

Advanced Topics from Scientific Computing

TU Berlin Winter 2023/24

Notebook 15

 Jürgen Fuhrmann

# Finite Volumes for systems of partial differential equations

---

CairoMakie

```
1 begin
2     using LinearAlgebra,Printf,LaTeXStrings
3     using ExtendableGrids, VoronoiFVM
4     using ExtendableSparse, LinearSolve
5     using GridVisualize
6     using CairoMakie
7     CairoMakie.activate!(type="svg")
8     default_plotter!(CairoMakie)
9 end
```

## 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:

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

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

Similar for nonlinear Robin boundary conditions:

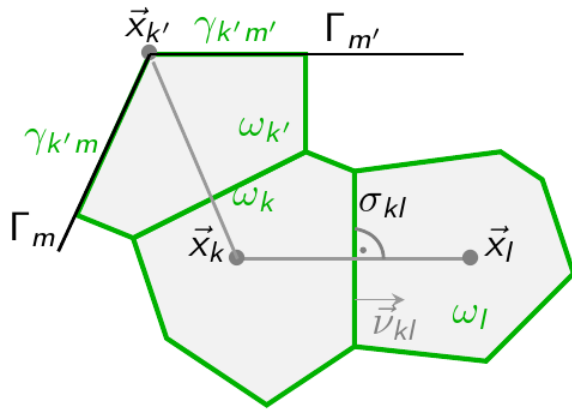
$$\begin{aligned} j_1(\mathbf{u}, \vec{\nabla} \mathbf{u}) \cdot \vec{n} + a_1(\mathbf{u}) &= b_1 \\ &\vdots \\ j_n(\mathbf{u}, \vec{\nabla} \mathbf{u}) \cdot \vec{n} + a_n(\mathbf{u}) &= b_n \end{aligned}$$

or

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

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

## The discrete version



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

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

- $\omega_k$ : control volume
- $\gamma_k$ : boundary interface ( $\emptyset$  for interior nodes)
- $\sigma_{kl}$ : interface between neighboring control volumes
- $h_{kl}$ : distance between neighboring 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  $\mathbf{g} : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ .

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  $\Omega$ ,  $u_1$  reacts to  $u_2$  with forward 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.

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

```
1 begin
2   const kp=1
3   const km=1
4   const D_1=0.5
5   const D_2=0.1
6 end;
7
```

storage (generic function with 1 method)

```
1 function storage(f,u,node)
2   f[1]=u[1]
3   f[2]=u[2]
4 end
5
```

reaction (generic function with 1 method)

```
1 function reaction(f,u,node)
2   r=kp*u[1]-km*u[2]
3   f[1]=r
4   f[2]=-r
5 end
6
```

bcondition (generic function with 1 method)

```

1 function bcondition(f,u,bnode)
2     v=ramp(bnode.time,du=(0,1),dt=(0,1.0e-2))
3     boundary_dirichlet!(f,u,bnode,species=1,region=1,value=v)
4     boundary_neumann!(f,u,bnode,species=1,region=2,value=0)
5     boundary_dirichlet!(f,u,bnode,species=2,region=2,value=0)
6     boundary_neumann!(f,u,bnode,species=2,region=1,value=0)
7 end
8

```

flux (generic function with 1 method)

```

1 function flux(f,u,edge)
2     f[1]=D_1*(u[1,1]-u[1,2])
3     f[2]=D_2*(u[2,1]-u[2,2])
4 end
5

```

```

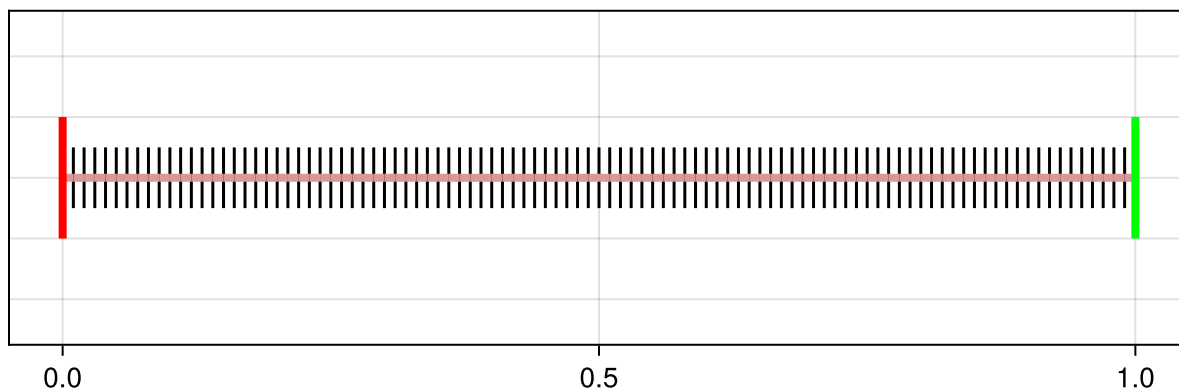
grid = ExtendableGrids.ExtendableGrid{Float64, Int32};
      dim: 1 nodes: 101 cells: 100 bfaces: 2

```

```

1 grid=simplexgrid(0:0.01:1)

```



```

1 gridplot(grid;size=(600,200))

```

```

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

```

```

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

```

```

tend = 10

```

```

1 tend=10

```

```

tsol =
t: 291-element Vector{Float64}:
 0.0
 1.0e-5
 2.2e-5
 3.64e-5
 5.3679999999999994e-5
 7.441599999999999e-5
 9.929919999999999e-5
 ⋮
 6.013788308027948
 6.747503004584986
 7.6279606404534315
 8.418640426968954
 9.209320213484478
10.0
u: 291-element Vector{Matrix{Float64}}:
 [0.0 0.0 ... 0.0 0.0; 0.0 0.0 ... 0.0 0.0]
 [0.001 4.554843410322946e-5 ... 1.5480027823016568e-136 1.4072624633427465e-137; 9.8144727
 [0.0021999999999999997 0.00015908130705043495 ... 1.4009955298788398e-128 1.50105056903607
 [0.0036399999999999996 0.0003701927174498243 ... 5.196095466939298e-121 6.540457552367901e
 [0.0053679999999999995 0.0007170798487162418 ... 1.0109039306116897e-113 1.489440927662297
 [0.0074415999999999999 0.0012480797898037478 ... 1.1121249039591834e-106 1.9100042604902482
 [0.0099299199999999998 0.002023311225746713 ... 7.064161917598479e-100 1.4075189502709288e-
 ⋮
 [1.0 0.9951828455360225 ... 0.6874277376172829 0.6873586244812638; 0.8128968654045824 0.81
 [1.0 0.9952259856410248 ... 0.6891158547224582 0.689046719975269; 0.8159125544273736 0.815
 [1.0 0.9952552929218204 ... 0.6902622563812656 0.6901931068670523; 0.817962531028575 0.817
 [1.0 0.9952708836996275 ... 0.6908719847120692 0.6908028273158894; 0.819053459466289 0.818
 [1.0 0.9952801198208002 ... 0.6912331459324508 0.6911639838573514; 0.8196998777537228 0.81
 [1.0 0.9952855915529846 ... 0.6914470904421581 0.6913779255916498; 0.8200828856140855 0.81

```

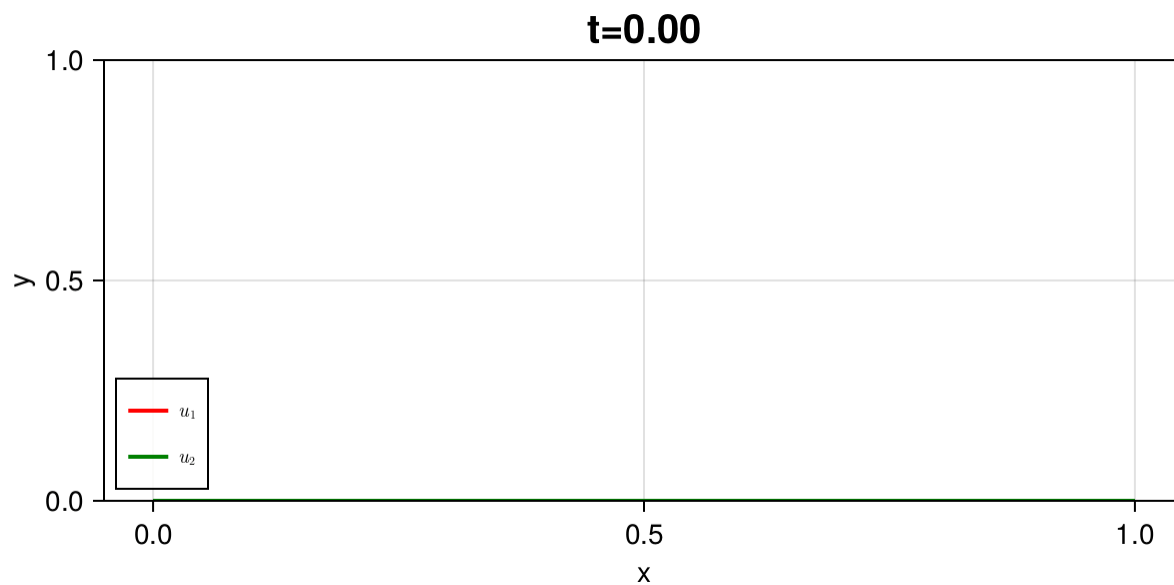
```

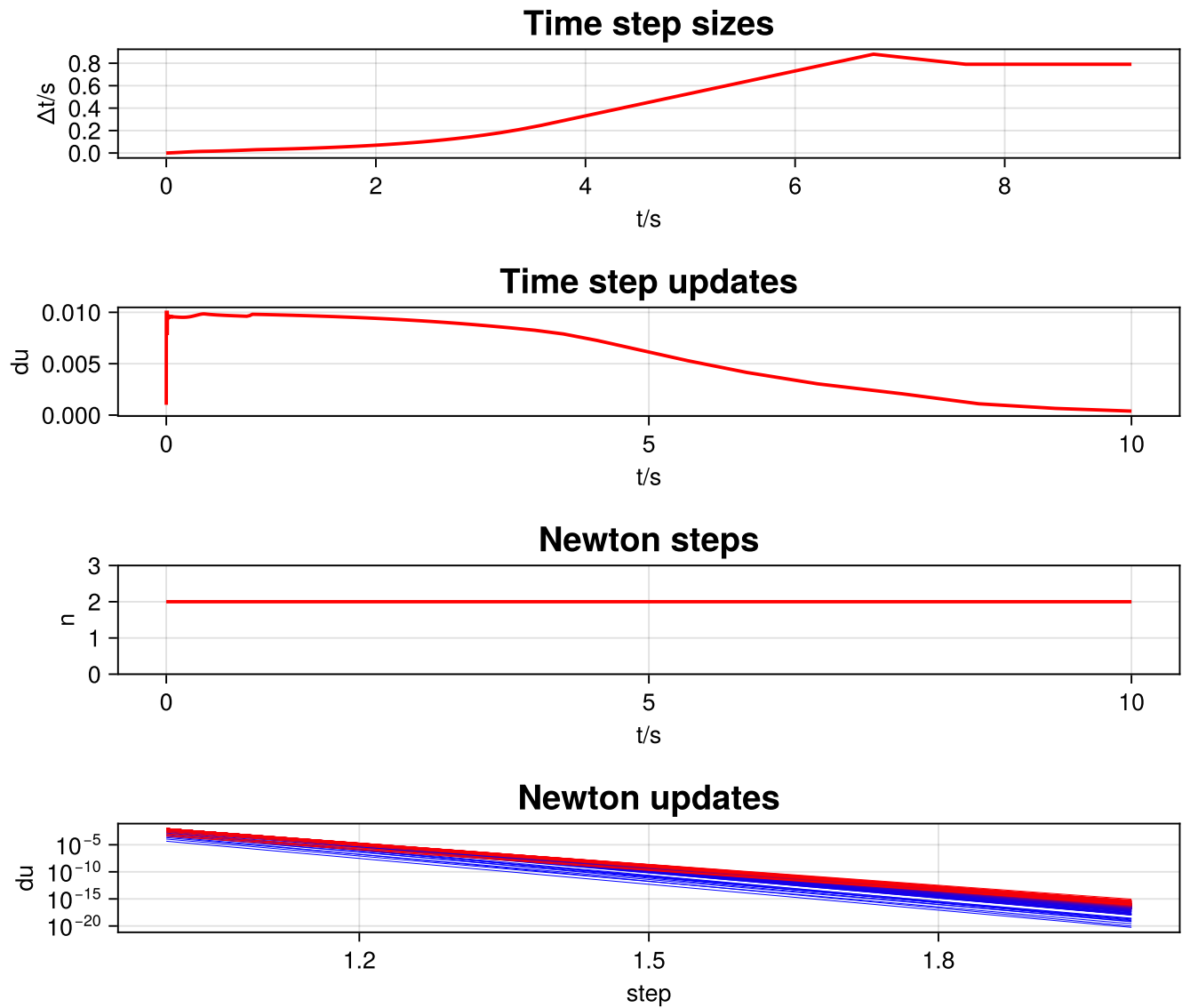
1 tsol=solve(system,times=(0,tend),Δu_opt=0.01,Δt_min=1.0e-5, Δt=1.0e-5,log=true)

```

animatesolution1d (generic function with 1 method)

```
1 function animatesolution1d(system, tsol;
2     video = "tmp.gif",
3     nframes = 201,
4     limits = (0,1),
5     size = (600, 300),
6     pscale_bar = 1.0,
7     Plotter = GridVisualize.default_plotter(),
8     vis = nothing)
9     title = ""
10    vis = GridVisualizer(; size, limits, title, Plotter, legend=:lb)
11    trange = range(extrema(tsol.t)...; length = nframes)
12    movie(vis; file = video) do vis
13        for t in trange
14            title=@sprintf("t=%.2f",t)
15            sol=tsol(t)
16            scalarplot!(vis,system, sol; title,species=1,label=L"u_1", color=:red)
17            scalarplot!(vis,system, sol; species=2,label=L"u_2",
18                color=:green,clear=false)
19            reveal(vis)
20        end
21    end
22    return video
23 end
```





```
1 plothistory(tsol)
```

## Example: the Brusselator system

Two species interacting via a reaction

$$\begin{aligned} \partial_t u_1 - \nabla \cdot (D_1 \nabla u_1) + (B + 1)u_1 - A - u_1^2 u_2 &= 0 \\ \partial_t u_2 - \nabla \cdot (D_2 \nabla u_2) + u_1^2 u_2 - B u_1 &= 0 \end{aligned}$$

with homogeneous Neumann boundary conditions



bruss\_storage (generic function with 1 method)

```
1 function bruss_storage(f,u,node)
2     f[1]=u[1]
3     f[2]=u[2]
4 end
5
```

ExtendableGrids.ExtendableGrid{Float64, Int32};  
dim: 2 nodes: 1681 cells: 3200 bfaces: 160

```
1 begin
2     const A=2.25
3     const B=7.0
4     const bruss_D_1=0.005
5     const bruss_D_2=0.1
6     const pert=0.1
7     const bruss_tend=50
8     const dim=1
9     bruss_X=-1:0.01:1
10    bruss_grid=simplexgrid(bruss_X)
11    bruss_X2=-1:0.05:1
12    bruss_grid2=simplexgrid(bruss_X2,bruss_X2)
13 end
14
```

bruss\_diffusion (generic function with 1 method)

```
1 function bruss_diffusion(f,u,edge)
2     f[1]=bruss_D_1*(u[1,1]-u[1,2])
3     f[2]=bruss_D_2*(u[2,1]-u[2,2])
4 end
5
```

bruss\_reaction (generic function with 1 method)

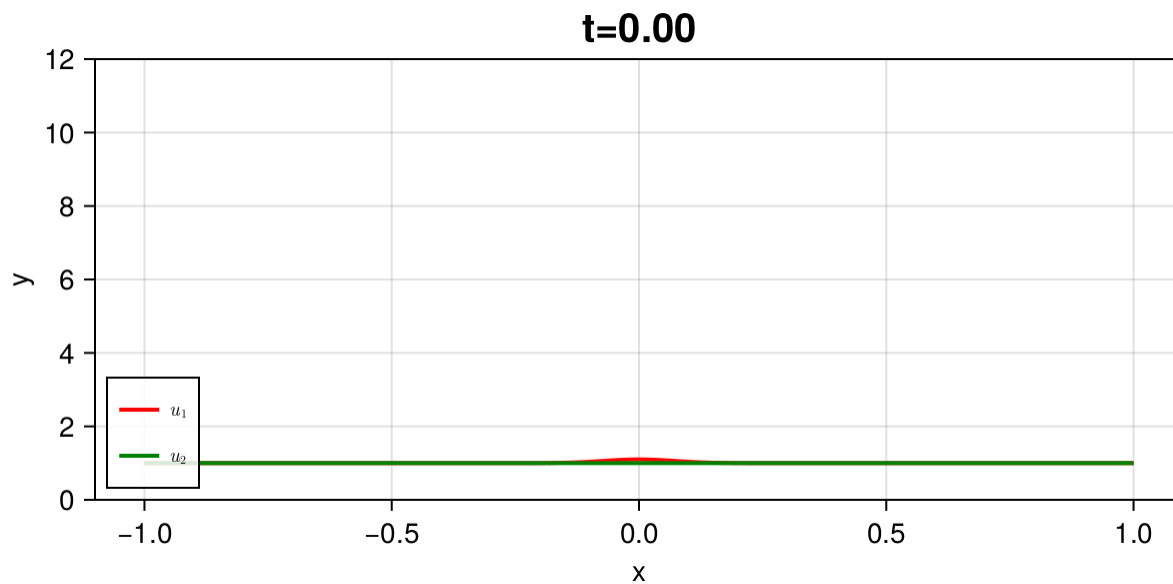
```
1 function bruss_reaction(f,u,node)
2     f[1]= (B+1.0)*u[1]-A-u[1]^2*u[2]
3     f[2]= u[1]^2*u[2]-B*u[1]
4 end
5
```

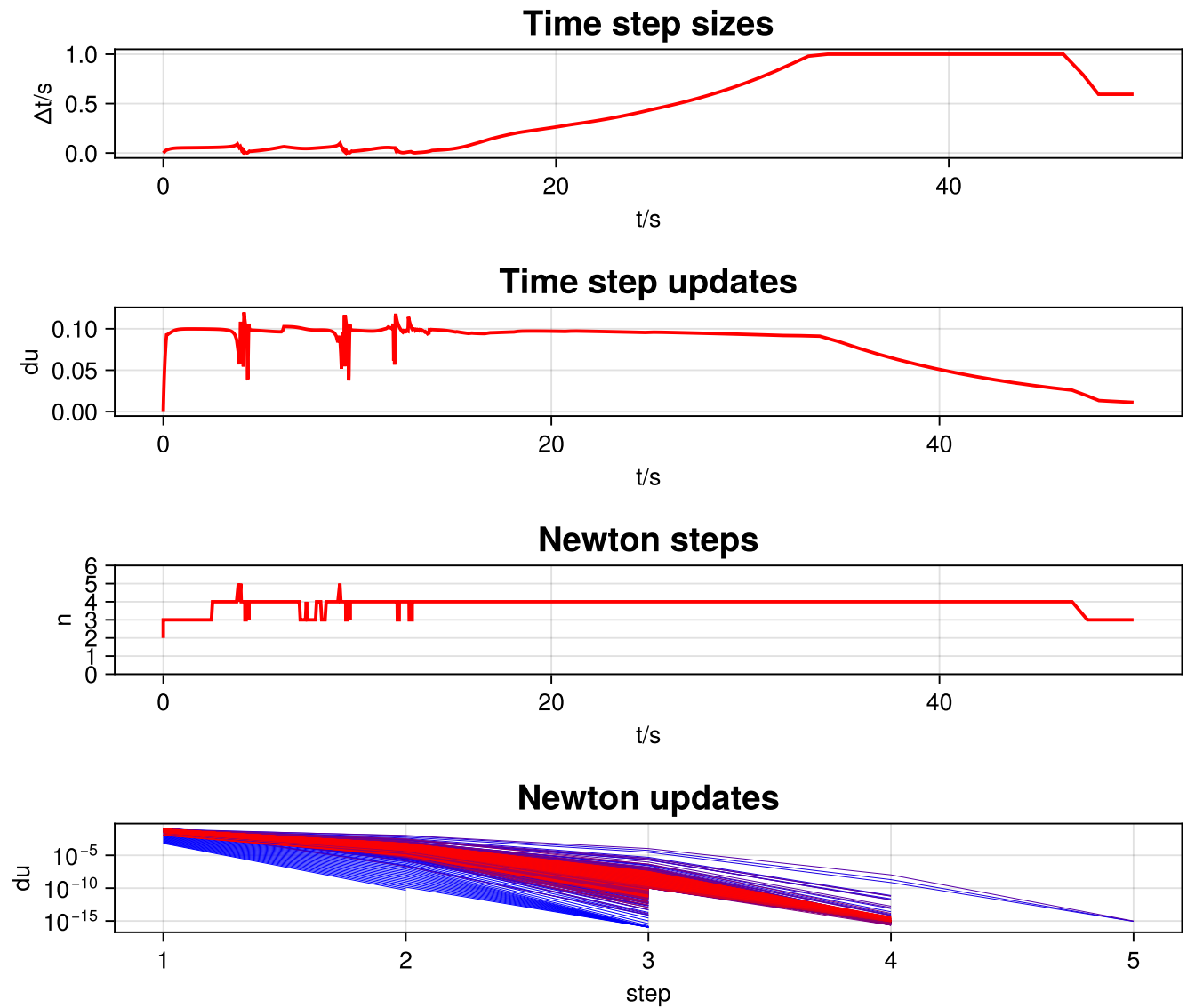
bruss\_system =

VoronoiFVM.System{Float64, Float64, Int32, Int64, Matrix{Int32}, Matrix{Float64}}(num\_spe

```
1 bruss_system=VoronoiFVM.System(bruss_grid,
2     flux=bruss_diffusion,
3     storage=bruss_storage,
4     reaction=bruss_reaction,
5     species=[1,2])
6
```

```
1 begin
2   inival=unknowns(bruss_system)
3   coord=bruss_grid[Coordinates]
4   fpeak(x)=exp(-norm(10*x)^2)
5   for i=1:size(inival,2)
6       inival[1,i]=1.0+0.1*fpeak(coord[:,i])
7       inival[2,i]=1.0
8   end
9   bruss_tsol=solve(bruss_system;inival,times=(0,bruss_tend),
10      Δu_opt=0.1,
11      Δt=1.0e-4,
12      Δt_min=1.0e-6,Δt_max=tend/10,log=true)
13 end;
14
```





```
1 plothistory(bruss_tsol)
```

```
bruss_system2 =
```

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

```
1 bruss_system2=VoronoiFVM.System(bruss_grid2,
2     flux=bruss_diffusion,
3     storage=bruss_storage,
4     reaction=bruss_reaction,
5     species=[1,2])
```

```

1 begin
2   inival2=unknowns(bruss_system2)
3   coord2=bruss_grid2[Coordinates]
4   for i=1:size(inival2,2)
5       inival2[1,i]=1.0+0.1*fpeak(coord2[:,i])
6       inival2[2,i]=1.0
7   end
8   control=SolverControl(BICGstabIteration(ILUZeroPreconditioner(),EquationBlock())
9   ,bruss_system2)
10  control.keeppreconditioner_linear=false
11  bruss_tsol2=solve(bruss_system2;inival2,times=(0,bruss_tend),
12  control,
13      Δu_opt=0.1,
14      reltol=1.0e-10,
15      Δt=1.0e-4,
16      Δt_min=1.0e-6,Δt_max=tend/10,log=true)
17 end;
18

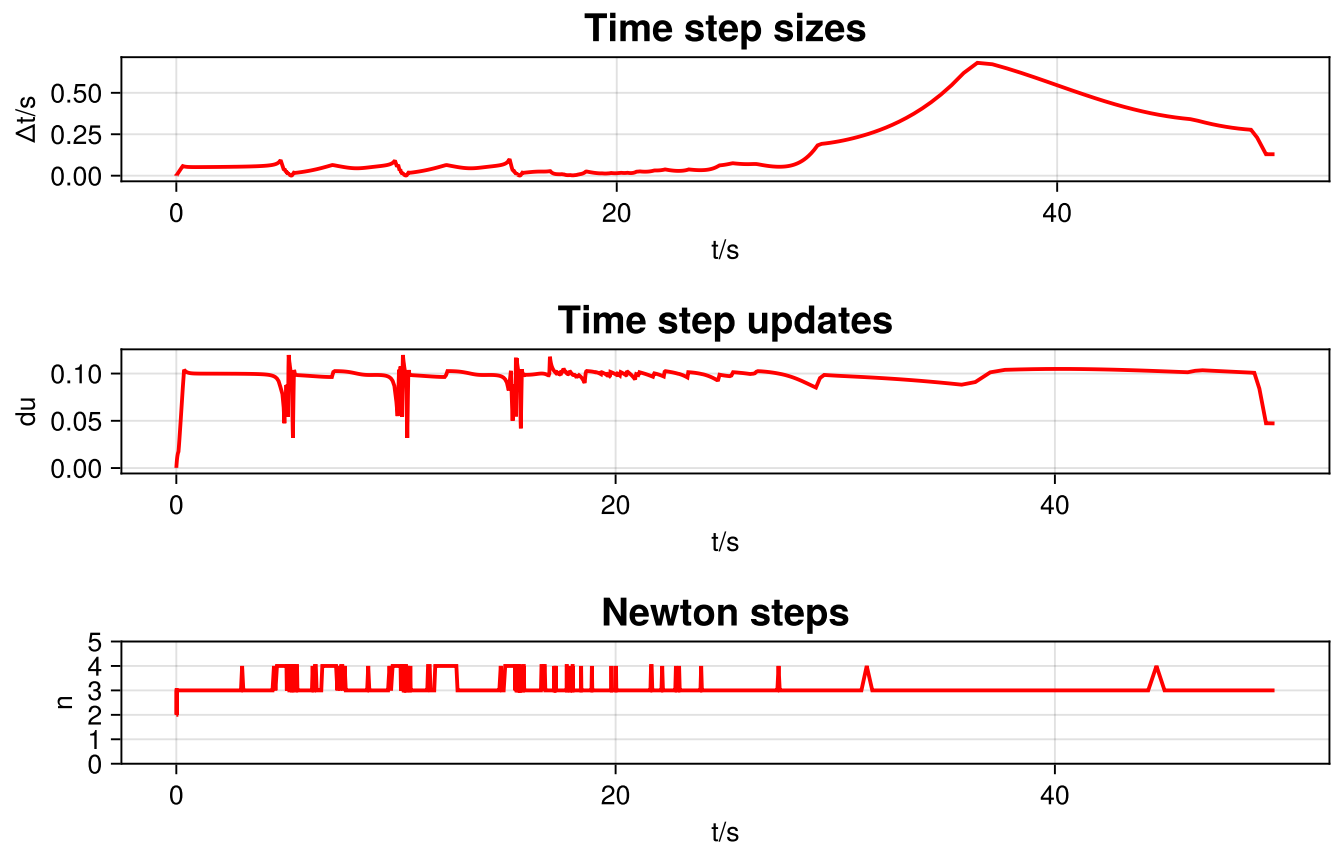
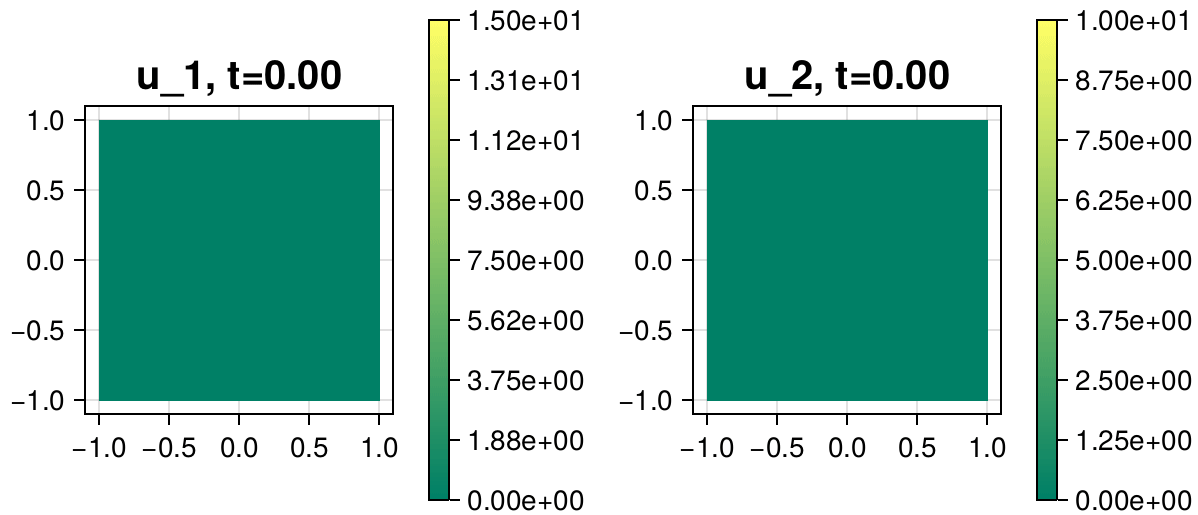
```

animatesolution2d (generic function with 1 method)

```

1 function animatesolution2d(system, tsol;
2     video = "tmp.gif",
3     nframes = 201,
4     size = (600, 300),
5     pscale_bar = 1.0,
6     Plotter = GridVisualize.default_plotter(),
7     vis = nothing)
8     title = ""
9     vis = GridVisualizer(;layout=(1,2), size, title, Plotter)
10    trange = range(extrema(tsol.t)...; length = nframes)
11    movie(vis; file = video) do vis
12        for t in trange
13            sol=tsol(t)
14            title=@sprintf("u_1, t=%.2f",t)
15            scalarplot!(vis[1,1],system, sol; title,species=1,label="u_1",
16            colormap=:summer, limits=(0,15))
17            title=@sprintf("u_2, t=%.2f",t)
18            scalarplot!(vis[1,2],system, sol; title,species=2,label="u_2",
19            colormap=:summer, limits=(0,10))
20            reveal(vis)
21        end
22    end
23    return video
24 end
25

```



```
1 plothistory(bruss_tsol2,plots=[:timestep sizes,:timestep updates,:newton steps])
```

# Table of Contents

## **Finite Volumes for systems of partial differential equations**

A system of reaction-diffusion equations

    The discrete version

Example: a reaction-diffusion problem

Example: the Brusselator system