Advanced Topics from Scientific Computing
TU Berlin Winter 2024/25
Notebook 18
 Jürgen Fuhrmann

CairoMakie

```
begin
    using Test
    using VoronoiFVM, ExtendableGrids
    using OrdinaryDiffEq
    using LinearAlgebra
    using PlutoUI, HypertextLiteral, UUIDs
    using DataStructures
    using GridVisualize
    import CairoMakie
    CairoMakie.activate!(type = "png")
    default_plotter!(CairoMakie)
end
```

```
Precompiling OrdinaryDiffEq...
   3644.9 ms  ? NonlinearSolve
    OrdinaryDiffEqNonlinearSolve Being precompiled by another process (pid:
3365, pidfile: /home/fuhrmann/.julia/compiled/v1.11/OrdinaryDiffEqNonlinearS
olve/jGadk_yYhPp.ji.pidfile)
   7928.8 ms  ✓ OrdinaryDiffEqNonlinearSolve
   9279.8 ms  ? OrdinaryDiffEqIMEXMultistep
  17710.5 ms  ? OrdinaryDiffEqPDIRK
  21971.3 ms  ? OrdinaryDiffEqSDIRK
  30483.5 ms  ? OrdinaryDiffEqStabilizedIRK
  43277.8 ms  ? OrdinaryDiffEqFIRK
  29887.0 ms  ? OrdinaryDiffEqBDF
  15480.8 ms  ? OrdinaryDiffEqDefault
Info Given OrdinaryDiffEq was explicitly requested, output will be shown liv
e
WARNING: Method definition __solve(Union{SciMLBase.NonlinearLeastSquaresProb
lem{var"#s17", iip, var"#s16", F, K} where K where F where var"#s16"<:(Union
{var"#s14", var"#s13"} where var"#s13"<:(AbstractArray{var"#s12", N} where N
where var"#s12"<:ForwardDiff.Dual{T, V, P}) where var"#s14"<:ForwardDiff.Dua
l{T, V, P}) where var"#s17"<:(Union{Number, var"#s15"} where var"#s15"<:(Abs
tractArray{T, N} where N where T)) where P where V where T where iip, SciMLB
ase.NonlinearProblem{var"#s17", iip, var"#s16", F, K, PT} where PT where K w
here F where var"#s16"<:(Union{var"#s14", var"#s13"} where var"#s13"<:(Abstr
actArray{var"#s12", N} where N where var"#s12"<:ForwardDiff.Dual{T, V, P}) w
here var"#s14"<:ForwardDiff.Dual{T, V, P}) where var"#s17"<:(Union{Number, v
ar"#s15"} where var"#s15"<:(AbstractArray{T, N} where N where T)) where P wh
ere V where T where iip}, Nothing, Any...) in module NonlinearSolveBaseForwa
rdDiffExt at /home/fuhrmann/.julia/packages/NonlinearSolveBase/Kek5u/ext/Non
linearSolveBaseForwardDiffExt.jl:124 overwritten in module NonlinearSolve at
/home/fuhrmann/.julia/packages/NonlinearSolve/GHHXX/src/forward_diff.jl:14.
ERROR: Method overwriting is not permitted during Module precompilation. Use
`__precompile__(false)` to opt-out of precompilation.
   6308.5 ms  ? OrdinaryDiffEq
  1 dependency successfully precompiled in 86 seconds. 247 already precompil
ed.
  9 dependencies failed but may be precompilable after restarting julia
  10 dependencies had output during precompilation:
┌ OrdinaryDiffEqFIRK
│   WARNING: Method definition __solve(Union{SciMLBase.NonlinearLeastSquaresP
roblem{var"#s17", iip, var"#s16", F, K} where K where F where var"#s16"<:(Un
ion{var"#s14", var"#s13"} where var"#s13"<:(AbstractArray{var"#s12", N} wher
e N where var"#s12"<:ForwardDiff.Dual{T, V, P}) where var"#s14"<:ForwardDif
f.Dual{T, V, P}) where var"#s17"<:(Union{Number, var"#s15"} where var"#s15"
<:(AbstractArray{T, N} where N where T)) where P where V where T where iip,
```

Table of Contents


```
TableOfContents()
```

1D Nonlinear Diffusion

Solve the nonlinear diffusion equation

$$\partial_t u - \Delta u^m = 0$$

in $\Omega = (-1, 1)$ with homogeneous Neumann boundary conditions.

This equation is also called "porous medium equation".

For space dimension d in the domain $\mathbb{R}^d \times (0, \infty)$ the equation has a radially symmetrict exact solution, the so-called Barenblatt solution:

$$b(x,t) = \max\left(0, t^{-\alpha}\left(1 - \frac{\alpha(m-1)|x|^2}{2dmt^{\frac{2\alpha}{d}}}\right)^{\frac{1}{m-1}}\right)$$

Here, $\alpha = \frac{1}{m-1+\frac{2}{d}}$. (see Barenblatt, G. I. "On nonsteady motions of gas and fluid in porous medium." Appl. Math. and Mech.(PMM) 16.1 (1952): 67-78.)

We initialize this problem with the exact solution for $t = t_0 = 0.001$.

```
function barenblatt(x, t, m)
    t1 = t^(-1.0 / (m + 1.0))
    x1 = 1 - (x * t1)^2 * (m - 1) / (2.0 * m * (m + 1))
    return x1 < 0.0 ? 0.0 : t1 * x1^(1.0 / (m - 1.0))
end;
```

```
0.01
```

```
begin
    const m = 2
    const ε = 1.0e-10
    const n = 50
    const t0 = 1.0e-3
    const tend = 1.0e-2
end
```

```
X = -1.0:0.04:1.0
```

```
X = range(-1, 1, length = n + 1)
```

```
grid = ExtendableGrids.ExtendableGrid{Float64, Int32}
           dim =       1
         nnodes =      51
         ncells =      50
        nbfaces =       2
```

```
grid = simplexgrid(X)
```

```
nld_flux! (generic function with 1 method)
```

```
function nld_flux!(f, u, edge, data)
    f[1] = u[1, 1]^m - u[1, 2]^m
    return nothing
end
```

```
nld_storage! (generic function with 1 method)
```

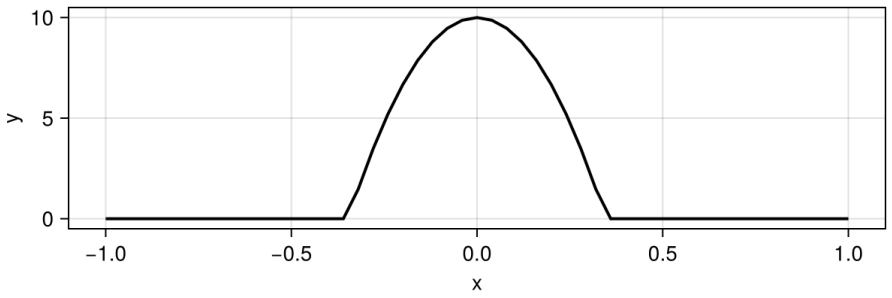
```
function nld_storage!(f, u, node, data)
    f[1] = u[1]
    return nothing
end
```

```
nld_fvm_sys =
VoronoiFVM.System{Float64, Float64, Int32, Int64, Matrix{Int32}}(
  grid = ExtendableGrids.ExtendableGrid{Float64, Int32}(dim=1, nnodes=51, ncells=50,
  nbfaces=2),
  physics = Physics(flux=nld_flux!, storage=nld_storage!, ),
  num_species = 1)
```

```
nld_fvm_sys = VoronoiFVM.System(grid, storage = nld_storage!, flux = nld_flux!,
  species = [1])
```

```
▸ view(::VoronoiFVM.DenseSolutionArray{Float64, 2}, 1, :): [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
```

```
begin
    nld_inival = unknowns(nld_fvm_sys)
    nld_inival[1, :] .= barenblatt.(X, t0, m)
end
```




```
scalarplot(grid, nld_inival[1, :], size = (600, 200))
```

```
nld_fvm_sol =
t: 109-element Vector{Float64}:
 0.001
 0.0010001
 0.00100022
 0.0010003640000000001
 0.0010005368
 0.00100074416
 0.001000992992
 ⋮
 0.008695060666880321
 0.009021295500160242
 0.00934753033344016
 0.009565020222293441
 0.009782510111146722
 0.01
u: 109-element Vector{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]
 ⋮
 [9.286214107895102e-227 2.724603587778204e-111 … 2.7246035877781957e-111 9.2862141078
 [2.667309166170285e-193 1.6739667796753464e-95 … 1.6739667796753448e-95 2.66730916617
 [3.469942273025066e-166 2.3030461318477954e-82 … 2.3030461318478106e-82 3.46994227302
 [1.4087541681634793e-155 7.722602656643459e-78 … 7.722602656643491e-78 1.408754168163
 [1.170009228469437e-143 7.268450326335965e-72 … 7.268450326335979e-72 1.1700092284694
 [5.429642603991764e-132 4.816385223877479e-66 … 4.816385223877502e-66 5.42964260399181
```

```
nld_fvm_sol = solve(nld_fvm_sys; inival = nld_inival, times = (t0, tend), Δt =
  1.0e-7, Δt_min = 1.0e-7, log = true)
```

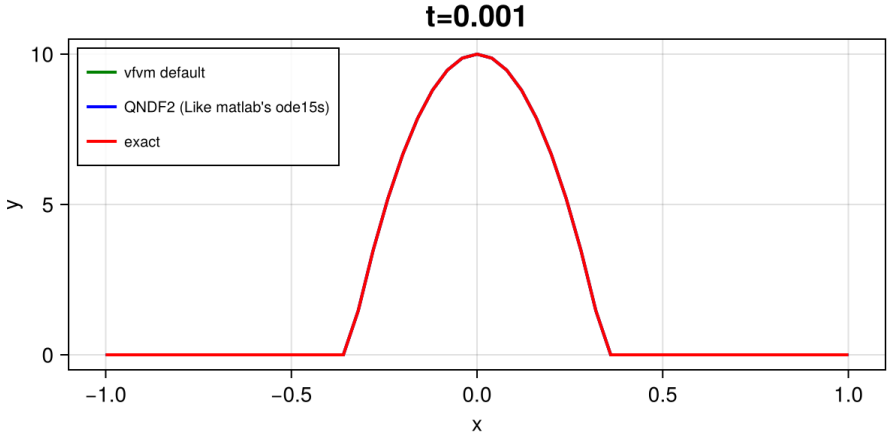
Implementation with ODE solvers from DifferentialEquations.jl

```
diffeqmethods =
▸ OrderedDict("QNDF2 (Like matlab's ode15s)" ⇒ QNDF2, "Rodas5" ⇒ Rodas5, "Rosenbrock23 (
```

method: QNDF2 (Like matlab's ode15s)

```
t: 36-element Vector{Float64}:
 0.001
 0.001001
 0.0010019999999999999
 0.0010119999999999999
 0.0010849458904049366
 0.0011578917808098732
 0.0012715147381319038
 ⋮
 0.0074832250250284876
 0.007942684428895293
 0.008450867960826622
 0.009009382156943125
 0.009567896353059628
 0.01
u: 36-element Vector{VoronoiFVM.DenseSolutionArray{Float64, 2}}:
 [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 1.594181423962512e-229 … 1.594181423962513e-229 0.0]
 [0.0 2.7946725005967386e-213 … 2.7946725005967386e-213 0.0]
 [0.0 3.2659064627353536e-174 … 3.2659064627353437e-174 0.0]
 [4.429365162062866e-267 3.385419378952872e-126 … 3.385419378952872e-126 4.42936516206
 [3.623366872623562e-182 8.406357665692234e-84 … 8.406357665692234e-84 3.623366872623
 [9.096594029046456e-161 6.079980857755867e-78 … 6.079980857755867e-78 9.096594029046
```

```
begin
    nld_ode_sys = VoronoiFVM.System(grid, storage = nld_storage!, flux =
nld_flux!, species = [1])
    nld_problem = ODEProblem(nld_ode_sys, nld_inival, (t0, tend))
    odesol = solve(
        nld_problem,
        diffeqmethods[nld_method](),
        adaptive = true,
        reltol = 1.0e-3,
        abstol = 1.0e-3,
        initializealg = NoInit()
    )
    nld_ode_sol = reshape(odesol, nld_ode_sys)
end
```



0.001

```
@bind nld_time Slider(range(t0, tend, length = 101), show_value = true)
```

1D Nonlinear Storage

This equation comes from the transformation of the nonlinear diffusion equation

$$\partial_t v - \Delta v^m = 0$$

to

$$\partial_t u^{\frac{1}{m}} - \Delta u = 0$$

in $\Omega = (-1, 1)$ with homogeneous Neumann boundary conditions. We can derive an exact solution from the Barenblatt solution of the equation for u.

```
u0 =
```

```
▸ [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, 2.151
```

```
1 u0 = map(x -> barenblatt(x, t0, m)^m, X)
```

Direct implementation with VoronoiFVM

```
nls_flux! (generic function with 1 method)
```

```
1 function nls_flux!(f, u, edge, data)
2     f[1] = u[1, 1] - u[1, 2]
3     return nothing
4 end
```

Storage term needs to be regularized as its derivative at 0 is infinity:

```
nls_storage! (generic function with 1 method)
```

```
1 function nls_storage!(f, u, node, data)
2     f[1] = (ε + u[1])^(1.0 / m)
3     return nothing
4 end
```

```
▸ (seconds = 0.861, tasm = 0.648, tlinsolve = 0.21, steps = 731, iters = 2920, maxabsnorm = 1.73
```

```
1 begin
2     nls_sys = VoronoiFVM.System(grid; flux = nls_flux!, storage = nls_storage!,
  species = [1])
3     nls_inival = unknowns(nls_sys)
4     nls_inival[1, :] .= u0
5     nls_sol = VoronoiFVM.solve(nls_sys; inival = nls_inival, times = (t0, tend),
  Δt_min = 1.0e-4, Δt = 1.0e-4, Δu_opt = 0.1, force_first_step = true, log = true)
6     history_summary(nls_sol)
7 end
```

Implementation as DAE

If we want to solve the problem with standard ODE solvers, we see that the problem structure does not fit into the setting of that package due to the nonlinearity under the time derivative. Here we propose a reformulation to a DAE as a way to achieve this possibility:

$$\begin{cases} \partial_t w - \Delta u & = 0 \\ w^m - u & = 0 \end{cases}$$

```
dae_storage! (generic function with 1 method)
```

```
1 function dae_storage!(y, u, node, data)
2     y[1] = u[2]
3     return nothing
4 end
```

```
dae_reaction! (generic function with 1 method)
```

```
1 function dae_reaction!(y, u, node, data)
2     y[2] = u[2]^m - u[1]
3     return nothing
4 end
```

First, we test this with the implicit Euler method of VoronoiFVM

▸ (seconds = 0.611, tasm = 0.505, tlinsolve = 0.103, steps = 732, iters = 2205, maxabsnorm = 9.7

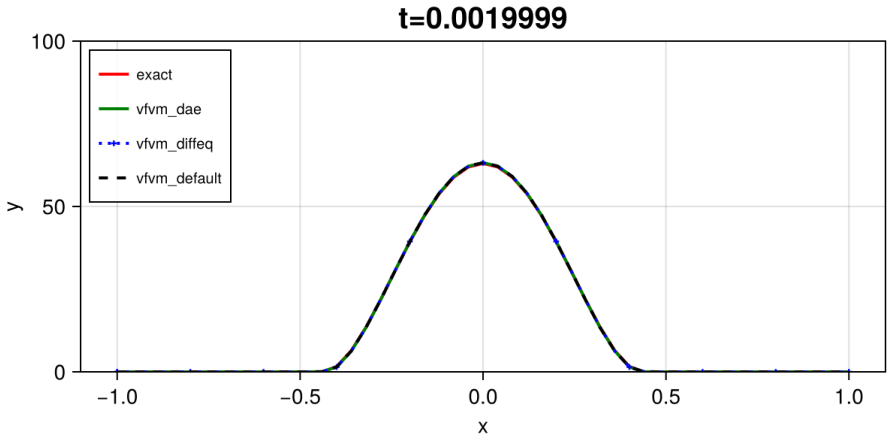
```
begin
    dae_sys = VoronoiFVM.System(
        grid; flux = nls_flux!, storage = dae_storage!,
        reaction = dae_reaction!, species = [1, 2]
    )
    dae_inival = unknowns(dae_sys)
    dae_inival[1, :] .= u0
    dae_inival[2, :] .= u0 .^ (1 / m)
    dae_control = VoronoiFVM.SolverControl()
    dae_sol = VoronoiFVM.solve(dae_sys; inival = dae_inival, times = (t0, tend), Δt_min = 1.0e-4, Δt = 1.0e-4, Δu_opt = 0.1, force_first_step = true, log = true)
    history_summary(dae_sol)
end
```

Implementation via OrdinaryDiffEq.jl

method: QNDF2 (Like matlab's ode15s)

```
begin
    dae_ode_sys = VoronoiFVM.System(grid; flux = nls_flux!, storage = dae_storage!, reaction = dae_reaction!, species = [1, 2])

    dae_ode_problem = ODEProblem(dae_ode_sys, dae_inival, (t0, tend))
    odesol2 = solve(
        dae_ode_problem,
        diffeqmethods[method](),
        adaptive = true,
        reltol = 1.0e-3,
        abstol = 1.0e-3,
        initializealg = NoInit()
    )
    dae_ode_sol = reshape(odesol2, dae_ode_sys)
end;
```



t= 0.0019999

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

```
function plotsolutions(t)
    vis = GridVisualizer(resolution = (600, 300), dim = 1, Plotter = CairoMakie, legend = :lt)
    u = nls_sol(t)
    u_dae = dae_sol(t)
    u_de = dae_ode_sol(t)
    scalarplot!(vis, X, map(x -> barenblatt(x, t, m) .^ m, X), clear = true, color = :red, linestyle = :solid, flimits = (0, 100), label = "exact")
    scalarplot!(vis, grid, u_dae[1, :], clear = false, color = :green, linestyle = :solid, label = "vfvm_dae")
    scalarplot!(vis, grid, u_de[1, :], clear = false, color = :blue, markershape = :cross, linestyle = :dot, label = "vfvm_diffeq")
    scalarplot!(vis, grid, u[1, :], clear = false, color = :black, markershape = :none, linestyle = :dash, title = "t=$(t)", label = "vfvm_default")
    return reveal(vis)
end
```

Brusselator system

Two species diffusing and interacting via a reaction

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

```
begin
    const bruss_A = 2.25
    const bruss_B = 7.0
    const bruss_D_1 = 0.025
    const bruss_D_2 = 0.25
    const pert = 0.1
    const bruss_tend = 150
end;

```

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

```

```
function bruss_diffusion(f, u, edge, data)
    f[1] = bruss_D_1 * (u[1, 1] - u[1, 2])
    f[2] = bruss_D_2 * (u[2, 1] - u[2, 2])
    return nothing
end;

```

```
function bruss_reaction(f, u, node, data)
    f[1] = (bruss_B + 1.0) * u[1] - bruss_A - u[1]^2 * u[2]
    f[2] = u[1]^2 * u[2] - bruss_B * u[1]
    return nothing
end;

```

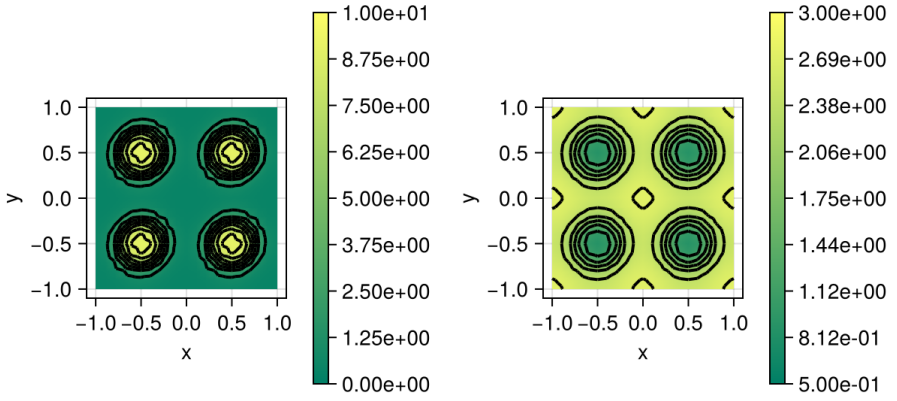
```
begin
    bruss_X = -1:0.1:1
    bruss_grid = simplexgrid(bruss_X, bruss_X)
    bruss_system = VoronoiFVM.System(
        bruss_grid, species = [1, 2],
        flux = bruss_diffusion, storage = bruss_storage, reaction = bruss_reaction
    )
    bruss_inival = unknowns(bruss_system, inival = 0)
    coord = bruss_grid[Coordinates]
    fpeak(x) = exp(-norm(10 * x)^2)
    for i in 1:size(bruss_inival, 2)
        bruss_inival[1, i] = fpeak(coord[:, i])
        bruss_inival[2, i] = 0
    end

end
```

```
begin
    bruss_ode_problem = ODEProblem(bruss_system, bruss_inival, (0, bruss_tend))
    odesol3 = solve(
        bruss_ode_problem,
        diffeqmethods[bruss_method](),
        adaptive = true,
        reltol = 1.0e-3,
        abstol = 1.0e-3,
        initializealg = NoInit()
    )
    bruss_tsol = reshape(odesol3, bruss_system)
end;
```

method: QNDF2 (Like matlab's ode15s)

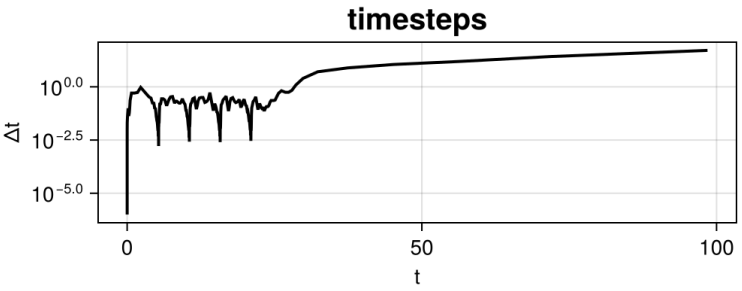
t: 150.0




```
let
    bruss_sol = bruss_tsol(t_bruss)

    vis = GridVisualizer(; layout = (1, 2), size = (600, 300))
    scalarplot!(vis[1, 1], bruss_grid, bruss_sol[1, :], limits = (0, 10), show = true, colormap = :summer)
    scalarplot!(vis[1, 2], bruss_grid, bruss_sol[2, :], limits = (0.5, 3), show = true, colormap = :summer)
end

```




```
scalarplot(bruss_tsol.t[1:(end - 1)], bruss_tsol.t[2:end] - bruss_tsol.t[1:(end - 1)], yscale = :log, resolution = (500, 200), xlabel = "t", ylabel = "Δt", title = "timesteps")
```