Advanced Topics from Scientific Computing
TU Berlin Winter 2023/24
Notebook 14
(cc) BY-SA **Jürgen Fuhrmann**

```
begin
    using VoronoiFVM, ExtendableGrids
    import CairoMakie
    using GridVisualize
    GridVisualize.default_plotter!(CairoMakie)
    CairoMakie.activate!(type="svg")
end
```

VoronoiFVM fluxes and convergence

Regard an evolution equation with homogeneous Neumann boundary conditions in a domain $\Omega \subset \mathbb{R}^d$.

For $(x,t) \in \Omega \times [0,T]$, look for a function $u(x,t)$ such that

$$\begin{aligned} u_t + \nabla \cdot \vec{j} &= 0 \text{ in } \Omega \\ \vec{j} \cdot n &= 0 \text{ on } \partial\Omega \\ u(x,0) &= u_0(x) \end{aligned}$$

Flux examples:

- Diffusion: $\vec{j} = -d\nabla u$
- Nonlinear diffusion: $\vec{j} = -d(u)\nabla u = -\nabla D(u)$ where $d(u) = D'(u)$
- Convection-diffusion: $\vec{j} = -d\nabla u + u\vec{q}$ for some given vector function $\vec{q}$
- Nonlinear convection-diffusion: $\vec{j} = -\nabla D(u) + C(u)\vec{q}$

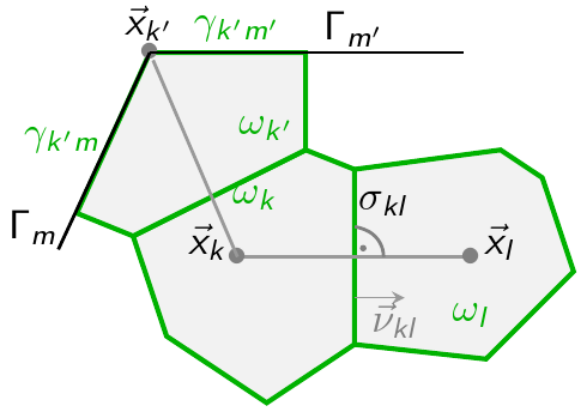
Let us look at the most general of this cases which includes the others:

- $u_0 \in L^2(\Omega)$, $\vec{q} \in C^1(\bar{\Omega}, \mathbb{R}^d)$
- $D(u) \in C^1(\mathbb{R}, \mathbb{R})$ Lipschitz continuous and strictly monotonically increasing
- $C(u) \in C^0(\mathbb{R}, \mathbb{R})$

Finite volumes and admissible fluxes

The following results come from the paper 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.

Assume that the computational domain Ω has been subivided into a finite number $\mathcal{N}$ of admissible control volumes ω_k for $k \in \mathcal{N}$ Remember the notations of the admissible finite volume partition:



Write the two-point flux finite volume space discretization combined with an implicit Euler time discretization:

$$|\omega_k| \frac{u_k^{n+1} - u_k^n}{t^{n+1} - t^n} + \sum_{l \in \mathcal{N}_k} \frac{|\sigma_{kl}|}{h_{kl}} \mathcal{G}(u_k^{n+1}, u_l^{n+1}, q_{kl}, h_{kl})$$

where

$$q_{kl} = \frac{1}{|\sigma_{kl}} \int_{\sigma_{kl}} \vec{q} \cdot \vec{\nu}_{kl} ds$$

$$u_k^0 = \frac{1}{|\omega_k|} \int_{\omega_k} u_0(x) dx.$$

We sometimes use $g(u, v) = \mathcal{G}(u, v, q, h)$, assuming that q enters implicitely.

We say that the flux $\mathcal{G}(u, v, q, h)$ is **admissible** if

1. $\mathcal{G}$ is Lipschitz continuous with respect to u, v
2. $\mathcal{G}$ is monotonically increasing with respect to u and monotonically decreasing with respect to v
3. Conservation of mass:

$$\mathcal{G}(u, v, q, h) = -\mathcal{G}(v, u, -q, h)$$

4. $\exists w \in [\mathbf{min}(u, v), \mathbf{max}(u, v)]$ such that

$$\mathcal{G}(u, v, q, h) = qhC(w) + D(u) - D(v)$$

5. For $\xi(s) = \int_0^s \sqrt{D'(t)}dt$,

$$(u - v)\mathcal{G}(a, b, q, h) \geq -h\int_u^v qC(s)ds + (\xi(v) - \xi(u))^2$$

Under the condition $\nabla \cdot \vec{q} = 0$, for admissible fluxes we get the following properties:

- Existence and uniqueness of the discrete solution u_k^n
- Mass conservation:

$$\sum_k |\omega_k| u_k^n = \sum_k |\omega_k| u_k^0 = \int_\Omega u_0 dx$$

- Boundedness of solution:

$$\mathbf{min}\{u_k^0\}_{k\in\mathcal{N}} \leq u_k^n \leq \mathbf{max}\{u_k^0\}_{k\in\mathcal{N}}$$

- Minimum and maximum principles:

$$u_k^{n+1} \leq \mathbf{max}(\{u_k^n\} \cup \{u_l^{n+1}\}_{l\in\mathcal{N}_k})$$
$$u_k^{n+1} \geq \mathbf{min}(\{u_k^n\} \cup \{u_l^{n+1}\}_{l\in\mathcal{N}_k})$$

- Convergence to the solution of the continuous problem

The results can be generalized to Dirichlet boundary conditions.

These properties give a hint on the flux functions we need to define to obtain a convergent discretization.

Linear Diffusion

In the case of linear diffusion, we have $\vec{q} = 0$. Assume that the diffusion coefficient d is constant. In this case, we can use

$$g(u,v) = \mathcal{G}(u,v,0,h) = d(u-v)$$

Conditions 1.-4. are easily verified. For condition 5, we establish that $\xi(s) = d \cdot s$ and get $d(u-v)^2$ on both sides of the inequality.

In what follows, this problem is implemented with `VoronoiFVM.jl`.

```
const d = 0.1
```

```
1 const d=0.1
```

```
1 begin
2     T=10
3     X=-1:0.1:1
4     grid1d=simplexgrid(X)
5     grid2d=simplexgrid(X,X)
6     n1d=num_nodes(grid1d)
7     n2d=num_nodes(grid2d)
8 end;
```

```
inival1d =
 [0.00673795, 0.0174224, 0.0407622, 0.0862936, 0.165299, 0.286505, 0.449329, 0.637628, 0.8
```

```
1 inival1d=map(x->exp(-5x^2),grid1d)
```

```
inival2d =
 [4.53999e-5, 0.000117391, 0.000274654, 0.000581442, 0.00111378, 0.00193045, 0.00302755, (
```

```
1 inival2d=map((x,y)-> exp(-5(x^2+y^2)), grid2d )
```

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

```
1 function lineardiffusion(y,u,edge)
2     y[1]=d*(u[1,1]-u[1,2])
3 end
```

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

```
1 storage(y,u,node)= y[1]=u[1]
```

```
linsys1d =
VoronoiFVM.System{Float64, Float64, Int32, Int64, Matrix{Int32}, Matrix{Float64}}(num_spe
```

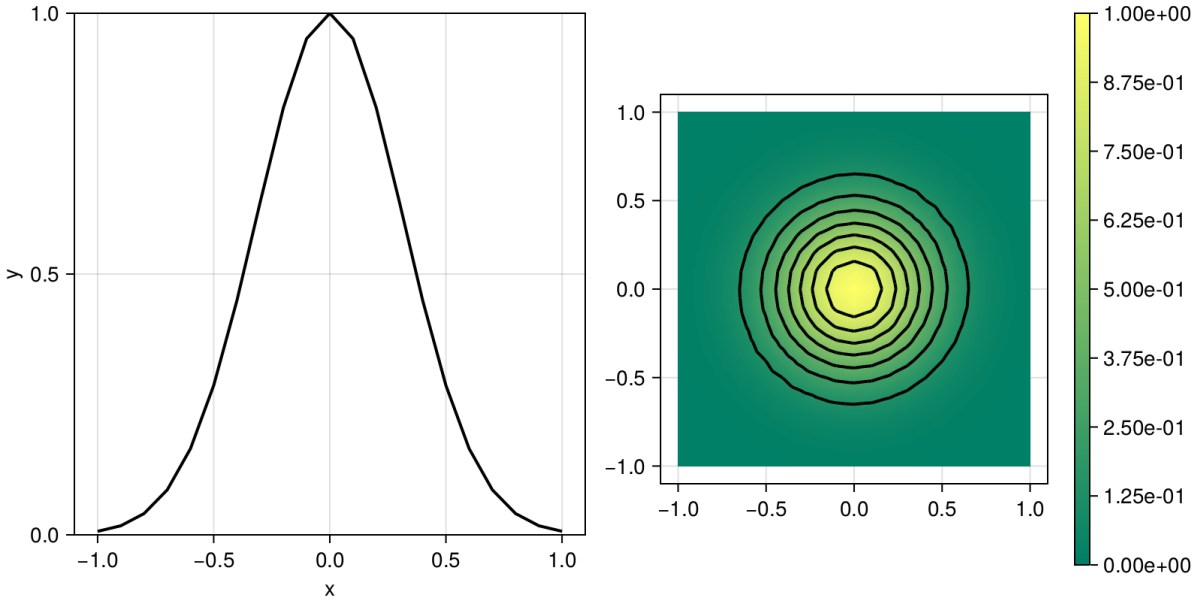
```
1 linsys1d=VoronoiFVM.System(grid1d; storage, flux=lineardiffusion,species=1)
```

```
linsys2d =
VoronoiFVM.System{Float64, Float64, Int32, Int64, Matrix{Int32}, Matrix{Float64}}(num_spe
```

```
1 linsys2d=VoronoiFVM.System(grid2d; storage, flux=lineardiffusion,species=1)
```

```
1 lintsol1d=solve(linsys1d;inival=reshape(inival1d, (1,n1d)),times=
  [0,T],du_opt=1.0e-5 );
```

```
1 lintsol2d=solve(linsys2d;inival=reshape(inival2d, (1,n2d)),times=
  [0,T],du_opt=1.0e-5 );
```



Nonlinear diffusion

Again, $\vec{q} = 0$, but now we assume that $d(u) = u^2$. There are several ways to define a flux:

- Averaging of d: $g(u,v) = \frac{(u+v)^2}{4}(u-v)$.
  - This misses monotonicity:

$$2\partial_u g = 2(u+v)(u-v) + (u+v)^2 = (u+v)(2u - 2v + u + v) = (u+v)(3u - v)$$

- Let $D(u) = \frac{u^3}{3}$, then we can define $g(u,v) = D(u) - D(v)$
  - This fulfills (1)-(4).
  - For (5) we remark that via Jensen's inequality,

$$|\int_0^v \sqrt{d(t)}dt - \int_0^u \sqrt{d(t)}dt| = |\int_u^v \sqrt{d(t)}dt| \geq \sqrt{|\int_u^v d(t)dt|}$$

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

```
function nonlineardiffusion(y,u,edge)
    D(u)=u^3/3
    y[1]=D(u[1,1])-D(u[1,2])
end

```

```
nonlinsys1d =
VoronoiFVM.System{Float64, Float64, Int32, Int64, Matrix{Int32}, Matrix{Float64}}(num_spe
```

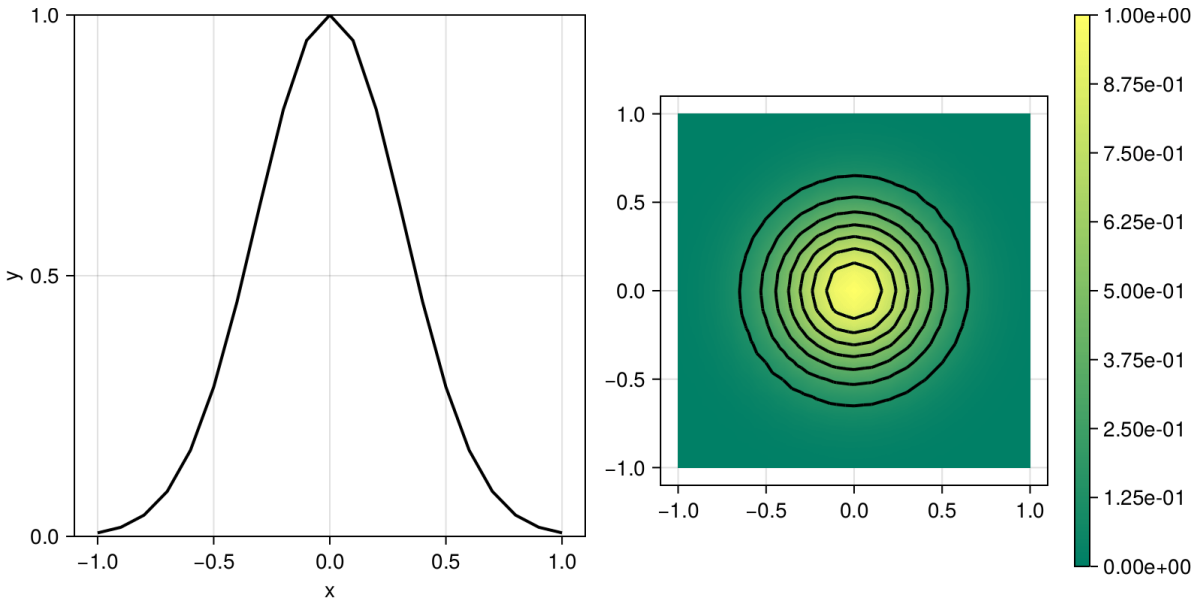
```
nonlinsys1d=VoronoiFVM.System(grid1d; storage, flux=nonlineardiffusion,species=1)

```

```
nonlinsys2d =
VoronoiFVM.System{Float64, Float64, Int32, Int64, Matrix{Int32}, Matrix{Float64}}(num_spe
```

```
nonlinsys2d=VoronoiFVM.System(grid2d; storage, flux=nonlineardiffusion,species=1)

```

```
nonlintsol1d=solve(nonlinsys1d;inival=reshape(inival1d, (1,n1d)),times
=[0,T],du_opt=1.0e-5 );

```

```
nonlintsol2d=solve(nonlinsys2d;inival=reshape(inival2d, (1,n2d)),times=
[0,T],du_opt=1.0e-5 );

```



Linear convection-diffusion

In this case, $\vec{j} = -D\nabla u + u\vec{q}$, and we would like to approximate

$$g(u_k, u_l) = \mathcal{G}(u_k, u_l, q_{kl}, h_{kl}) \approx h_{kl}\vec{j} \cdot \vec{n}_{kl}$$

We regard three possibilities.

Central difference scheme

This discretizes the convective term via arithmetic averaging of the unknowns:

$$g_c(u, v) = D(u - v) + qh\frac{u + v}{2}$$

We have

$$\partial_u g(u, v) = D + \frac{qh}{2}$$
$$\partial_v g(u, v) = -D + \frac{qh}{2}$$

So the monotonicity condition (2) can be fulfilled if

$$Pe = \frac{|q|h}{2D} < 1,$$

where Pe is called mesh Peclet number. Bear in mind that this needs to be fulfilled for all interfaces between control volumes - also for the larger ones.

In practice, this will lead to instabilities on coarse meshes.

All the other conditions (1) and (3)-(5) are fulfilled, in particular, (5) again is an identity:

$$(u - v)(d(u - v) + qh(u + v)/2) \geq -h/2q(v^2 - u^2) + d(u - v)^2$$
$$d(u - v)^2 - qh(v^2 - u^2)/2 = -h/2q(v^2 - u^2) + d(u - v)^2$$

So, here, we get all the results if the mesh size is small enough.

Simple upwind scheme

Here, we just force the monotonicity:

$$\begin{aligned} g_u(u,v) &= D(u-v) + qh \begin{cases} u, & q>0 \\ v, & q<0 \end{cases} \\ &= \left(D + \frac{|qh|}{2}\right)(u-v) + qh\frac{u+v}{2} \end{aligned}$$

This forcing of monotonicity can be intepreted as the addition of some "artificial diffusion".

Conditions (1)-(4) are clearly fulfilled, for condition (5) we use the fact that it was an identity for the central difference scheme, and artificial diffusion adds $\frac{|qh|}{2}(u-v)^2$ on the left side.

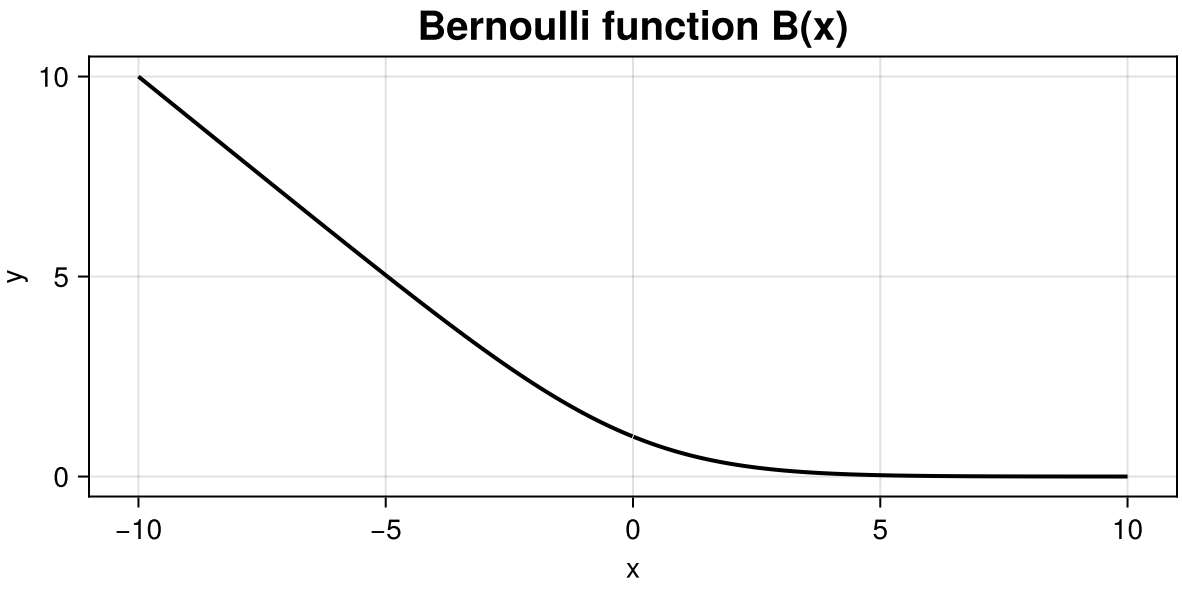
Exponential fitting scheme

This scheme is known also as Scharfetter-Gummel scheme or Il'in scheme.

Let $B(x) = \frac{x}{e^x-1}$ be the Bernoulli function.

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

```
1 B(x)=x/(exp(x)-1)
```



Here are some properties:

- $B(x) > 0$
- $B(-x) = B(x) + x$

As a discretization scheme, we use

$$\begin{aligned} g_b(u,v) &= D\left(B\left(-\frac{qh}{D}\right)u - B\left(\frac{qh}{D}\right)v\right) \\ &= (D + D_{art})(u - v) + qh\frac{u+v}{2} \end{aligned}$$

where due to $B(x) \leq 1$ for $x > 0$,

$$0 \leq D_{art} = D\left(\frac{|qh|}{2D} + B\left(\frac{|qh|}{D}\right) - 1\right) \leq \frac{|qh|}{2}$$

Thus the artificial diffusion added trough the exponential scheme tends to be less than for the simple upwind scheme.

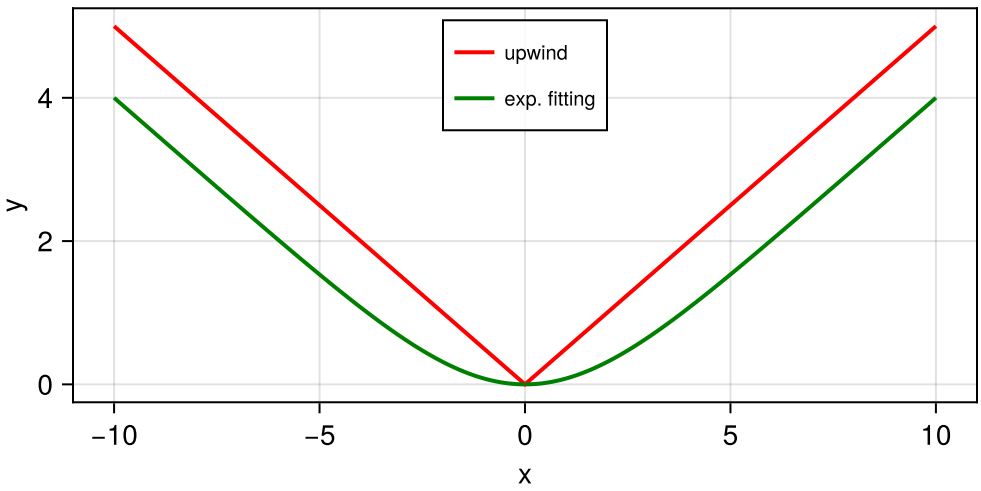
Besides of the approximation property (4), all other conditions are fulfilled.

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

```
dartupw(qh)=abs(qh)*0.5
```

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

```
dartexp(qh)=abs(qh)/2+B(abs(qh))-1
```



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

```
function convection_diffusion(;
        n=20,
        dim=1,
        tstep=1.0e-5,
        tend=1,
        dirichlet=true,
        D=0.001,
        qx=10.0,
        qy=10.0,
        scheme="expfit")

    X=collect(0:1.0/n:1)
    fpeak(x)=exp(-100*(x-0.25)^2)
    fpeak(x,y)=exp(-100*((x-0.25)^2+(y-0.25)^2))

    # copy vx, vy into vector
    if dim==1
        q=[qx]
        grid=simplexgrid(X)
    else
        q=[qx,qy]
        grid=simplexgrid(X,X)
    end

    function flux_expfit!(f,u,edge)
        qh=project(edge,q)  # Calculate projection q * (x_L-x_K)
        f[1]=D*(B(-qh/D)*u[1,1]- B(qh/D)*u[1,2])
    end

    function flux_centered!(f,u,edge)
        qh=project(edge,q)
        f[1]=D*(u[1,1]-u[1,2])+ qh*0.5*(u[1,1]+u[1,2])
    end

    function flux_upwind!(f,u,edge)
        qh=project(edge,q)
        f[1]=D*(u[1,1]-u[1,2])+ ( qh>0.0 ? qh*u[1,1] : qh*u[1,2] )
    end

    flux! =flux_upwind!
    if scheme=="expfit"
        flux! =flux_expfit!
    elseif scheme=="centered"
        flux! =flux_centered!
    end

    ## Storage term (under time derivative)
    function storage!(f,u,node)
        f[1]=u[1]
    end
    function bc!(f,u,node)
        if dirichlet
        boundary_dirichlet!(f,u,node,region=2, value=0)
        boundary_dirichlet!(f,u,node,region=3, value=0)
```

```
    end
    end

    sys=VoronoiFVM.System(grid;flux=flux!,storage=storage!, species=
[1],bcondition=bc!)

    ## Create a solution array
    inival=unknowns(sys)

    ## Broadcast the initial value
    inival[1,:].=map(fpeak,grid)

    control=VoronoiFVM.SolverControl()
    control.Δt_min=0.01*tstep
    control.Δt=tstep
    control.Δt_max=0.01*tend
    control.Δu_opt=1.0e10 # let timestep change become large
    control.verbose=""

    tsol=solve(sys;inival,times=[0,tend],control)
    return sys,tsol
end
```

nn = 10

```
nn=10
```

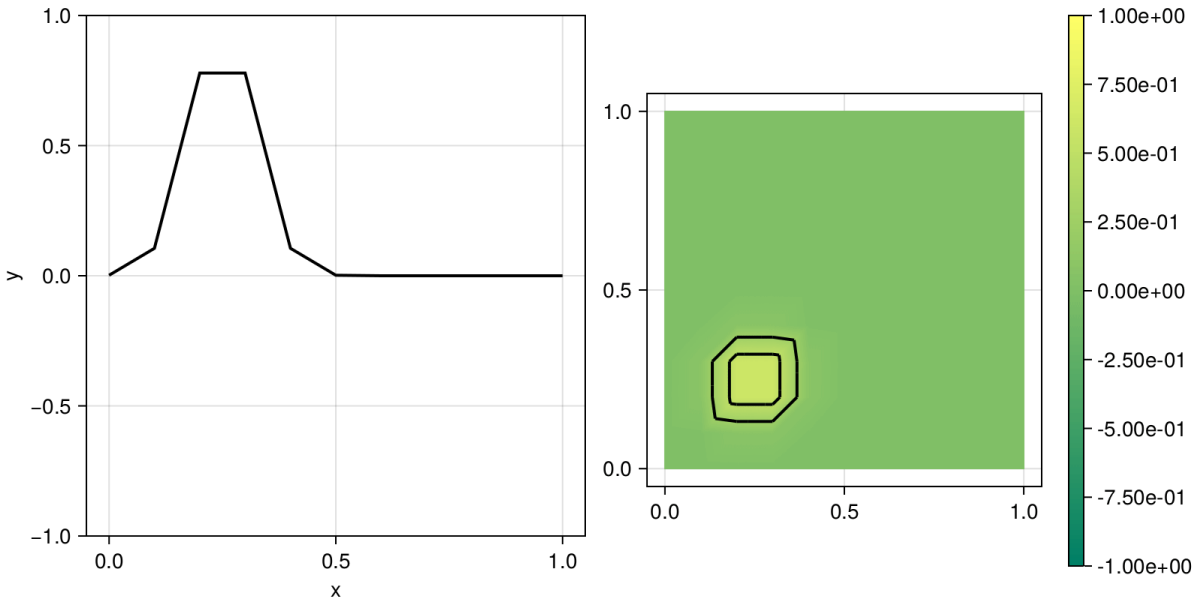
(VoronoiFVM.System{Float64, Float64, Int32, Int64, Matrix{Int32}, Matrix{Float64}}(num_

```
s1d,tsol1d=convection_diffusion(;scheme=scheme[1],dim=1,dirichlet,n=nn)
```

(VoronoiFVM.System{Float64, Float64, Int32, Int64, Matrix{Int32}, Matrix{Float64}}(num_

```
s2d,tsol2d=convection_diffusion(;scheme=scheme[1],dim=2,dirichlet,n=nn)

```

scheme: centered / expfit / upwind dirichlet: ☐


1.0
0.5
0.0
−0.5
−1.0
y
0.0
0.5
1.0
x
1.0
0.5
0.0
0.0
0.5
1.0
1.00e+00
7.50e-01
5.00e-01
2.50e-01
0.00e+00
-2.50e-01
-5.00e-01
-7.50e-01
-1.00e+00

How to get to this ?

- Project equation onto edge $x_K x_L$ of length $h = h_{kl}$, let $q = -q_{kl}$, integrate once

$$\begin{aligned} w' - wq &= j \\ w|_0 &= u \\ w|_h &= v \end{aligned}$$

- Linear ODE

Solution of the homogeneus problem:

$$\begin{aligned} w' - wq &= 0 \\ w'/w &= q \\ \ln w &= w_0 + qx \\ w &= K \exp(qx) \end{aligned}$$

Solution of the inhomogeneous problem:

- Set $K = K(x)$:

$$\begin{aligned} K' \exp(qx) + qK \exp(qx) - qK \exp(qx) &= -j \\ K' &= -j \exp(-qx) \\ K &= K_0 + \frac{1}{q} j \exp(-qx) \end{aligned}$$

- Therefore,

$$\begin{aligned} w &= K_0 \exp(qx) + \frac{1}{q} j \\ u &= K_0 + \frac{1}{q} j \\ v &= K_0 \exp(qh) + \frac{1}{q} j \end{aligned}$$

Insert boundary conditions

$$
\begin{aligned}
K_0 &= \frac{u-v}{1-\exp(qh)} \\
u &= \frac{u-v}{1-\exp(qh)} + \frac{1}{q}j \\
j &= \frac{q}{\exp(qh)-1}(u-v) + qu \\
&= q\left(\frac{1}{\exp(qh)-1}+1\right)u - \frac{q}{\exp(qh)-1}v \\
&= q\left(\frac{\exp(qh)}{\exp(qh)-1}\right)u - \frac{q}{\exp(qh)-1}v \\
&= \frac{-q}{\exp(-qh)-1}u - \frac{q}{\exp(qh)-1}v \\
&= \frac{B(-qh)u - B(qh)v}{h}
\end{aligned}
$$

where $B(\xi) = \frac{\xi}{\exp(\xi)-1}$: Bernoulli function

- General case: $Dw' - wq = D(u' - w\frac{q}{D})$

The general case

This "exact solution" approach was the idea of Scharfetter and Gummel 1969.

In the paper of Eymard, Fuhrmann and Gärtner, it is shown that this approach can be generalized to then nonlinear flux $\vec{j} = -\nabla D(u) + C(u)\vec{q}$.

Let for $u < v$

$$H(x) = \int_u^v \frac{D'(s)}{qC(s) - x}\mathrm{h} ds$$

The we define $\mathcal{G}$ from the solution of $H(\mathcal{G}/h) = 0$ and set $\mathcal{G}(u, u, q, h) = hqC(u)$,
$\mathcal{G}(u, v, q, h) = \mathcal{G}(v, u, -q, h)$ for $u > v$. This value solves the nonlinear two point Dirichlet problem

$$\begin{aligned} D(w)' - C(w)q &= j \\ w|_0 &= u \\ w|_h &= v \end{aligned}$$

The resulting flux fulfills all the conditions (1)-(5).

- The Scharfetter-Gummel scheme falls into this category, so it also fulfills (4)
- The nonlinar diffusion flux $D(u) - D(v)$ falls into this category
- In general the integral equation cannot be solved analytically
- These facts inspired further research based on the Idea to approximate the integral equation in a certain way.
- In general, the ideas described here provide some heuristics on how to approach more complicated problem, e.g. systems of partial differential equations.
- In particular the monotonicity condition leads to a Jacobi matrix with M property.

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

```
function animatesolution(system, tsol;
                         video = "tmp.gif",
                         nframes = 201,
                         limits = (0,1),
                         size = (400, 400),
                         pscale_bar = 1.0,
                         Plotter = GridVisualize.default_plotter(),
                         vis = nothing)
    title = ""
    vis = GridVisualizer(; size, limits, title, Plotter)
    trange = range(extrema(tsol.t)...; length = nframes)
    movie(vis; file = video) do vis
        for t in trange
            scalarplot!(vis,system, tsol(t); title, colormap=:summer)
            reveal(vis)
        end
    end
    return video
end

```

Table of Contents