Advanced Topics from Scientific Computing
TU Berlin Winter 2024/25
Notebook 17
 Jürgen Fuhrmann

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

Table of Contents

Postprocessing solutions with VoronoiFVM.jl
- Sample problem
- Stationary case
  - Reaction == creation ?
  - Outflow == reaction ?
    - The test function trick
- Transient problem

Postprocessing solutions with VoronoiFVM.jl

After calculating solutions based on the finite volume method, it may be interesting to obtain information about the solution besides of the graphical representation.

Here, we focus on the following data:

- integrals of the solution
- flux through parts of the boundary

Sample problem

Define a sample problem for discussing these issues.

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

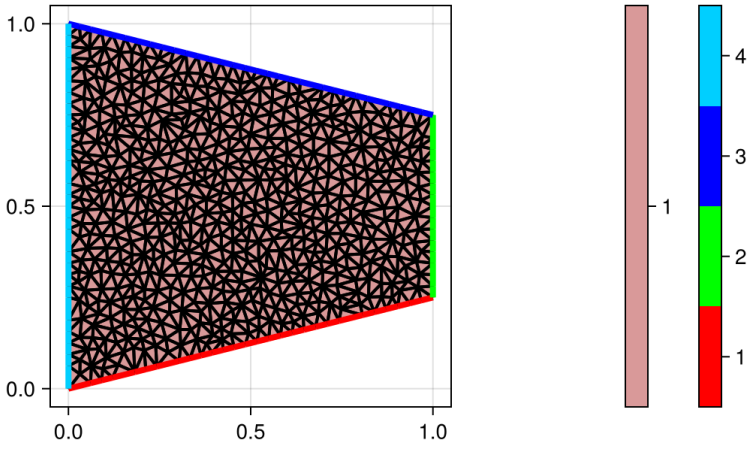
```
function make_grid(; maxvolume = 0.001)
    builder = SimplexGridBuilder(Generator = Triangulate)
    p00 = point!(builder, 0, 0)
    p10 = point!(builder, 1, 0.25)
    p11 = point!(builder, 1, 0.75)
    p01 = point!(builder, 0, 1)

    facetregion!(builder, 1)
    facet!(builder, p00, p10)
    facetregion!(builder, 2)
    facet!(builder, p10, p11)
    facetregion!(builder, 3)
    facet!(builder, p11, p01)
    facetregion!(builder, 4)
    facet!(builder, p00, p01)

    return simplexgrid(builder, maxvolume = maxvolume)
end
```

```
grid = ExtendableGrids.ExtendableGrid{Float64, Int32}
           dim =       2
        nnodes =     631
        ncells =    1159
       nbfaces =     101
```

```
grid = make_grid()
```



Let us define the following reaction - diffusion system in Ω:

$$\partial_t u_1 - \nabla \cdot (\nabla u_1) + r(u_1, u_2) = f = 1.0$$
$$\partial_t u_2 - \nabla \cdot (\nabla u_2) - r(u_1, u_2) = 0$$

with Dirichlet boundary conditions $u_2 = 0$ on $\Gamma_2 \subset \partial\Omega$ and homogeneous Neumann boundary conditions $\partial_n u_i = 0$ for $i = 1, 2$ otherwise.

Let $r(u_1, u_2) = u_1 + 0.1u_2$.

Intuitively, the source f creates species u_1 which reacts to u_2, which then leaves the domain at boundary Γ_2.

```
function storage(f, u, node, data)
    f .= u
    return nothing
end;
```

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

```
r(u1, u2) = u1 - 0.1 * u2;
```

```
function reaction(f, u, node, data)
    f[1] = r(u[1], u[2])
    f[2] = -r(u[1], u[2])
    return nothing
end;
```

... be careful with the sign: reaction is on the left hand side, source on the right hand side.

```
function source(f, node, data)
    f[1] = 1.0
    return nothing
end;
```

```
function bcondition(f, u, node, data)
    boundary_dirichlet!(f, u, node, region = 2, species = 2, value = 0)
    return nothing
end;
```

Create the system:

```
system =
VoronoiFVM.System{Float64, Float64, Int32, Int64, Matrix{Int32}}(
  grid = ExtendableGrids.ExtendableGrid{Float64, Int32}(dim=2, nnodes=631, ncells=1159
  nbfaces=101),
  physics = Physics(flux=flux, storage=storage, reaction=reaction, source=source,
  breaction=bcondition, ),
  num_species = 2)
```

```
system = VoronoiFVM.System(grid; flux, storage, reaction, source, bcondition,
  species = [1, 2])
```

Stationary case

For this problem, we have the following flux balances derived from the equations and from Gauss' theorem:

$$\int_\Omega r(u_1, u_2) d\omega = \int_\Omega f d\omega$$
$$\int_\Omega -r(u_1, u_2) d\omega = \int_{\Gamma_2} \nabla u_2 \cdot \vec{n} ds$$

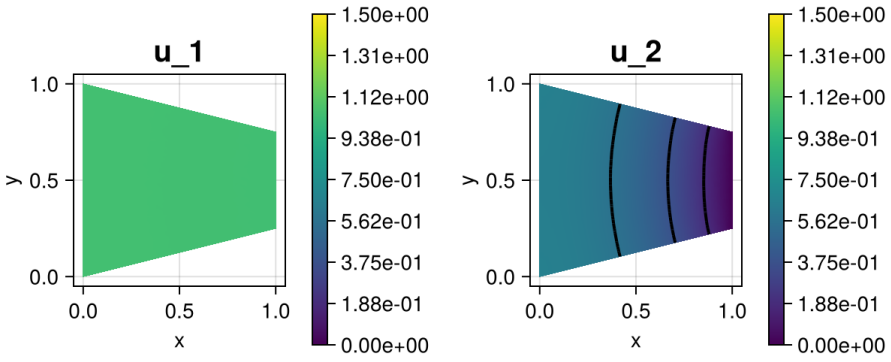
The volume integrals can be approximated based on the finite volume subdivision $\Omega = \cup_{i \in \mathcal{N}} \omega_i$:

$$\int_\Omega r(u_1, u_2) d\omega \approx \sum_{i \in \mathcal{N}} |\omega_i| r(u_{1,i}, u_{2,i})$$

$$\int_\Omega f d\omega \approx \sum_{i \in \mathcal{N}} |\omega_i| f_i$$

```
u = 2×631 VoronoiFVM.DenseSolutionArray{Float64, 2}:
    1.04961   1.04503      1.04503      …  1.0496    1.04956   1.04573  1.04954
    0.647343  2.35291e-30  1.99461e-30     0.646598  0.644086  0.26481  0.642655
```

```
1 u = solve(system)
```



The `integrate` method of `VoronoiFVM` provides a possibility to calculate the volume integral of a function of a solution as described above. It returns a `num_species` × `num_regions` matrix of the integrals of the function of the unknowns over the different subdomains (here, we have only one):

- Amount of u_1 and u_2 in the domain aka integral over identity storage function:

```
U = 2×1 VoronoiFVM.SolutionIntegral{Float64}:
    0.7858573677959029
    0.35857367795902967
```

```
1 U = integrate(system, storage, u)
```

- Amount of species created by source term per unit time:

```
F = 2×1 VoronoiFVM.SolutionIntegral{Float64}:
    0.7499999999999993
    0.0
```

```
1 F = integrate(system, (f, u, node, data) -> source(f, node, data), u)
```

- Amount of reaction per unit time:

```
R = 2×1 VoronoiFVM.SolutionIntegral{Float64}:
     0.7499999999999993
    -0.7499999999999993
```

```
1 R = integrate(system, reaction, u)
```

Reaction == creation ?

Let us check our first identity: creation rate = reaction rate:

```
true
```

```
1 F[1] ≈ R[1]
```

Outflow == reaction ?

The test function trick

Literature references:

- H. Gajewski "Analysis und Numerik von Ladungstransport in Halbleitern", WIAS Berlin, Report No.6
- Yoder, P. D., K. Gärtner, and W. Fichtner. "A generalized Ramo–Shockley theorem for classical to quantum transport at arbitrary frequencies." Journal of Applied Physics 79.4 (1996): 1951-1954.
- P. Farrell, N. Rotundo, D. H. Doan, M. Kantner, J. Fuhrmann, and T. Koprucki, "Numerical methods for drift-diffusion models", in Handbook of optoelectronic device modeling and simulation: Lasers, modulators, photodetectors, solar cells, and numerical methods, vol. 2, J. Piprek, Ed. Boca Raton: CRC Press, 2017, pp. 733–771.

But what about the boundary integral ? Here, we use a trick to cast the surface integral to a volume integral with the help of a test function:

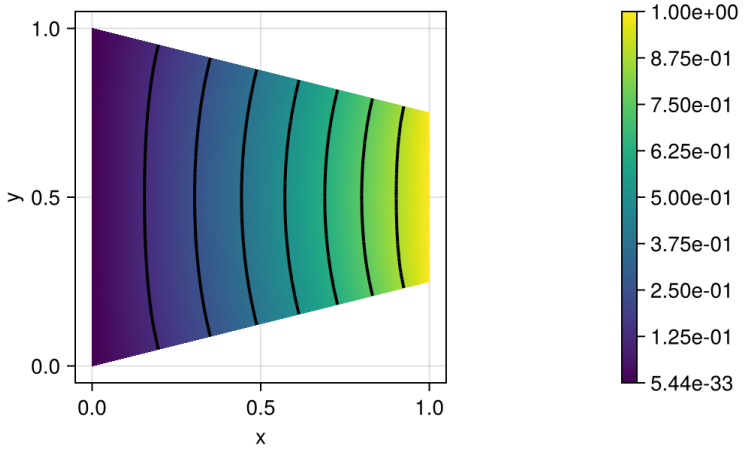
Let $T(x)$ be the solution of the Laplace problem $-\nabla^2 T = 0$ in Ω and the boundary conditions

$$\begin{aligned} T &= 0 \quad \text{at } \Gamma_4 \\ T &= 1 \quad \text{at } \Gamma_2 \\ \partial_n T &= 0 \quad \text{at } \Gamma_1, \Gamma_3 \end{aligned}$$

`VoronoiFVM.jl` provides a special API for obtaining such a test function:

```
▸ [1.30582e-32, 1.0, 1.0, 5.43952e-33, 0.400827, 1.24937e-32, 0.372123, 0.613776, 1.0, 0.6
```

```
begin
    tf = VoronoiFVM.TestFunctionFactory(system)
    Γ_where_T_equal_1 = [2]
    Γ_where_T_equal_0 = [4]
    T = testfunction(tf, Γ_where_T_equal_0, Γ_where_T_equal_1)
end
```



Write $\vec{j} = -\nabla u$. and assume $\nabla \cdot \vec{j} + r = f$

$$\begin{aligned}\int_{\Gamma_2} \vec{j}\cdot\vec{n}ds &= \int_{\Gamma_2} T\vec{j}\cdot\vec{n}ds \quad \text{due to } T = 1 \text{ on } \Gamma_2\\ &= \int_{\partial\Omega} T\vec{j}\cdot\vec{n}ds \quad \text{due to } T = 0 \text{ on } \Gamma_4, \quad \vec{j}\cdot\vec{n} = 0 \text{ on } \Gamma_1, \Gamma_3\\ &= \int_\Omega \nabla\cdot(T\vec{j})d\omega \quad \text{(Gauss)}\\ &= \int_\Omega \nabla T\cdot\vec{j}d\omega + \int_\Omega T\nabla\cdot j d\omega\\ &= \int_\Omega \nabla T\cdot\vec{j}d\omega + \int_\Omega T(f-r)d\omega\end{aligned}$$

and we approximate

$$\int_\Omega \nabla T\cdot\vec{j}d\omega \approx \sum_{k,l}\frac{|\omega_k\cap\omega_l|}{h_{k,l}}g(u_k,u_l)(T_k - T_l)$$

where the sum runs over pairs of neigboring control volumes.

The `integrate` method with a test function parameter returns a value for each species, the sign convention assumes that species leaving the domain lead to negative values.

```
I = [-2.59422e-17, -0.75]
```
```
1 I = integrate(system, T, u)
```

Check that none of u_1 leaves the domain through the boundary:

```
true
```
```
1 isapprox(I[1], 0.0, atol = 1.0e-16)
```

Check that creation of u_2 in the reaction is balanced by u_2 leaving the domain through Γ_2:

```
true
```
```
1 R[2] ≈ I[2]
```

So we indeed can confirm the requirement for the right balance of source, reaction and outflow.

Transient problem

For the transient case, in addition, we need to consider the time derivative part along with reaction and source. In the derivation of the test function procedure, under the assumption of the implicit Euler time discretization method, this can be achieved by handling the finite difference in time along with source and reaction.

```
1 t0 = 0.0; tend = 10;
```

```
1.0
```
```
1 begin
2     control = VoronoiFVM.SolverControl()
3     control.Δu_opt = 0.025
4     control.Δt_min = 1.0e-4
5     control.Δt = 0.1
6     control.Δt_max = 1.0
7 end
```

```
tsol =
t: 53-element Vector{Float64}:
  0.0
  0.025
  0.05062345404004061
  0.0769001854088209
  0.10386217678426005
  0.1315438095163105
  0.1599821265631871
  ⋮
  5.595054837294435
  6.4363060481432965
  7.4363060481432965
  8.290870698762198
  9.1454353493811
 10.0
u: 53-element Vector{Matrix{Float64}}:
 [0.0 0.0 … 0.0 0.0; 0.0 0.0 … 0.0 0.0]
 [0.024391715458153492 0.02439103559145379 … 0.024391316041926007 0.024391713663289672
 [0.04877017950721862 0.04876745359410663 … 0.048768424218756025 0.04877016887908831; (
 [0.07313478503919922 0.07312797251876164 … 0.07313012375705118 0.07313474984519147; 0
 [0.09748485827766361 0.09747128170921528 … 0.09747516504251821 0.09748477187771133; 0
 [0.12181966040959864 0.1217960609707473 … 0.12180227874056154 0.12181948453833605; 0.(
 [0.14613838921202293 0.14610099205942248 … 0.14611019067265374 0.14613807441782972; 0
 ⋮
 [1.0388506922394751 1.0343650580116264 … 1.0350524254121611 1.0387812653541038; 0.634(
 [1.0435274931163017 1.0389987142479964 … 1.0396924386784223 1.043457375027734; 0.6401(
 [1.0464325650660216 1.0418768931114988 … 1.0425745794433452 1.046362016077883; 0.6435(
 [1.0478267509453023 1.043258145685866 … 1.0439577371886333 1.0477559947146662; 0.6452(
 [1.0486097474357252 1.0440338687069672 … 1.0447345315829788 1.0485388746495086; 0.646(
 [1.0490495039233827 1.0444695365183674 … 1.0451708016285324 1.0489785656149326; 0.646(
```

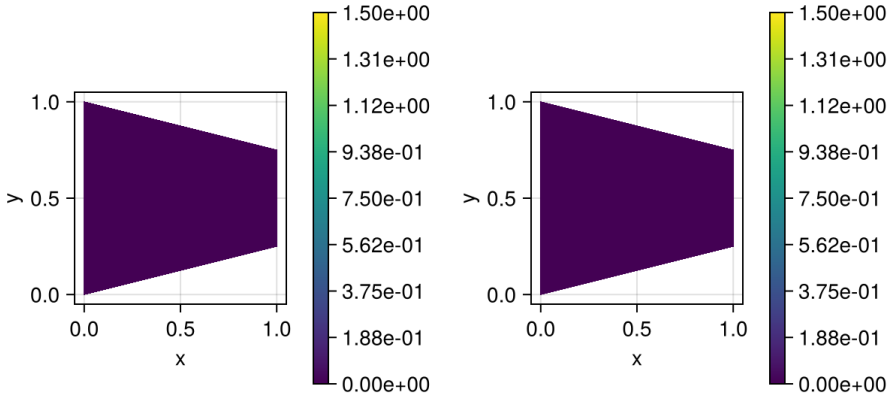
```
1 tsol = solve(system; inival = 0, times = [t0, tend], control = control)
```

The call to `solve` returns a solution object which is compatible to that from `SciML`.

In particular, a TransientSolution `tsol` can be accessed as follows:

- `tsol[:,:,it]` returns the solution for timestep `i`
- `tsol[ispec,:,it]` returns the solution for component `ispec` at timestep `i`
- `tsol(t)` returns a (linearly) interpolated solution value for `t`.
- `tsol.t[it]` is the corresponding time
- `tsol[ispec,ix,it]` refers to solution of component `ispec` at node `ix` at moment `it`

Time: 0.0

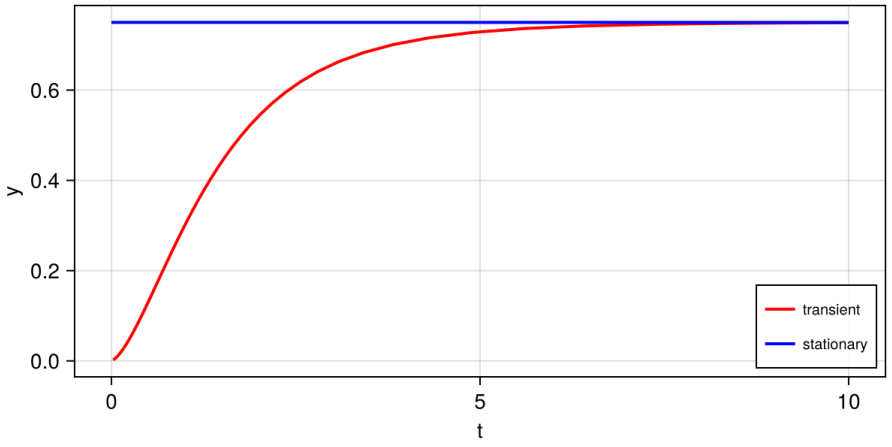


From the solution we now can calculate the normal flux via our test function "trick", once again through the API provided by VoronoiFVM:

▸ [-0.00208035, -0.00529592, -0.00943777, -0.0144092, -0.0201561, -0.0266454, -0.0338557,

```
begin
    outflow_rate = Float64[]
    for i in 2:length(tsol)
        ofr = integrate(system, T, tsol[i], tsol[i - 1], tsol.t[i] - tsol.t[i - 1])
        push!(outflow_rate, ofr[2])
    end
    outflow_rate
end
```

For increasing time, the outflow rate should approach the value we calculated from the stationary solution:




```
let
    vis = GridVisualizer(resolution = (600, 300), legend = :rb, xlabel = "t")
    scalarplot!(vis, tsol.t[2:end], -outflow_rate, label = "transient", color = :red)
    scalarplot!(vis, [0, tend], -[I[2], I[2]], label = "stationary", color = :blue, clear = false)
    reveal(vis)
end
```

The overall amount of species which left the domain is can be calculated integrating the discrete outflow rate over time

$$I_{all} = \int_{t_0}^{t_{end}} \int_{\Gamma_2} \nabla u_2 \cdot \vec{n} ds$$

```
6.356355030740734
```

```
begin
    I_all = 0.0
    for i in 1:(length(tsol) - 1)
        I_all -= outflow_rate[i] * (tsol.t[i + 1] - tsol.t[i])
    end
    I_all
end
```

The amount of species created via the source term (measured in `F`) integrated over time should be equal to the sum of the amount of species left in the domain at the very end of the evolution and the amount of species which left the domain:

$$\int_{t_0}^{t_{end}} \int_{\Omega} f d\omega dt = \int_{\Omega} (u_1 + u_2) d\omega + I_{all}$$

```
Uend = 2×1 VoronoiFVM.SolutionIntegral{Float64}:
       0.7854363436552765
       0.3582086256039905
```

```
Uend = integrate(system, storage, tsol[end])
```

```
true
```

```
1 F[1] * (tend - t0) ≈ (Uend[1] + Uend[2] + I_all)
```