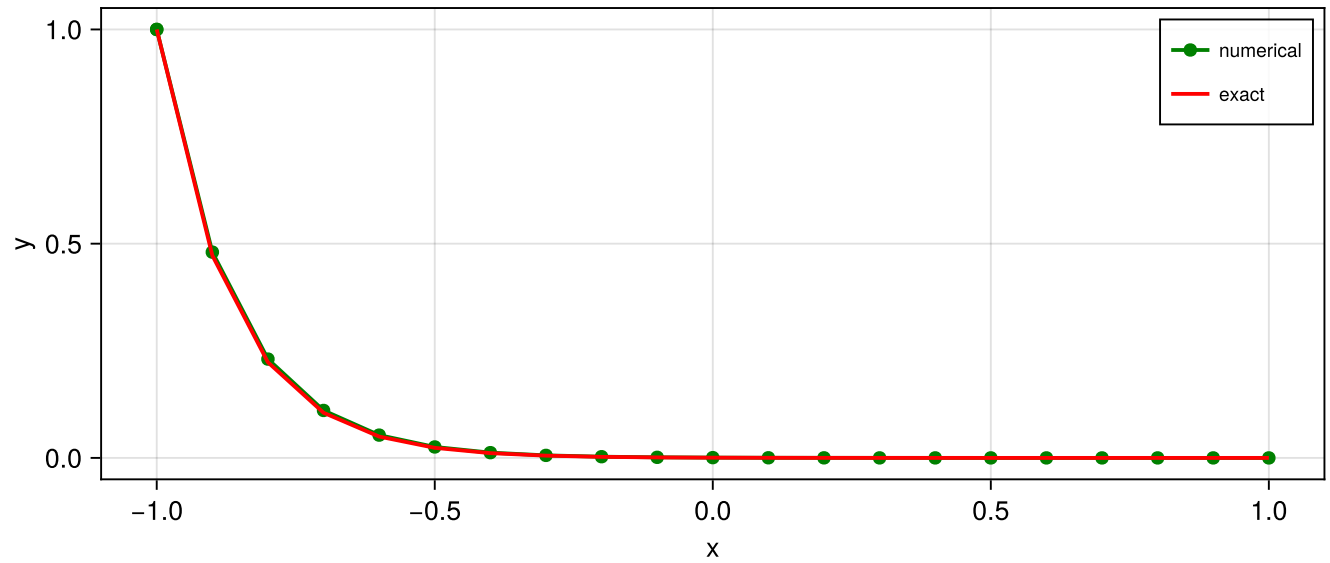
```

system2 (generic function with 1 method)

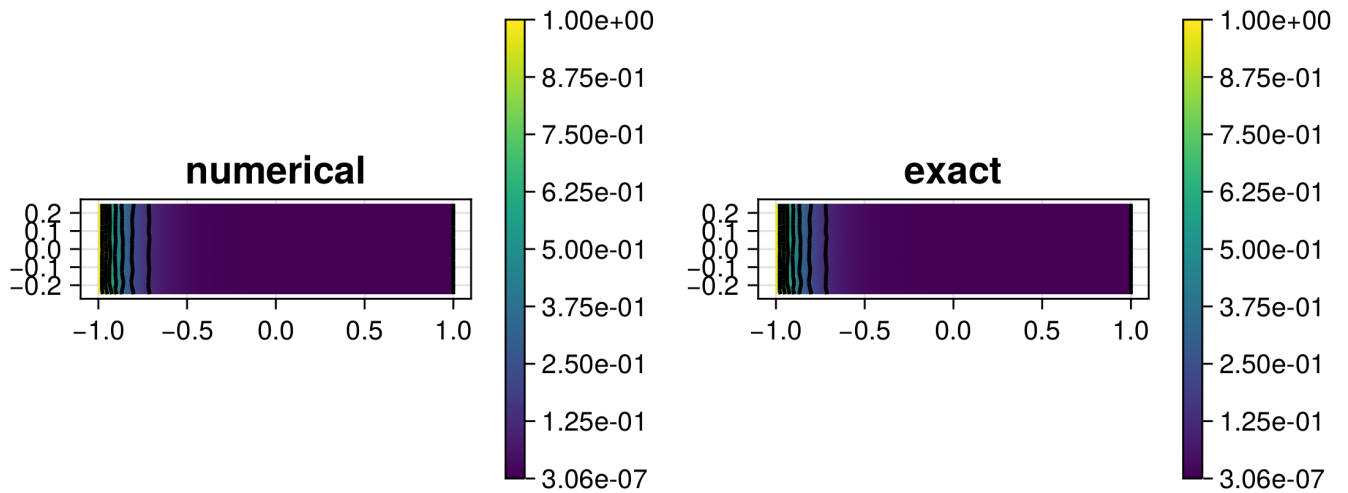
```

1 function system2(;h=0.1,dim=1)
2   g=grid1(;h,dim)
3   sys=VoronoiFVM.System(g;flux=flux2,reaction=rea2,bcondition=bc2,species=
4   [1],assembly)
5   control=SolverControl(strategy,sys)
6   u=solve(sys; control)
7   sys,g,u
8 end

```

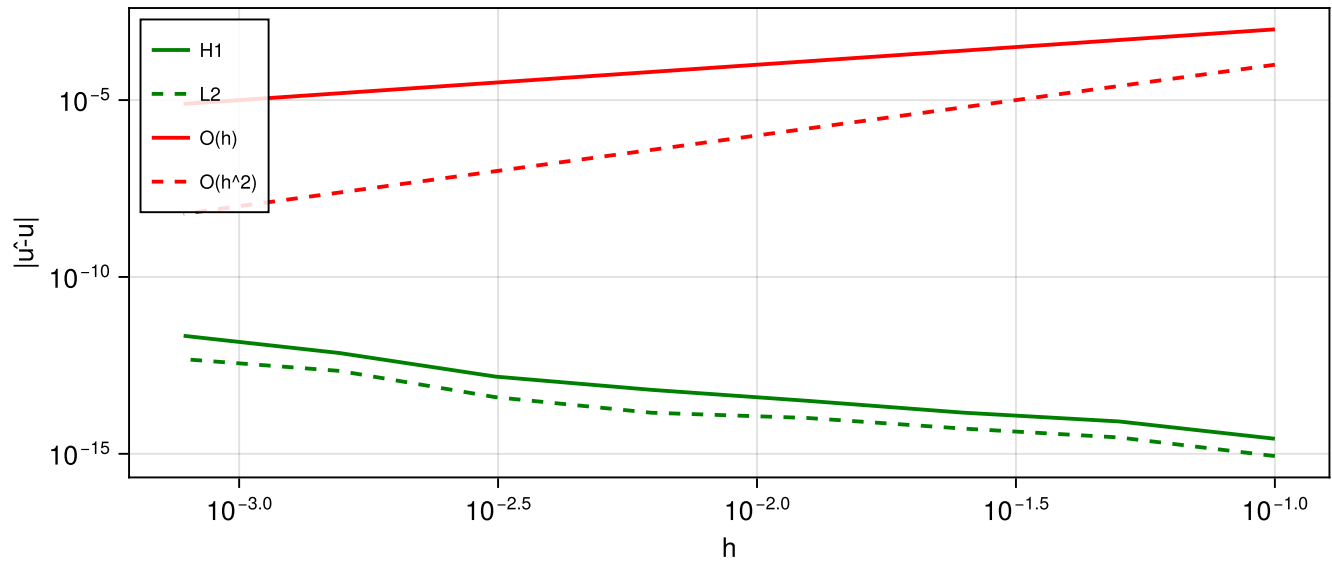


```
1 plot1d(system2, fexact2)
```

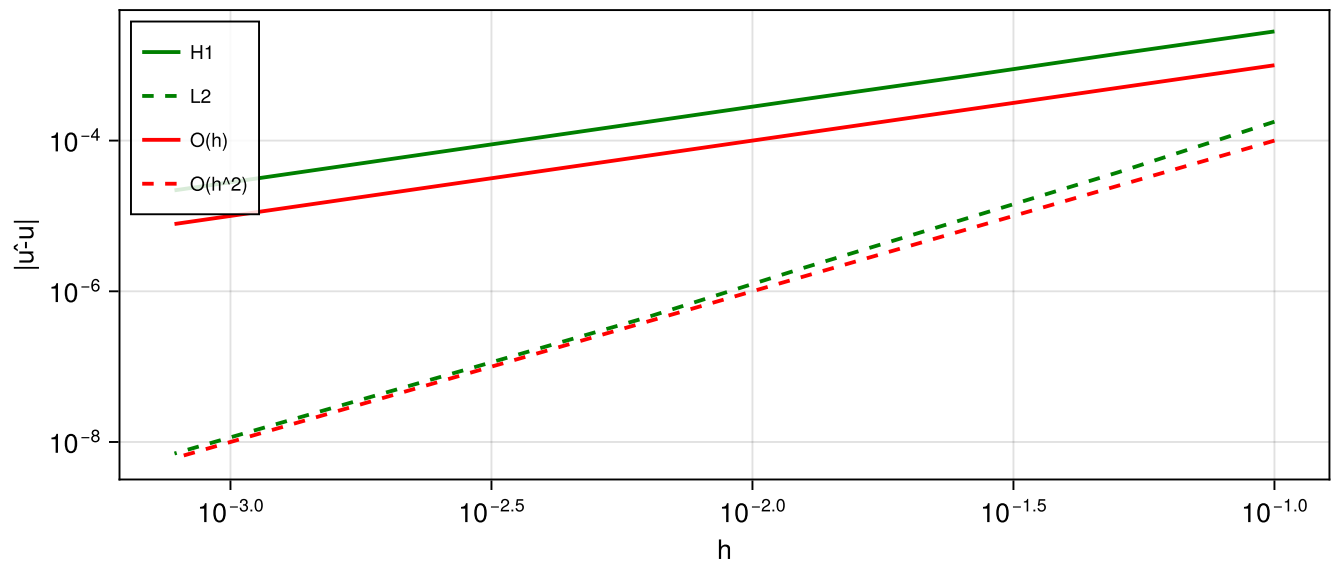


```
1 plot2d(system2, fexact2)
```





```
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

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

Exact solution

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

flux3\_upwind (generic function with 1 method)

```

1 function flux3_upwind(f,u,edge,data)
2   vh=data.avelo[edge.index]
3   fdiff = (u[1,1] - u[1, 2])
4   if vh > 0
5     f[1] = fdiff + vh * u[1, 1]
6   else
7     f[1] = fdiff + vh * u[1, 2]
8   end
9 end
10

```

flux3\_sg (generic function with 1 method)

```

1 function flux3_sg(f,u,edge,data)
2   vh=data.avelo[edge.index]
3   Bplus = fbernoulli(vh)
4   Bminus = fbernoulli(-vh)
5   f[1] = Bminus * u[1, 1] - Bplus * u[1, 2]
6 end
7

```

flux3\_central (generic function with 1 method)

```

1 function flux3_central(f,u,edge,data)
2   vh=data.avelo[edge.index]
3   fdiff = (u[1,1] - u[1, 2])
4   f[1]=fdiff+ 0.5*vh*(u[1,1] + u[1, 2])
5 end
6

```

bc3 (generic function with 1 method)

```

1 function bc3(f,u,bnode,data)
2   boundary_dirichlet!(f,u,bnode,region=1,value=1)
3   boundary_dirichlet!(f,u,bnode,region=2,value=fexact2(1))
4 end
5

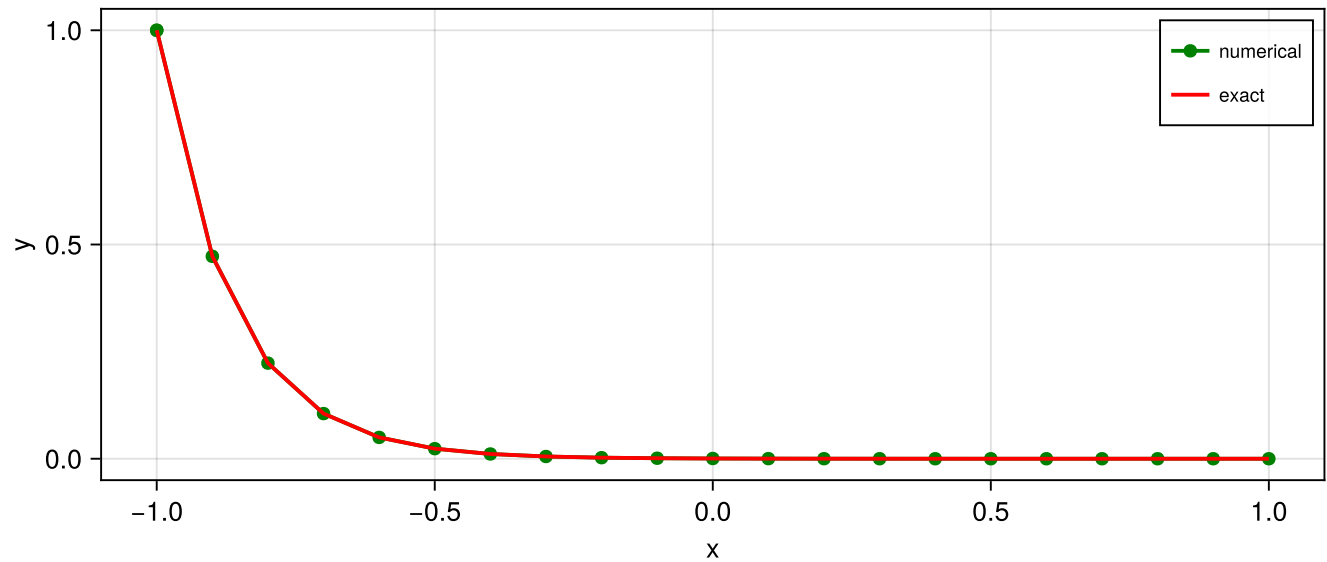
```

system3 (generic function with 1 method)

```

1 function system3(;h=0.1,dim=1,flux3=flux3_sg)
2   g=grid1(;h,dim)
3   if dim==1
4     X=g[Coordinates][1,:]
5     n=length(X)
6     ev=[-k*(X[i+1]-X[i]) for i=1:n-1]
7   else
8     ev=edgevelocities(g, (x,y)->(k,0))
9   end
10  data=(evelo=ev,)
11  sys=VoronoiFVM.System(g;data,flux=flux3,bcondition=bc3,species=[1],assembly)
12  control=SolverControl(strategy,sys)
13  u=solve(sys)
14  sys,g,u
15 end

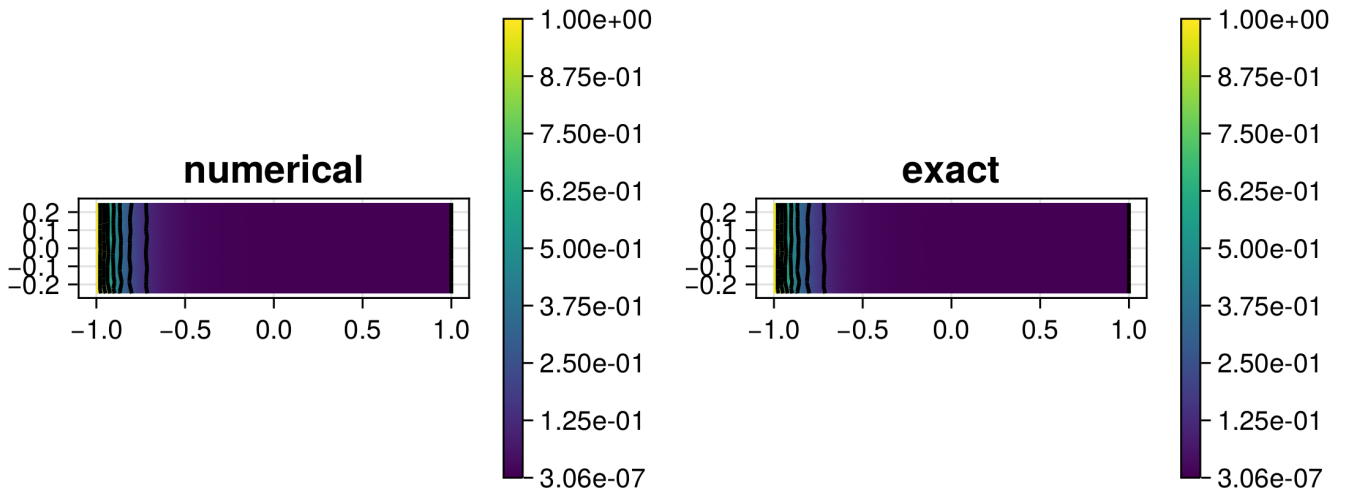
```



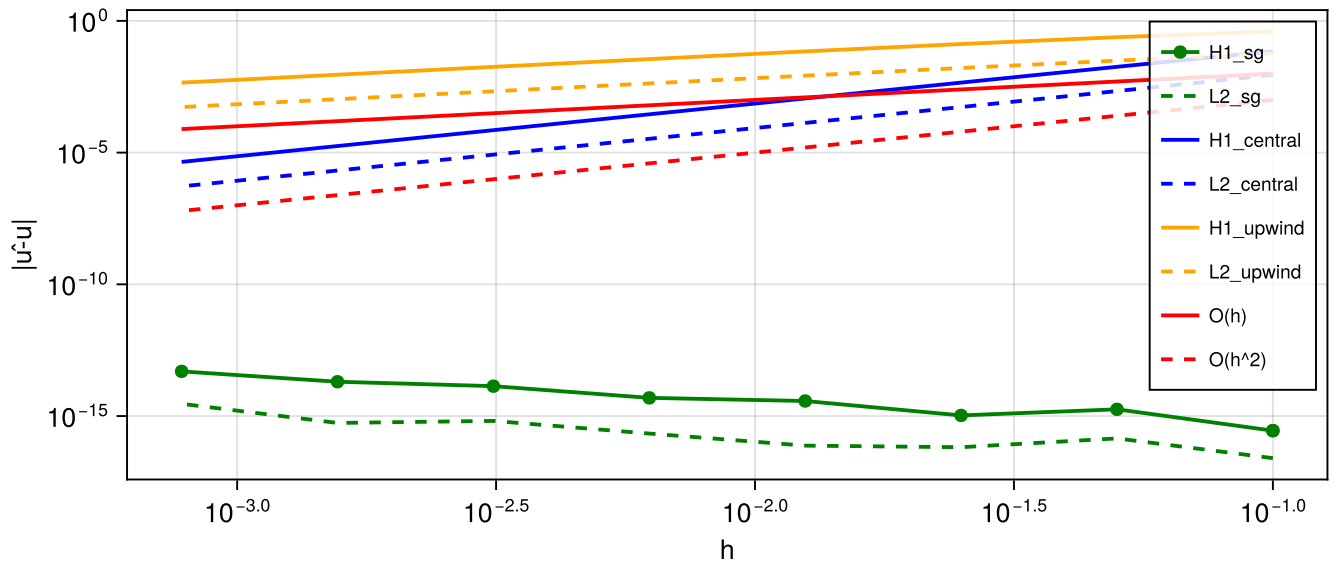
```

1 plot1d(system3,fexact2)

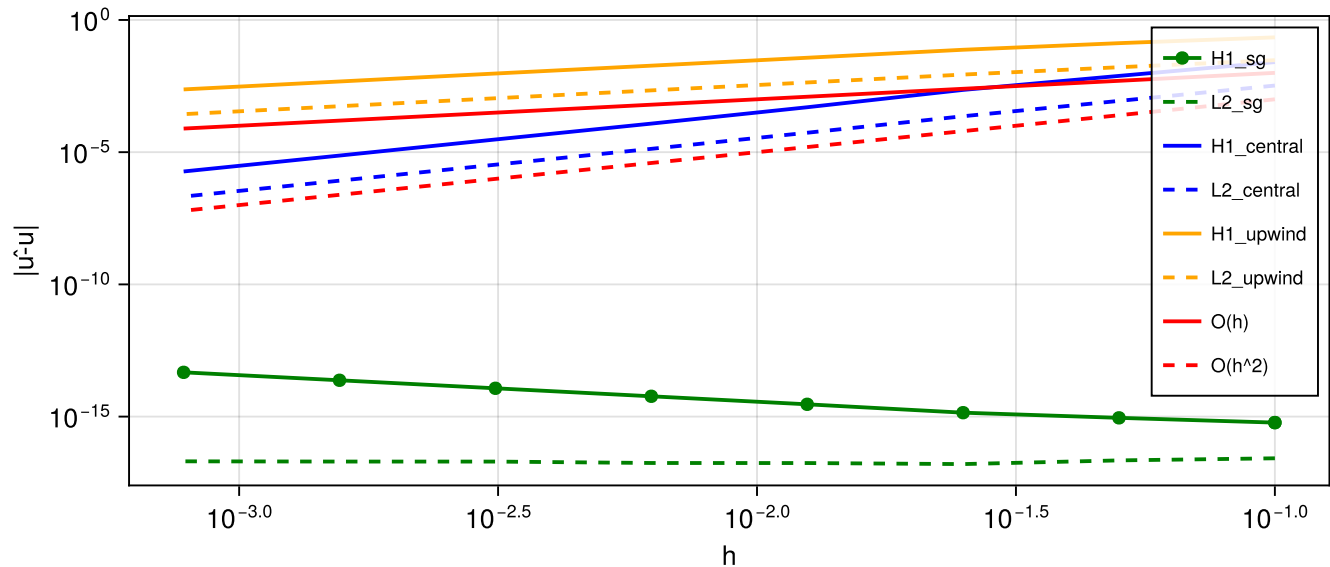
```



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



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



```
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 both 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

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

Exact solution:

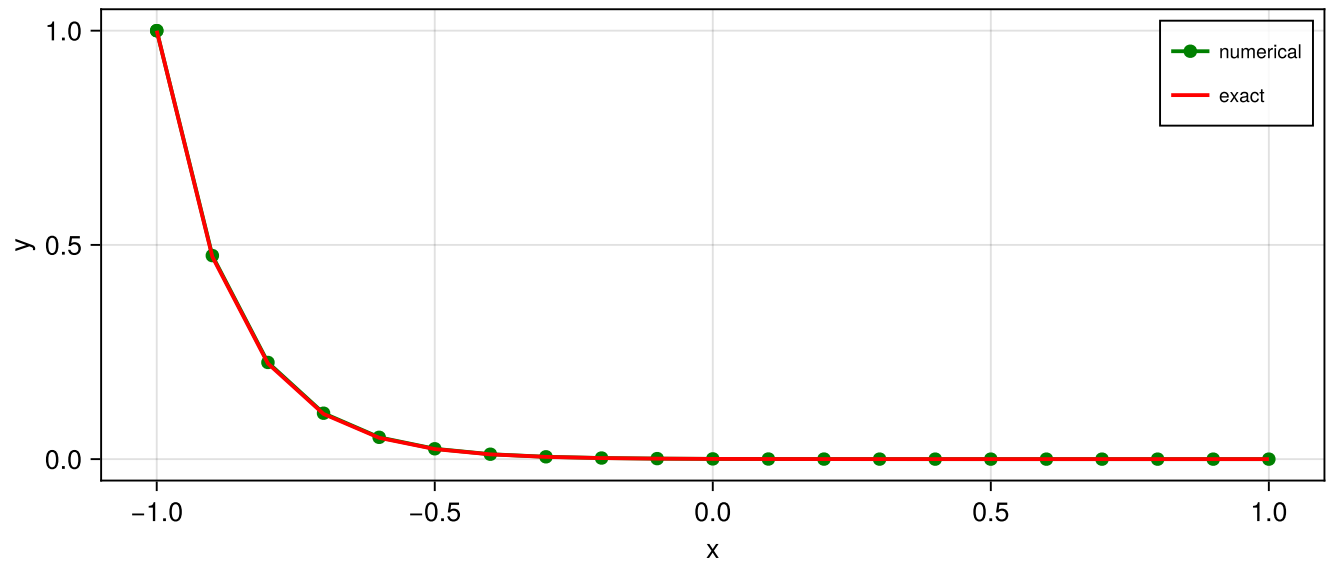
$$\hat{u}(x) = \exp(-k(x+1))$$

rea4 (generic function with 1 method)

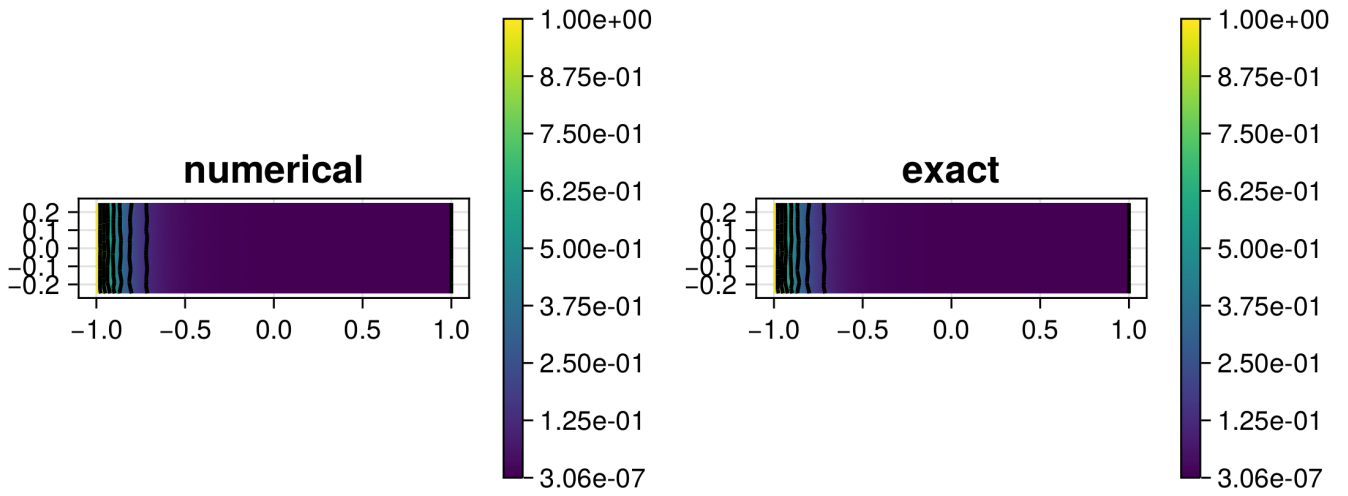
```
1 function rea4(f,u,node,data)
2   f[1]=0.5*k^2*u[1]
3 end
```

system4 (generic function with 1 method)

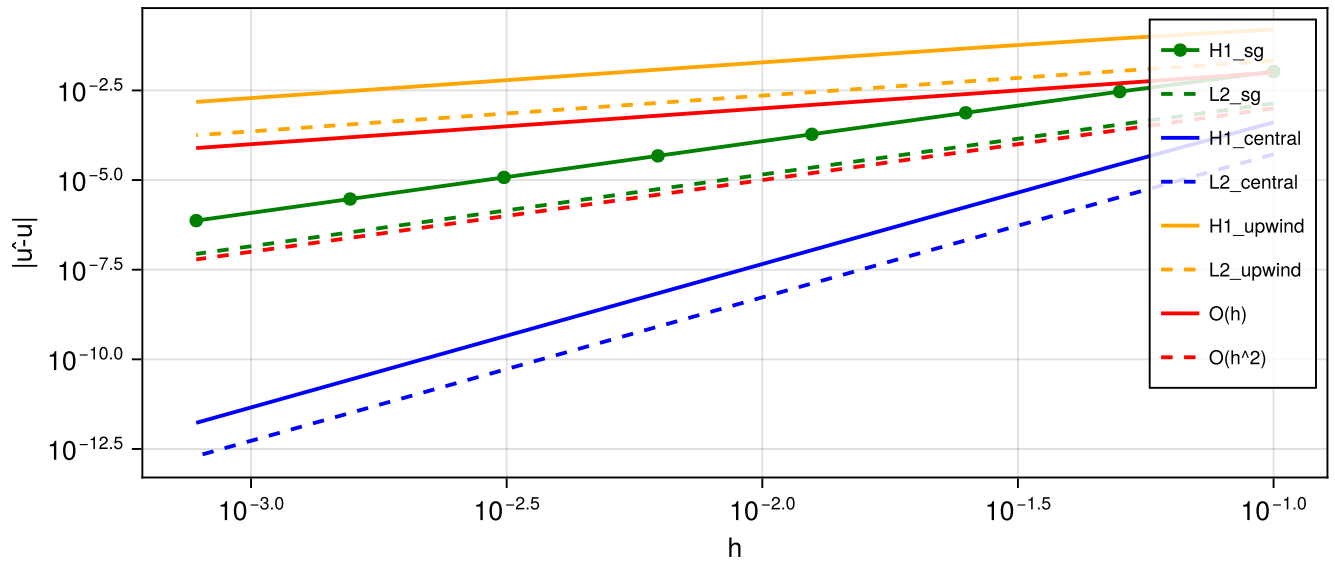
```
1 function system4(;h=0.1,dim=1,flux3=flux3_sg)
2   g=grid1(;h,dim)
3   if dim==1
4     X=g[Coordinates][1,:]
5     n=length(X)
6     ev=[-k*(X[i+1]-X[i])/2 for i=1:n-1]
7   else
8     ev=edgevelocities(g, (x,y)->(k/2,0))
9   end
10  data=(evelo=ev,)
11  sys=VoronoiFVM.System(g;data,flux=flux3,reaction=rea4,bcondition=bc3,species=
12    [1],assembly)
13  control=SolverControl(strategy,sys)
14  u=solve(sys)
15  sys,g,u
16 end
```



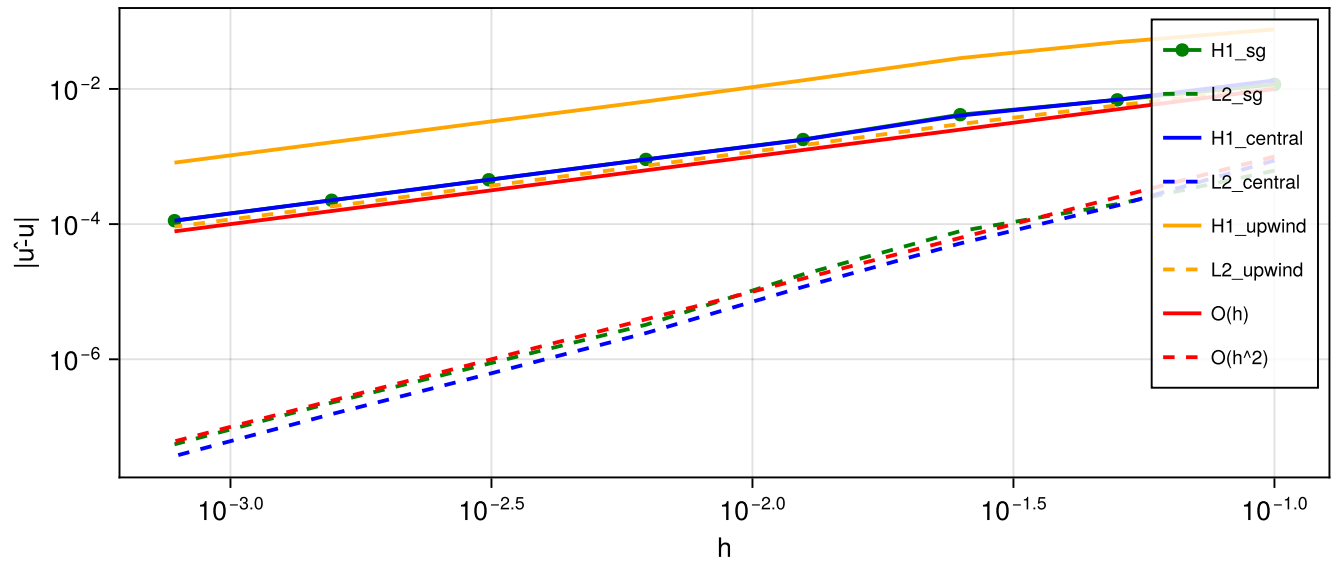
```
1 plot1d(system4,fexact2)
```



```
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 first 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)

```

1 function grid1(;h=0.1,dim=1,x0=-1,x1=1,y0=-0.25,y1=0.25)
2   if dim==1
3     simplexgrid(x0:h:x1)
4   else
5     b=SimplexGridBuilder(Generator=Triangulate)
6     p00=point!(b,x0,y0)
7     p10=point!(b,x1,y0)
8     p11=point!(b,x1,y1)
9     p01=point!(b,x0,y1)
10    facetregion!(b,3)
11    facet!(b,p00,p10)
12    facet!(b,p01,p11)
13    facetregion!(b,1)
14    facet!(b,p00,p01)
15    facetregion!(b,2)
16    facet!(b,p10,p11)
17    simplexgrid(b, maxvolume=h^2/2)
18  end
19 end

```

plot1d (generic function with 1 method)

```

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

```

plot2d (generic function with 1 method)

```

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

```

convtest (generic function with 1 method)

```
1 function convtest(system,fexact;dim=2)
2   H=[0.1*2.0^(-i) for i=0:Nref]
3   L2=Float64[]
4   H1=Float64[]
5   for h in H
6     sys,g,u=system1(;h,dim)
7     uex=unknowns(sys)
8     uex[1,:].=map(fexact,g)
9     push!(H1,h1norm(sys,u-uex))
10    push!(L2,l2norm(sys,u-uex))
11  end
12  vis=GridVisualizer(size=(700,300),xscale=:log,yscale=:log, legend=:lt,
13  xlabel="h", ylabel="|u-û|")
14  scalarplot!(vis,H,H1, color=:green, label="H1")
15  scalarplot!(vis,H,L2, color=:green,label="L2",clear=false, linestyle=:dash)
16  scalarplot!(vis,H,H*1.0e-2, color=:red, linestyle=:solid,
17  label="O(h)",clear=false)
18  scalarplot!(vis,H,H.^2*1.0e-2, color=:red, linestyle=:dash,
19  label="O(h^2)",clear=false)
20  reveal(vis)
21 end
```

convtest\_cd (generic function with 1 method)

```

1 function convtest_cd(system,fexact;dim=2)
2   H=[0.1*2.0^(-i) for i=0:Nref]
3   L2_sg=Float64[]
4   H1_sg=Float64[]
5   L2_upwind=Float64[]
6   H1_upwind=Float64[]
7   L2_central=Float64[]
8   H1_central=Float64[]
9   for h in H
10      sys,g,u=system(;h,dim, flux3=flux3_sg)
11      uex=unknowns(sys)
12      uex[1,:].=map(fexact,g)
13      push!(H1_sg,h1norm(sys,u-uex))
14      push!(L2_sg,l2norm(sys,u-uex))
15      sys,g,u=system(;h,dim, flux3=flux3_upwind)
16      push!(H1_upwind,h1norm(sys,u-uex))
17      push!(L2_upwind,l2norm(sys,u-uex))
18      sys,g,u=system(;h,dim, flux3=flux3_central)
19      push!(H1_central,h1norm(sys,u-uex))
20      push!(L2_central,l2norm(sys,u-uex))
21   end
22   vis=GridVisualizer(size=(700,300),xscale=:log,yscale=:log, legend=:rt,
23   xlabel="h", ylabel="|u-û|")
24   scalarplot!(vis,H,H1_sg, color=:green, label="H1_sg", markershape=:circle,
25   markersize=10,markevery=1)
26   scalarplot!(vis,H,L2_sg, color=:green,label="L2_sg",clear=false,markershape=:none, linestyle=:dash)
27   scalarplot!(vis,H,H1_central, color=:blue,
28   label="H1_central",clear=false,linestyle=:solid)
29   scalarplot!(vis,H,L2_central,
30   color=:blue,label="L2_central",clear=false,linestyle=:dash)
31   scalarplot!(vis,H,H1_upwind, color=:orange, label="H1_upwind",clear=false,
32   linestyle=:solid)
33   scalarplot!(vis,H,L2_upwind, color=:orange,label="L2_upwind",clear=false,
34   linestyle=:dash)
35   scalarplot!(vis,H,H*1.0e-1, color=:red, linestyle=:solid,
36   label="O(h)",clear=false)
37   scalarplot!(vis,H,H.^2*1.0e-1, color=:red, linestyle=:dash,
38   label="O(h^2)",clear=false)
39   reveal(vis)
40 end

```