Table of Contents

Extracting integral data from Finite Volume solutions

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

Here, we define a sample problem for discussing these issues, which could be formulated in a more general way as well.

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

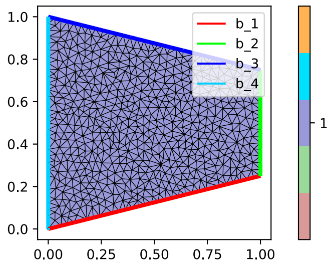
```
function make_grid(;maxvolume=0.001)
    builder=SimplexGridBuilder(Generator=Triangulate)
    p00=point!(builder, 0,0)
    p10=point!(builder, 1,0.25)
    p11=point!(builder, 1,0.75)
    p01=point!(builder, 0,1)

    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)

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

```
grid = ExtendableGrids.ExtendableGrid{Float64, Int32};
       dim: 2 nodes: 631 cells: 1159 bfaces: 101
```

```
grid=make_grid()
```



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

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

with boundary conditions $u_2 = 0$ on $\Gamma_2 \subset \partial\Omega$ and $r(u_1, u_2) = u_1 + 0.1 u_2$.

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

```
function storage(f,u,node)
    f.=u
end;
```

```
function flux(f,_u,edge)
    u=unknowns(edge,_u)
    f[1]=u[1,1]-u[1,2]
    f[2]=u[2,1]-u[2,2]
end;
```

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

```
function reaction(f,u,node)
    f[1]= r(u[1],u[2])
    f[2]=-r(u[1],u[2])
end;
```

```
function source(f,node)
    f[1]=1.0
end;
```

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

Create the system, enable species, set boundary condition, solve, create initial value:

```
physics =
VoronoiFVM.Physics(num_species=2, flux=flux, storage=storage, reaction=reaction, source=so
```

```
physics=VoronoiFVM.Physics(num_species=2,
    flux=flux,
    storage=storage,
    reaction=reaction,
    source=source)
```

```
begin
    system=VoronoiFVM.System(grid,physics)
    enable_species!(system,1,[1])
    enable_species!(system,2,[1])
end
```

```
boundary_dirichlet!(system,2,2,0.0);
```

```
inival =
2×631 Matrix{Float64}:
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
inival=unknowns(system,inival=0.0)
```

Stationary case

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

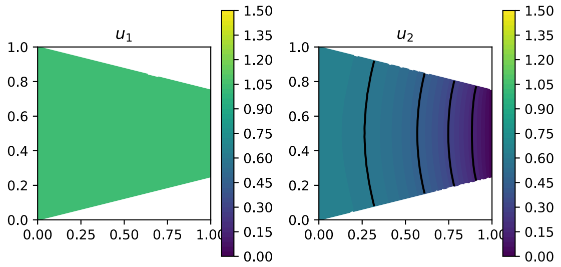
$$\int_\Omega r(u_1,u_2)d\omega = \int_\Omega f d\omega$$
$$\int_\Omega -r(u_1,u_2)d\omega = \int_{\Gamma_2} \nabla u_2 \cdot \vec{n} ds$$

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 Matrix{Float64}:
     1.04961   1.04503      1.04503     …  1.0496    1.04956   1.04573  1.04954
     0.647343  3.67642e-32  6.23316e-32    0.646598  0.644086  0.26481  0.642655
```

```
u=solve(inival,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 Matrix{Float64}:
     0.7858573677959029
     0.35857367795902967
```

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

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

```
F = 2×1 Matrix{Float64}:
     0.7499999999999993
     0.0
```

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

- Amount of reaction per unit time:

```
R = 2×1 Matrix{Float64}:
      0.7499999999999992
     -0.7499999999999992
```

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

Reaction == creation ?

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

```
true
```

```
F[1] ≈ R[1]
```

Outflow == reaction ?

The test function trick

This trick goes back to work of H. Gajewski in the field of semiconductor device simulation.

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:

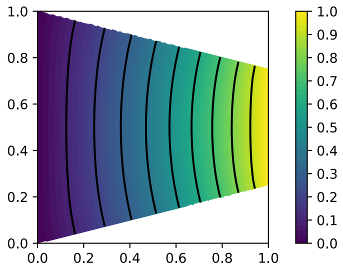
Let $T(x)$ be the solution of the Laplace problem $-\nabla^2 T = 0$ in Ω and the boundary conditions

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

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

```
Float64[1.30582e-32,  1.0,  1.0,  5.43952e-33,  0.400827,  1.24937e-32,  0.372123,  0.61377
```

```
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 =  Float64[-2.7333e-17,  -0.75]
```

```
I=integrate(system,T,u)
```

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

```
true
```

```
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
```

```
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.

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

```
control=VoronoiFVM.NewtonControl();
```

```
control.Δu_opt=0.025;
```

```
control.Δt_min=1.0e-4;
```

```
control.Δt=0.1;
```

```
control.Δt_max=1.0;
```

```
tsol =
t: 52-element Vector{Float64}:
  0.0
  0.05
  0.07624404074558075
  0.10317181117166707
  0.1308175835099931
  0.15921827342132694
  0.18841372976572393
  ⋮
  5.559040374037019
  6.3928478027881015
  7.3928478027881015
  8.3928478027881
  9.3928478027881
 10.0
u: 52-element Vector{Matrix{Float64}}:
 [0.0 0.0 … 0.0 0.0; 0.0 0.0 … 0.0 0.0]
 [0.04762986051262805 0.04762515338868721 … 0.047626533643388674 0.04762983009855026; 0
 [0.07199507932790919 0.07198603534863536 … 0.07198859299349966 0.071995017790104; 0.00
 [0.09634579206667453 0.09632984681193643 … 0.09633412101648231 0.09634567399379049; 0.0
 [0.1206812680926302 0.12065526905151462 … 0.1206618529370015 0.12068105702750248; 0.00
 [0.1450007114816121 0.14496097925801335 … 0.14497051166089883 0.14500035994329577; 0.0
 [0.1693032610197695 0.16924565599166444 … 0.16925881538494889 0.16930271135296024; 0.0
 ⋮
```

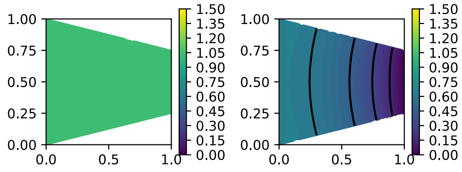
```
tsol=solve(inival,system,[t0,tend],control=control)
```

The call to `solve` corresponds to a new API for transient solutions. It returns a solution object which is compatible to that from `DifferentialEquations.jl`.

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

- `tsol[it]` contains the solution for timestep `i`
- `tsol[ispec,:,it]` contains 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: 6.7

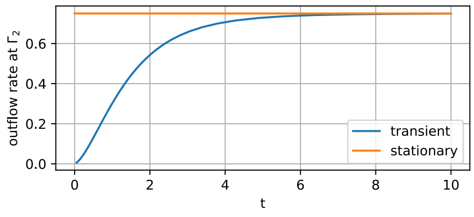


From the solution we now can calculate the normal flux via our test function "trick", once again through the API provided by VoronoiFVM:

```
Float64[-0.00586532, -0.00987319, -0.0147437, -0.0204071, -0.0268231, -0.0339665, -0.
```

```
begin
    outflow_rate=Float64[]
   for i=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:



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

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

```
6.35637165496992
```

```
begin
    I_all=0.0
    for i=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 Matrix{Float64}:
        0.7854274358734347
        0.35820090915664426
```

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

```
true
```

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

```
    Status `/tmp/jl_9t6uW1/Project.toml`
  [cfc395e8] ExtendableGrids v0.7.4
  [7f904dfe] PlutoUI v0.7.4
  [d330b81b] PyPlot v2.9.0
  [57bfcd06] SimplexGridFactory v0.5.1
  [f7e6ffb2] Triangulate v1.0.1
  [82b139dc] VoronoiFVM v0.10.8
```