```
begin
    ENV["LANG"]="C"
    using Pkg
    Pkg.activate(mktempdir())
    Pkg.add(["PyPlot","PlutoUI","ExtendableGrids","GridVisualize","VoronoiFVM"])
    using PlutoUI,PyPlot,ExtendableGrids,VoronoiFVM,GridVisualize
    PyPlot.svg(true)
end;
```

Working with VoronoiFVM.jl

We show how to define scalar linear and nonlinear diffusion problems in the VoronoiFVM package.

In general, the package allows to work with multiple species, so all constitutive functions and the solution array need to handle species indices.

For more information, see its **documentation**.

Linear diffusion problem with Dirichlet boundary conditions

Regard

$$\begin{aligned} -\nabla \cdot (D\vec{\nabla}u) &= f \quad \text{in } \Omega \\ u &= g \text{ on } \partial\Omega \end{aligned}$$

The following data characterize the problem:

- Flux $\vec{j} = -D\vec{\nabla}u$
- Dirichlet data g
- Source/sink term f
- Domain Ω

These can be replaced by the following discretization data:

Diffusion coefficient D

```
D = 10.0
```

```
D=10.0
```

Diffusion flux $g(u_k, u_L) = D(u_k - u_l)$.

The following function receives the parameters in `_u`. This is an array containing the unkowns u_k, u_l and possibly several more parameters. The `unknowns` method extracts the unknowns from `_u`, creatimg a two-dimensional array, where the first parameter is the species index and the second the local number of the node wrt. the edge. The result is written into `f` for species index 1.

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

```
function diffusion_flux!(f,_u, edge)
    u=unknowns(edge,_u)
    spec_idx=1
    f[spec_idx]=D*(u[spec_idx,1]-u[spec_idx,2])
end
```

Right hand side function $f(x) = 1$ (just for an example). Once again, the species index is 1.

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

```
function diffusion_source!(f,node)
    f[1]=1
end
```

Discretization grid

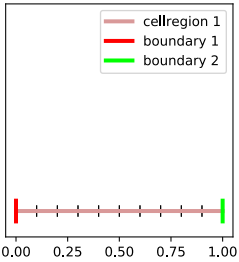
Grid in domain $\Omega = (0,1)$ consisting of N=11 points.

```
X =  Float64[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
```

```
X=collect(range(0,1,length=N))
```

```
grid1d = ExtendableGrids.ExtendableGrid{Float64,Int32};
         dim: 1 nodes: 11 cells: 10 bfaces: 2
```

```
grid1d=simplexgrid(X)
```




```
gridplot(grid1d,Plotter=PyPlot,resolution=(600,200))
```

System creation and solution methods

The problem description is subdivided in two parts: the discretization grid which is used to define the Voronoi cells and the form factors, and the physics part containing the constitutive functions, like the fluxes between neigboring species, the source term, and, possibly, more.

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

```
function create_system(grid,flux, source)
    physics=VoronoiFVM.Physics(num_species=1,flux=flux,source=source) # physics object
    system=VoronoiFVM.System(grid,physics) # system object
    enable_species!(system,1,[1])
    system
end
```

After setting boundary conditions, we can solve the system. Dirichlet boundary conditions are implemented by the penalty method.

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

```
function mysolve(system, ul,ur; bcl=1, bcr=2)
    boundary_dirichlet!(system, 1, bcl,ul) # Set left Dirichlet boundary conditions
    boundary_dirichlet!(system, 1, bcr,ur) # Set right Dirichlet boundary conditions
    inival=unknowns(system,inival=0.5*(ul+ur)) # constant initial value
    VoronoiFVM.solve(inival,system,log=true) # Return a pair (solution, history) when log=true
end
```

Set the Dirichlet boundary data:

- u_L= 0.01
- u_R= 0.01

1D Linear diffusion

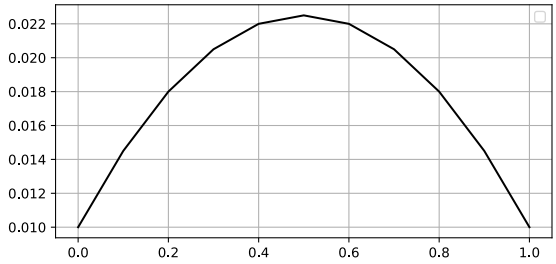
```
system1d = VoronoiFVM.DenseSystem{Float64,Int32,Int32}(num_species=1)
```

```
system1d=create_system(grid1d,diffusion_flux!,diffusion_source!)
```

Using default settings, the system is solved.

```
(1×11 Array{Float64,2}:                                                , Float6
  0.01  0.0145  0.018  0.0205  0.022  0.0225  0.022  0.0205  0.018  0.0145  0.01
```

```
(solution,history)=mysolve(system1d,u_L,u_R)
```



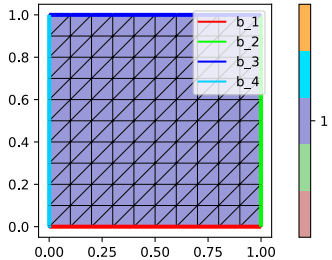

```
scalarplot(grid1d,solution[1,:],Plotter=PyPlot,resolution=(600,300))
```

2D Linear diffusion

For solving a 2D problem, we just need to replace the 1D grid with a 2D grid.

```
grid2d = ExtendableGrids.ExtendableGrid{Float64,Int32};
         dim: 2 nodes: 121 cells: 200 bfaces: 40
```

```
grid2d=simplexgrid(X,X)
```




```
gridplot(grid2d,Plotter=PyPlot)
```

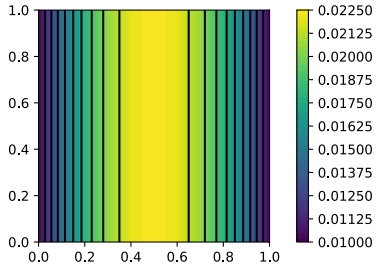
```
system2d = VoronoiFVM.DenseSystem{Float64,Int32,Int32}(num_species=1)
```

```
system2d=create_system(grid2d,diffusion_flux!,diffusion_source!)
```

By default, the left boundary is marked by 4 and the right boundary is marked by 2, and we pass the data this way.

```
(1×121 Array{Float64,2}:                                                , Flo
  0.01  0.0145  0.018  0.0205  0.022  0.0225  …  0.022  0.0205  0.018  0.0145  0.01
```

```
solution2d,history2d=mysolve(system2d,u_L,u_R,bcl=4, bcr=2)
```




```
scalarplot(grid2d,solution2d[1,:],Plotter=PyPlot,resolution=(300,300),fignumber=2)
```

Nonlinear diffusion

Here, we define a nonlinear diffusion problem:

Let $\vec{j} = -D(u)\vec{\nabla}u$ with $D(u) = 2u$.

Then $\mathcal{D}(u) = \int_0^u D(\xi)d\xi = u^2$.

So we can define $g(u_k, u_l) = u_k^2 - u_l^2$

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

```
function nldiffusion_flux!(f,_u, edge)
    u=unknowns(edge,_u)
    f[1]=u[1,1]^2-u[1,2]^2
end
```

1D Nonlinear diffusion

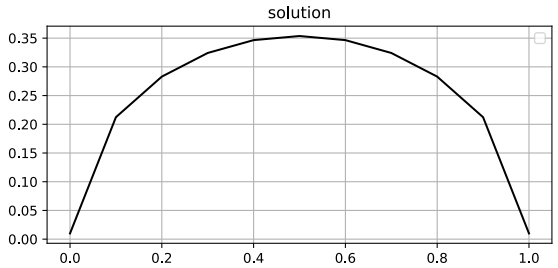
```
nlsystem1d = VoronoiFVM.DenseSystem{Float64,Int32,Int32}(num_species=1)
```

```
nlsystem1d=create_system(grid1d,nldiffusion_flux!,diffusion_source!)
```

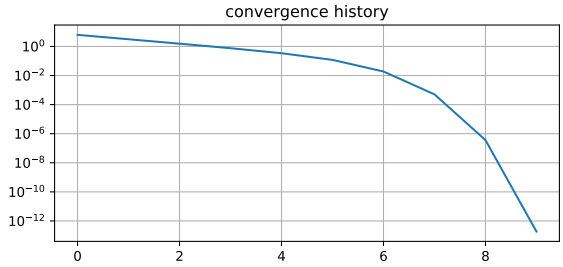
Here, Newton's method is used in order to solve the nonlinear system of equations. The Jacobi matrix is assembled from the partial derivatives of the flux function $g(u_k, u_l)$.

```
1×11 Array{Float64,2}:
 0.01  0.212368  0.283019  0.324191  0.346554  …  0.324191  0.283019  0.212368  0.01
  Float64[6.25,  3.12001,  1.55008,  0.755617,  0.342177,  0.118959,  0.0189612,  0.000507516,
```

```
(nlsolution1d,nlhistory1d)=mysolve(nlsystem1d,u_L,u_R)
```




```
scalarplot(grid1d,nlsolution1d[1,:],Plotter=PyPlot,resolution=
(600,300),title="solution")
```




```
let
    clf()
    semilogy(nlhistory1d)
    title("convergence history")
    grid()
    gcf().set_size_inches(6,3)
    gcf()
end
```

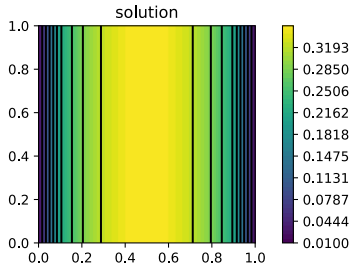
2D Nonlinear diffusion

```
nlsystem2d = VoronoiFVM.DenseSystem{Float64,Int32,Int32}(num_species=1)
```

```
nlsystem2d=create_system(grid2d,nldiffusion_flux!,diffusion_source!)
```

```
(1×121 Array{Float64,2}:
  0.01  0.212368  0.283019  0.324191  0.346554  …  0.324191  0.283019  0.212368  0.01
```

```
(nlsolution2d,nlhistory2d)=mysolve(nlsystem2d,u_L,u_R,bcl=4, bcr=2)
```




```
scalarplot(grid2d,nlsolution2d[1,:],Plotter=PyPlot,resolution=
(300,300),fignumber=2,title="solution")
```

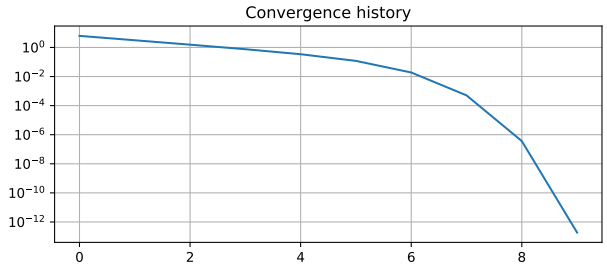


Table of Contents

```
Status `/tmp/jl_Vy3imp/Project.toml`
  [cfc395e8] ExtendableGrids v0.7.4
  [5eed8a63] GridVisualize v0.1.2
  [7f904dfe] PlutoUI v0.7.1
  [d330b81b] PyPlot v2.9.0
  [82b139dc] VoronoiFVM v0.10.3
```