Scientific Computing TU Berlin Winter 2021/22 © Jürgen Fuhrmann
Notebook 29

PlutoVista

```
begin
    using PlutoUI   ,HypertextLiteral   , ExtendableGrids   , VoronoiFVM   ,
PlutoVista   ,GridVisualize   ,LinearAlgebra
    default_plotter!(PlutoVista);
end
```

Finite Volumes for systems of partial differential equations

A system of reaction-diffusion equations

n coupled PDEs in $\Omega \subset \mathbb{R}^d$:

Denote n-vectors by bold face and d-vectors by arrows. Let $\mathbf{u}(\vec{x},t) = (u_1(\vec{x},t) \dots u_n(\vec{x},t))$ be a n-vector function.

$$\begin{aligned} \partial_t s_1(\mathbf{u}) - \nabla \cdot \vec{j}_1(\mathbf{u}, \vec{\nabla}\mathbf{u}) + r_1(\mathbf{u}) &= f_1 \\ &\vdots \\ \partial_t s_n(\mathbf{u}) - \nabla \cdot \vec{j}_n(\mathbf{u}, \vec{\nabla}\mathbf{u}) + r_n(\mathbf{u}) &= f_n \end{aligned}$$

In vector form, this can be rewritten as:

$$\partial_t s(\mathbf{u}) - \nabla \cdot \vec{\mathbf{j}}(\mathbf{u}, \vec{\nabla}\mathbf{u}) + \mathbf{r}(\mathbf{u}) = \mathbf{f}$$

- "Storage" $\mathbf{s} : \mathbb{R}^n \to \mathbb{R}^n$
- "Reaction" $\mathbf{r} : \mathbb{R}^n \to \mathbb{R}^n$
- "Flux" $\vec{\mathbf{j}} : \mathbb{R}^n \times \mathbb{R}^{nd} \to \mathbb{R}^{nd}$
- "Source" $\mathbf{f} : \Omega \to \mathbb{R}^n$
- $\mathbf{s}, \vec{\mathbf{j}}, \mathbf{r}$ can depend on $\vec{x}, t$ as well.

Similar for nonlinear Robin boundary conditions:

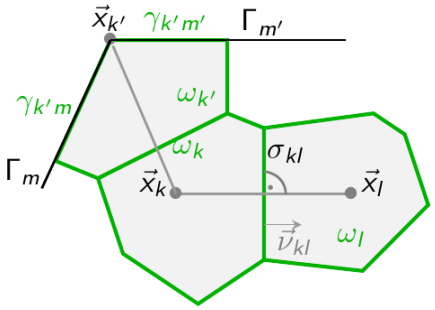
$$\begin{aligned} j_1(\mathbf{u}, \vec{\nabla}\mathbf{u}) \cdot \vec{n} + a_1(\mathbf{u}) &= b_1 \\ &\vdots \\ j_n(\mathbf{u}, \vec{\nabla}\mathbf{u}) \cdot \vec{n} + a_n(\mathbf{u}) &= b_n \end{aligned}$$

or

$$\vec{\mathbf{j}}(\mathbf{u}, \vec{\nabla}\mathbf{u}) + \mathbf{a}(\mathbf{u}) = \mathbf{b}$$

- "Boundary reaction" $\mathbf{a} : \mathbb{R}^n \to \mathbb{R}^n$
- "Boundary source" $\mathbf{b} : \partial\Omega \to \mathbb{R}^n$

The discrete version



Let N be the number of control volumes ω_k / collocation points $\vec{x}_k$. For $k = 1 \dots N$ write

$$|\omega_k| \frac{\mathbf{s}(\mathbf{u}_k) - \mathbf{s}(\mathbf{u}_k^{\text{old}})}{\Delta t} + \sum_{l \in \mathcal{N}_k} \frac{|\sigma_{kl}|}{h_{kl}} \mathbf{g}(\mathbf{u}_k, \mathbf{u}_l) + |\omega_k| \mathbf{r}(\mathbf{u}_k) + |\gamma_k| \mathbf{a}(\mathbf{u}_k) = |\omega_k| \mathbf{f}_k + |\gamma_k| \mathbf{b}_k$$

- ω_k: control volume
- γ_k: boundary interface ($\emptyset$ for interior nodes)
- σ_{kl}: interface between neigboring control volumes
- h_{kl}: distance between neigboring collocation points

With exception of $\vec{\mathbf{j}}$, all constitutive functions introduced above can be used in the discrete version as well. The flux $\vec{\mathbf{j}}$ is replaced by the discrete edge flux $\mathbf{g} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$.

Dirichlet boundary conditions can be described within this formulation via the penalty method.

Example: a reaction-diffusion problem

Two species u_1 and u_2 move by diffusion in $\Omega = (0, 1)$. Both have initial concentration zero, and starting with $t = 0$, at $x = 0$, u_1 enters the domain with concentration 1. u_1 is not allowed to leave the domain at $x = 1$.

Within Ω, u_1 reacts to u_2 with foward reaction constant k^+ and backward reaction constant k^- The boundary $x = 0$ is insulating for u_2, and at $x = 1$, u_2 has forced concentration 0.

$$\begin{aligned}
\partial_t u_1 - \nabla \cdot D_1 \vec{\nabla} u_1 + r_1(u_1, u_2) &= 0 \\
\partial_t u_2 - \nabla \cdot D_2 \vec{\nabla} u_1 + r_2(u_1, u_2) &= 0 \\
r_1(u_1, u_2) &= k^+ u_1 - k^- u_2 \\
r_2(u_1, u_2) &= -r_1(u_1, u_2) \\
u_1|_{x=0} &= 1 \\
D_1 \vec{\nabla} u_1 \cdot \vec{n}|_{x=1} &= 0 \\
D_2 \vec{\nabla} u_2 \cdot \vec{n}|_{x=0} &= 0 \\
u_2|_{x=1} &= 0 \\
u_1|_{t=0} &= 0 \\
u_2|_{t=0} &= 0
\end{aligned}$$

```
begin
    const kp=1
    const km=1
    const D_1=0.5
    const D_2=0.1
end;
```

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

```
function storage(f,u,node)
    f[1]=u[1]
    f[2]=u[2]
end
```

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

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

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

```
function bcondition(f,u,bnode)
    v=ramp(bnode.time,du=(0,1),dt=(0,1.0e-2))
    boundary_dirichlet!(f,u,bnode,species=1,region=1,value=v)
    boundary_neumann!(f,u,bnode,species=1,region=2,value=0)
    boundary_dirichlet!(f,u,bnode,species=2,region=2,value=0)
    boundary_neumann!(f,u,bnode,species=2,region=1,value=0)
end
```

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

```
function flux(f,u,edge)
    f[1]=D_1*(u[1,1]-u[1,2])
    f[2]=D_2*(u[2,1]-u[2,2])
end
```

```
grid = ExtendableGrids.ExtendableGrid{Float64, Int32};
       dim: 1 nodes: 101 cells: 100 bfaces: 2
```

```
grid=simplexgrid(0:0.01:1)
```

```
system =
VoronoiFVM.System{Float64, Int32, Int64, Matrix{Int32}, Matrix{Float64}}(num_species=2)
```

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

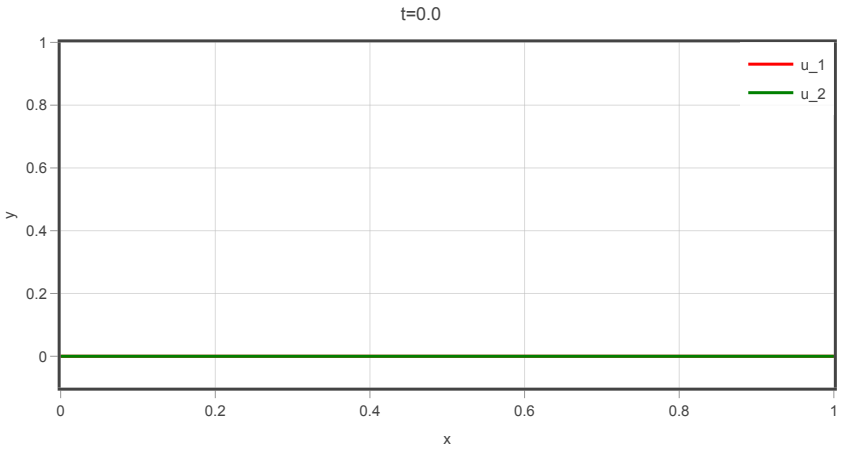
```
tend = 10
```

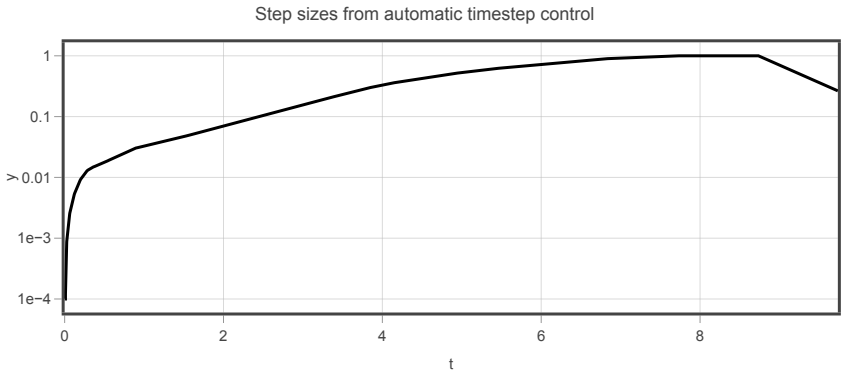
```
tend=10
```

```
tsol =
t: 283-element Vector{Float64}:
  0.0
  0.0001
  0.00019999999999990002
  0.0002999999999998
  0.0003999999999997
  0.0004999999999995999
  0.0005999999999995
  ⋮
  6.088252936255219
  6.836735787735569
  7.734915209511989
  8.734915209511989
  9.734915209511989
 10.0
u: 283-element Vector{Matrix{Float64}}:
 [0.0 0.0 … 0.0 0.0; 0.0 0.0 … 0.0 0.0]
 [0.01 0.002679337251044692 … 2.5399194575525486e-59 1.269896233964575e-59; 8.839560475852
 [0.01999999999999 0.006905487493957543 … 1.5044248852517111e-57 7.585239975954252e-58; 2.
 [0.029999999999979998 0.012093695822217709 … 4.50654080021457e-56 2.2910820458836526e-56;
 [0.039999999999970004 0.017923222803408443 … 9.1019616358618e-55 4.665301655140104e-55; 8
 [0.04999999999995999 0.024206976679911087 … 1.39430186507939e-53 7.204414187441208e-54; 1
 [0.05999999999995 0.030829142611898012 … 1.727813657001802e-52 8.998839052424306e-53; 1.6
 ⋮
 [1.0 0.9951883557459132 … 0.6876434006182843 0.6875742847305666; 0.8132819261323024 0.813
 [1.0 0.995229847689691 … 0.6892669443041967 0.6891978076149408; 0.8161826380342794 0.8160
 [1.0 0.9952577916241877 … 0.690359981837421 0.6902908310610963; 0.818137355358477 0.81804
 [1.0 0.9952744267628608 … 0.6910105297856106 0.6909413705945978; 0.8193014309580468 0.819
 [1.0 0.9952833218955313 … 0.6913583461569452 0.6912891824575133; 0.8199240174520337 0.819
 [1.0 0.9952852379238408 … 0.6914332634710633 0.691364098799908; 0.8200581326963924 0.8199
```

```
tsol=solve(system,times=(0,tend),Δu_opt=0.01,Δt_min=1.0e-4, Δt=1.0e-4)
```

t=





Example: the Brusselator system

Two species interacting via a reaction

$$\partial_t u_1 - \nabla \cdot (D_1 \nabla u_1) + (B+1)u_1 - A - u_1^2 u_2 = 0$$
$$\partial_t u_2 - \nabla \cdot (D_2 \nabla u_2) + u_1^2 u_2 - B u_1 = 0$$

with homogeneous Neumann boundary conditons

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

```
function bruss_storage(f,u,node)
    f[1]=u[1]
    f[2]=u[2]
end
```

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

```
function bruss_diffusion(f,u,edge)
    f[1]=bruss_D_1*(u[1,1]-u[1,2])
    f[2]=bruss_D_2*(u[2,1]-u[2,2])
end
```

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

```
function bruss_reaction(f,u,node)
    f[1]= (B+1.0)*u[1]-A-u[1]^2*u[2]
    f[2]= u[1]^2*u[2]-B*u[1]
end
```

```
ExtendableGrids.ExtendableGrid{Float64, Int32};
dim: 1 nodes: 201 cells: 200 bfaces: 2
```

```
begin
    A=2.25
    B=7.0
    bruss_D_1=0.005
    bruss_D_2=0.1
    pert=0.1
    bruss_tend=50
    dim=1
    if dim==1
        bruss_X=-1:0.01:1
        bruss_grid=simplexgrid(bruss_X)
    else
        bruss_X=-1:0.05:1
        bruss_grid=simplexgrid(bruss_X,bruss_X)
    end
end
```

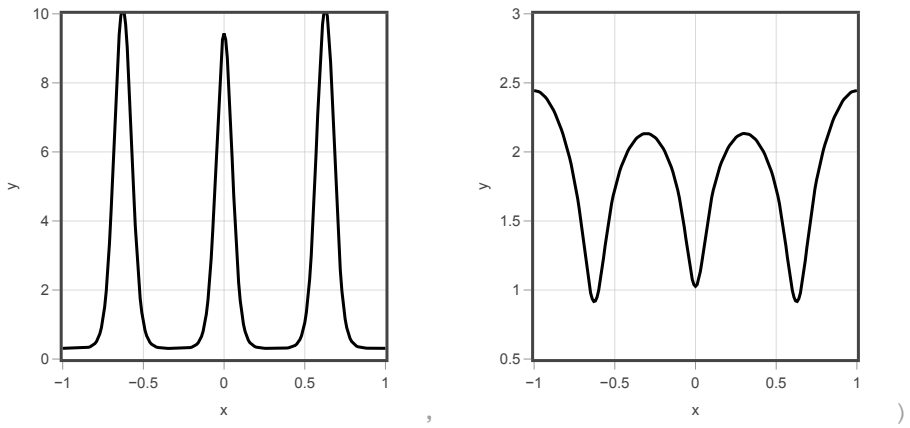
```
bruss_system =
VoronoiFVM.System{Float64, Int32, Int64, Matrix{Int32}, Matrix{Float64}}(num_species=2)
```

```
bruss_system=VoronoiFVM.System(bruss_grid,
            flux=bruss_diffusion,
            storage=bruss_storage,
            reaction=bruss_reaction,
            species=[1,2])
```

```
begin
    inival=unknowns(bruss_system)
    coord=bruss_grid[Coordinates]
    fpeak(x)=exp(-norm(10*x)^2)
    for i=1:size(inival,2)
            inival[1,i]=1.0+0.1*fpeak(coord[:,i])
            inival[2,i]=1.0
    end
    bruss_tsol=solve(bruss_system;inival,times=(0,bruss_tend),
            Δu_opt=0.1,
        Δt=1.0e-4,
        Δt_min=1.0e-6,Δt_max=tend/10,log=true)
end;
```

t= 17.95

```
bruvis =
```



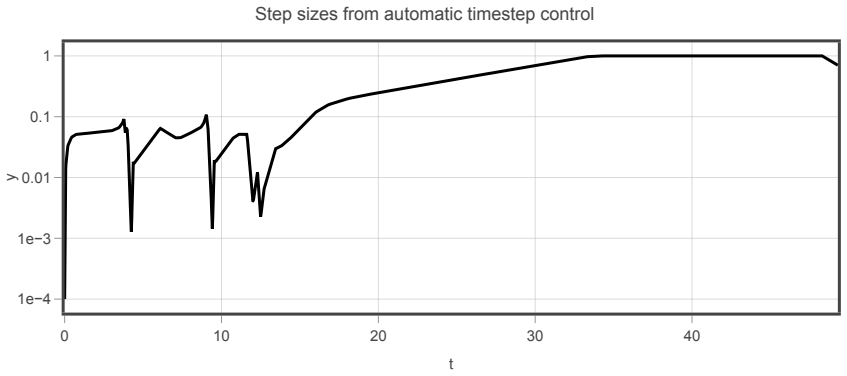

```
bruvis=(GridVisualizer(;size=(300,300),dim=dim),GridVisualizer(;size=(300,300),dim))
```

```
bruss_sol =
2×201 Matrix{Float64}:
 0.31079  0.310789  0.310788  0.310792  …  0.310792  0.310788  0.310789  0.31079
 2.44455  2.44357   2.44063   2.43573      2.43573   2.44063   2.44357   2.44455
```

```
bruss_sol=bruss_tsol(t_bruss)
```

```
scalarplot!(bruvis[1],bruss_grid,bruss_sol[1,:],limits=
(0,10),show=true,colormap=:summer)
```

```
scalarplot!(bruvis[2],bruss_grid,bruss_sol[2,:],limits=
(0.5,3),show=true,colormap=:summer)
```




```
(seconds = 1.47, steps = 797, iters = 2978, maxabsnorm = 9.99e-11, maxrelnorm = 1.06e-7, r
```