

# Gridap: introduction for FluidDynamics

---

## About Gridap

---

Project started in 2020, official [Gridap](#) repository

What:

- Written in Julia
- Provides a set of tools for solving PDEs
- FEM framework
- Almost 1:1 mathematical notation

Support for:

- MPI, [GridapDistributed](#)
- PETSc, [GridapPETSc](#)
- Pardiso, [GridapPardiso](#)
- Gmsh, [GridapGmsh](#)

Features:

- MultiField problems
- Time depended
- Discontinuous Galerkin
- Raviart Thomas elements

## WorkFlow:

---

1. Mesh creation with Gmsh
2. Julia-MPI-Gridap
3. Open the results in [Paraview](#)

# Solve the Poisson equation in 3D

## Problem

$$\begin{aligned} -\Delta u &= f \text{ in } \Omega, \\ u &= g \text{ on } \Gamma_D, \\ \nabla u \cdot n &= h \text{ on } \Gamma_N, \end{aligned}$$

We choose  $f(x) = 1$ ,  $g(x) = 2$ ,  $h(x) = 3$

```
• f(x) = 1.0
```

```
• g(x) = 2.0;
```

```
• h(x) = 3.0;
```

Find  $u \in U_h$  such that  $a(u, v) = b(v)$  for all  $v \in V_h$

where  $a(u, v) \doteq \int_{\Omega} \nabla v \cdot \nabla u \, d\Omega$ ,  $b(v) \doteq \int_{\Omega} v f \, d\Omega + \int_{\Gamma_N} v h \, d\Gamma_N$

```
• using Gridap ↻
```

```
• using GridapGmsh ↻
```

```
• msh_file = joinpath(@__DIR__, "models", "toy_model.msh")
```

```
• model = GmshDiscreteModel(msh_file)
```

```
• order = 1;
```

```
• reffe = ReferenceFE(lagrangian,Float64,order);
```

```
• V = TestFESpace(model,reffe,dirichlet_tags=["sides"]);
```

```
• U = TrialFESpace(V,g);
```

```
• neumann_tags = ["circle", "triangle", "square"];
```

```
• Γ = BoundaryTriangulation(model,tags=neumann_tags);
```

Set up to perform the integrals in the weak form numerically. Define integration mesh, plus a Gauss-like quadrature in each of the cells in the triangulation

```
• degree = 2;
```

```
• dΓ = Measure(Γ, degree);
```

```
• Ω = Triangulation(model);
```

```
• dΩ = Measure(Ω, degree);
```

Weak form

$$a(u, v) \doteq \int_{\Omega} \nabla v \cdot \nabla u \, d\Omega$$

```
• a(u, v) = ∫(∇(u)•∇(v))dΩ ;
```

$$b(v) \doteq \int_{\Omega} v f \, d\Omega + \int_{\Gamma_N} v h \, d\Gamma_N$$

```
• b(v) = ∫( v*f )*dΩ + ∫( v*h )*dΓ ;
```

```
• op = AffineFEOperator(a, b, U, V) ;
```

Linear Solver

```
• ls = LUSolver();
```

There are other solvers available:

- BackSlash(), like \ in MATLAB
- NLSolver()
- PETSc

```
• solver = LinearFESolver(ls) ;
```

```
• uh = solve(solver, op);
```

```
• writevtk(Ω, "Poisson", cellfields = ["uh" => uh]);
```



# Lid Driven Cavity Flow (2D)

Incompressible Lid Driven Cavity Flow, solved using first order elements and  $\theta$ -method ode method

In this example you are going to see:

- Cartesian Mesh
- Multifield problem
- Time dependent problem
- Non Linear solver

```
• using Gridap 
```

```
• function stretching_y_function(x)
•     gamma1 = 2.5
•     S = 0.5815356159649889 #for rescaling the function over the domain -0.5 ->
•     0.5
•     -tanh.(gamma1 .* (x)) ./ tanh.(gamma1) .* S
• end
```

```
• function stretching(x::Point)
•     m = zeros(length(x))
•     m[1] = stretching_y_function(x[1])
•
•
•     m[2] = stretching_y_function(x[2])
•     Point(m)
• end
```

Parameters defintion

```

• begin
•   L = 0.5 #Domain dimension
•   N = 50 #Number of cells for each dimension
•   order = 1 #element order
•   t0 = 0.0
•   tF = 5.0
•   dt = 0.01
•
•   D = 2 #dimension (2d or 3d)
•   u_in = 1.0 #Lid Velocity
•
•   Re = 1000
•    $\theta$  = 1.0 # ODE parameter
•
• end

```

```

• v = u_in*2*L/Re

```

## Model definition

---

```

• begin
•   domain = (-L, L, -L, L)
•   partition = (N, N)
•   model = CartesianDiscreteModel(domain, partition, map=stretching)
• end

```

## Set boundary conditions

---

```

• begin
•   u_wall(x,t) = VectorValue(0.0, 0.0)
•   u_wall(t::Real) = x -> u_wall(x,t)
•
•
•
•   u_top(x,t) = VectorValue(u_in, 0.0)
•   u_top(t::Real) = x -> u_top(x,t)
• end

```

```

• begin
•   labels = get_face_labeling(model)
•   add_tag_from_tags!(labels, "diri1", [5,])
•   add_tag_from_tags!(labels, "diri0", [1, 2, 4, 3, 6, 7, 8])
•   add_tag_from_tags!(labels, "p", [4,])
•   u_diri_tags=["diri0", "diri1"]
•   u_diri_values = [u_wall, u_top]
•   p_diri_tags= "p"
•   p_diri_values = 0.0
•
• end

```

Creation FE spaces, MULTIFIELD

```

• begin
•   reffe_u = ReferenceFE(lagrangian, VectorValue{D,Float64}, order)
•   reffe_p = ReferenceFE(lagrangian, Float64, order)
•   V = TestFESpace(model, reffe_u, conformity=:H1, dirichlet_tags=u_diri_tags)
•   U = TransientTrialFESpace(V, u_diri_values)
•   Q = TestFESpace(model, reffe_p, conformity=:H1, dirichlet_tags=p_diri_tags)
•   P = TrialFESpace(Q, p_diri_values)
•   Y = MultiFieldFESpace([V, Q])
•   X = TransientMultiFieldFESpace([U, P])
• end

```

## Numerical discretization

---

```

• degree = 4*order;

```

```

• Ω = Triangulation(model);

```

```

• dΩ = Measure(Ω, degree);

```

Volume force definition

```

• begin
•   hf(x,t) = VectorValue(0.0, 0.0);
•   hf(t::Real) = x -> hf(x,t);
• end

```

## Variational Formulation

---

### Conservation equations

```

• Rm(t, (u, p)) = ∂t(u) + u · ∇(u) + ∇(p) - hf(t); #- v*Δ(u)

```

```

• dRm((u, p), (du, dp), (v, q)) = du · ∇(u) + u · ∇(du) + ∇(dp); #- v*Δ(du)

```

$$\cdot \operatorname{Rc}(u) = \nabla \cdot u;$$

$$\cdot d\operatorname{Rc}(du) = \nabla \cdot du;$$

$$\int_{\Omega} \frac{\partial u}{\partial t} \cdot v \, d\Omega + \int_{\Omega} \nabla v \cdot \nabla u \, d\Omega - \int_{\Omega} (\nabla \cdot v) p \, d\Omega + \int_{\Omega} q (\nabla \cdot u) \, d\Omega + \int_{\Omega} (u \cdot \nabla(u)) \cdot v \, d\Omega -$$

$$\cdot \operatorname{var\_eq}(t, (u, p), (v, q)) = \int (\partial t(u) \cdot \underline{v}) d\Omega + \int ((u \cdot \nabla(u)) \cdot \underline{v}) d\Omega - \int ((\nabla \cdot \underline{v}) * \underline{p}) d\Omega + \int ((q * (\nabla \cdot u))) d\Omega + \underline{v} * \int (\nabla(\underline{v}) \odot \nabla(u)) d\Omega - \int (\underline{hf}(t) \cdot \underline{v}) d\Omega$$

## Stabilization Parameters

Get cell dimension

$$\cdot h = \operatorname{lazy\_map}(h \rightarrow h^{(1 / \underline{D})}, \operatorname{get\_cell\_measure}(\underline{\Omega}))$$

$$\tau = \left( \frac{2u}{h} + \frac{4\nu}{h^2} + \frac{2}{dt} \right)^{-1}$$

$$\begin{aligned} &\cdot \operatorname{function} \tau(u, h) \\ &\cdot \quad r = 1 \\ &\cdot \quad \tau_2 = h^2 / (4 * \underline{v}) \\ &\cdot \quad \operatorname{val}(x) = x \\ &\cdot \quad \operatorname{val}(x::\operatorname{Gridap.Fields.ForwardDiff.Dual}) = x.\operatorname{value} \\ &\cdot \quad u = \operatorname{val}(\operatorname{norm}(u)) \\ &\cdot \\ &\cdot \quad \operatorname{if} \operatorname{iszero}(u) \\ &\cdot \quad \quad \operatorname{return} \tau_2 \\ &\cdot \quad \operatorname{end} \\ &\cdot \\ &\cdot \quad \tau_3 = \underline{dt} / 2 \\ &\cdot \quad \tau_1 = h / (2 * u) \\ &\cdot \quad \operatorname{return} 1 / (1 / \tau_1^r + 1 / \tau_2^r + 1 / \tau_3^r) \\ &\cdot \operatorname{end}; \end{aligned}$$

$$\tau_b = (u \cdot u) \tau$$

$$\cdot \tau_b(u, h) = (u \cdot u) * \underline{\tau}(u, h);$$

## Stabilization Equation

$$\int_{\Omega} \tau \cdot (u \cdot \nabla(v) + \nabla(q)) R_m \, d\Omega + \int_{\Omega} \tau_b \cdot (\nabla \cdot v) R_c \, d\Omega$$



- $\text{stab\_eq}(\mathbf{t}, (\mathbf{u}, \mathbf{p}), (\mathbf{v}, \mathbf{q})) = \int ((\boldsymbol{\tau} \circ (\mathbf{u}, \mathbf{h}) * (\mathbf{u} \cdot \nabla(\mathbf{v}) + \nabla(\mathbf{q}))) \odot \mathbf{Rm}(\mathbf{t}, (\mathbf{u}, \mathbf{p}))) + \boldsymbol{\tau} \mathbf{b} \circ (\mathbf{u}, \mathbf{h}) * (\nabla \cdot \mathbf{v}) \odot \mathbf{Rc}(\mathbf{u}) \, d\Omega;$

- $\text{res\_eq}(\mathbf{t}, (\mathbf{u}, \mathbf{p}), (\mathbf{v}, \mathbf{q})) = \text{var\_eq}(\mathbf{t}, (\mathbf{u}, \mathbf{p}), (\mathbf{v}, \mathbf{q})) + \text{stab\_eq}(\mathbf{t}, (\mathbf{u}, \mathbf{p}), (\mathbf{v}, \mathbf{q}));$

Jacobians

- `begin`
- $\text{dvar\_eq}(\mathbf{t}, (\mathbf{u}, \mathbf{p}), (\mathbf{du}, \mathbf{dp}), (\mathbf{v}, \mathbf{q})) = \int (((\mathbf{du} \cdot \nabla(\mathbf{u})) \cdot \mathbf{v}) + ((\mathbf{u} \cdot \nabla(\mathbf{du})) \cdot \mathbf{v}) + (\nabla(\mathbf{dp}) \cdot \mathbf{v}) + (\mathbf{q} * (\nabla \cdot \mathbf{du}))) \, d\Omega + \mathbf{v} * \int (\nabla(\mathbf{v}) \odot \nabla(\mathbf{du})) \, d\Omega$
- $\text{dstab\_eq}(\mathbf{t}, (\mathbf{u}, \mathbf{p}), (\mathbf{du}, \mathbf{dp}), (\mathbf{v}, \mathbf{q})) = \int (((\boldsymbol{\tau} \circ (\mathbf{u}, \mathbf{h}) * (\mathbf{u} \cdot \nabla(\mathbf{v})' + \nabla(\mathbf{q}))) \odot \mathbf{dRm}((\mathbf{u}, \mathbf{p}), (\mathbf{du}, \mathbf{dp}), (\mathbf{v}, \mathbf{q}))) + ((\boldsymbol{\tau} \circ (\mathbf{u}, \mathbf{h}) * (\mathbf{du} \cdot \nabla(\mathbf{v})')) \odot \mathbf{Rm}(\mathbf{t}, (\mathbf{u}, \mathbf{p}))) + (\boldsymbol{\tau} \mathbf{b} \circ (\mathbf{u}, \mathbf{h}) * (\nabla \cdot \mathbf{v}) \odot \mathbf{dRc}(\mathbf{du}))) \, d\Omega$
- $\text{jac}(\mathbf{t}, (\mathbf{u}, \mathbf{p}), (\mathbf{du}, \mathbf{dp}), (\mathbf{v}, \mathbf{q})) = \text{dvar\_eq}(\mathbf{t}, (\mathbf{u}, \mathbf{p}), (\mathbf{du}, \mathbf{dp}), (\mathbf{v}, \mathbf{q})) + \text{dstab\_eq}(\mathbf{t}, (\mathbf{u}, \mathbf{p}), (\mathbf{du}, \mathbf{dp}), (\mathbf{v}, \mathbf{q}))$
- `end;`

- $\text{jac\_t}(\mathbf{t}, (\mathbf{u}, \mathbf{p}), (\mathbf{dut}, \mathbf{dpt}), (\mathbf{v}, \mathbf{q})) = \int (\mathbf{dut} \cdot \mathbf{v}) \, d\Omega + \int (\boldsymbol{\tau} \circ (\mathbf{u}, \mathbf{h}) * (\mathbf{u} \cdot \nabla(\mathbf{v}) + (1/\theta) * \nabla(\mathbf{q})) \odot \mathbf{dut}) \, d\Omega;$

- `op = TransientFEOperator(res_eq, jac, jac_t, X, Y);`

## ODE settings

- `uh0 = interpolate_everywhere(VectorValue(0.0,0.0), U(0));`

- `ph0 = interpolate_everywhere(0.0, P(0));`

- `xh0 = interpolate_everywhere([uh0, ph0], X(0));`

## Solver settings

- `nls_solver = NLSolver(show_trace=true, method=:newton);`

- `ode_solver = ThetaMethod(nls_solver, dt, theta);`

- `sol_t = solve(ode_solver, op, xh0, t0, tF)`

## View Results

- `using Plots` 

- `x = range(domain[1], domain[2]; length=32)`

```
• y = range(domain[3],domain[4]; length=32)
```

```
• function gettimeseries(sol)
•   pressures = []
•   vmagnitudes = []
•   coords = Point(x',y)
•   for ((uh, ph),tn) in sol
•     push!(pressures, ph.(coords))
•     push!(vmagnitudes, norm.(uh.(coords)))
•   end
•   return pressures, vmagnitudes
• end
```

```
• # Actual solution happens here, grab a Ⓢ
• p,v = gettimeseries(sol_t)
```

```
• @bind timestep Slider(1:length(p))
```

```
• heatmap(x,y,p[timestep]; aspect_ratio=:equal)
```

```
• heatmap(x,y,v[timestep]; aspect_ratio=:equal)
```

## Using PETSc as solver

=====


```
• Petsc_options = "-snes_type newtonls -snes_linesearch_type basic -
snes_linesearch_damping 1.0 -snes_rtol 1.0e-14 -snes_atol 0.0 -snes_monitor -
pc_type asm -sub_pc_type lu -ksp_type gmres -ksp_gmres_restart 30 -
snes_converged_reason -ksp_converged_reason -ksp_error_if_not_converged true ";
```

```
• # using GridapPETSc
```

```
• # GridapPETSc.with(args=split(petsc_options)) do
• #   Petsc_nls_solver = NLSolver(show_trace=true, method=:newton);
• #   Petsc_ode_solver = ThetaMethod(Petsc_nls_solver,dt,θ);
• #   Petsc_sol_t = solve(Petsc_ode_solver,op,xh0,t0,tF);
• #   createpvd("lid_driven") do pvd
• #     for (xh_tn,tn) in Petsc_sol_t
• #       uh_tn, ph_tn = xh_tn
• #       pvd[tn] = createvtk(Q,"lid_driven_$tn".vtu,cellfields=["uh"=>uh_tn,
• "ph"=>ph_tn])
• #     end
• #   end
• # end
```

## Other features

=====

- 
- Alpha-method for ODE
  - Higher-order elements
  - Discontinuous Galerkin with interior penalty
  - Adjoint optimization
  - Uniform refinement