A Simple Finite Element Code written in Julia

Bill McLean, UNSW

ANZMC Melbourne December 2014

Purpose

Required for the computational component of an honours course (3 hours/week for 6 weeks) on finite elements.

Purpose

Required for the computational component of an honours course (3 hours/week for 6 weeks) on finite elements.

Suffices for code to handle

- second-order, linear elliptic PDEs in 2D,
- P_1-elements (triangular, degree 1).

Purpose

Required for the computational component of an honours course (3 hours/week for 6 weeks) on finite elements.

Suffices for code to handle

- second-order, linear elliptic PDEs in 2D,
- P_1-elements (triangular, degree 1).

Expect students to study the source code to understand FEM, particularly matrix assembly. Should not be a "black box".

Purpose

Required for the computational component of an honours course (3 hours/week for 6 weeks) on finite elements.

Suffices for code to handle

- second-order, linear elliptic PDEs in 2D,
- P_1-elements (triangular, degree 1).

Expect students to study the source code to understand FEM, particularly matrix assembly. Should not be a “black box”.

Students also use the code in an assignment.

Why Julia?

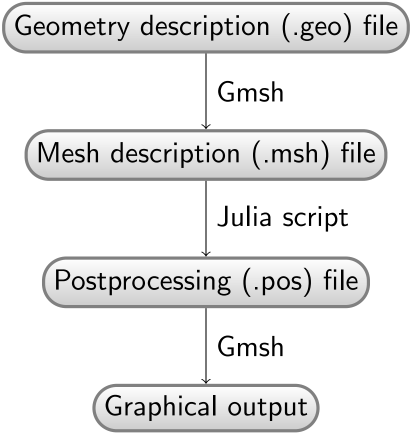
- Modern alternative to Matlab.
- MIT licence and good documentation.
- Cross-platform: Linux, Windows and OS X.
- Easy, student-friendly syntax.
- Fast execution thanks to LLVM.
- Extensive bindings to standard numerical libraries.

Why Julia?

- Modern alternative to Matlab.
- MIT licence and good documentation.
- Cross-platform: Linux, Windows and OS X.
- Easy, student-friendly syntax.
- Fast execution thanks to LLVM.
- Extensive bindings to standard numerical libraries.

Ported code from previous version written in Python.

Overview of FEM solution process



Gmsh

C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering* 79(11), pp. 1309–1331, 2009.

- GPL software with comprehensive documentation.
- Cross-platform: Linux, Windows and OS X.
- Fast and robust meshes in 2D and 3D.
- Convenient data format for FEM.
- Extensive visualisation features.
- Reasonably easy to handle simple geometries.

Gmsh

C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering* 79(11), pp. 1309–1331, 2009.

- GPL software with comprehensive documentation.
- Cross-platform: Linux, Windows and OS X.
- Fast and robust meshes in 2D and 3D.
- Convenient data format for FEM.
- Extensive visualisation features.
- Reasonably easy to handle simple geometries.

FEM code has no other software dependency.

Package modules

Gmsh.jl Handles reading and writing of Gmsh data files.

FEM.jl Handles assembly of linear system.

PlanarPoisson.jl Routines to compute element stiffness matrix, element load vector, etc.

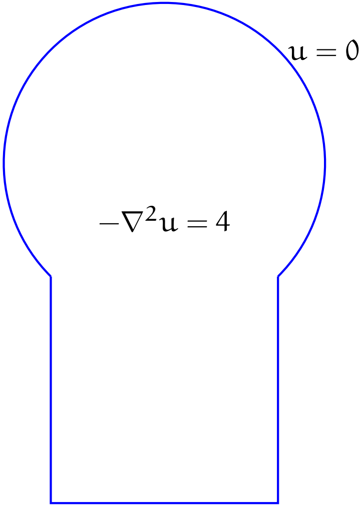
Package modules

Gmsh.jl	Handles reading and writing of Gmsh data files.
FEM.jl	Handles assembly of linear system.
PlanarPoisson.jl	Routines to compute element stiffness matrix, element load vector, etc.

How many lines of code?

```
$ wc *.jl
 229   657  6469 FEM.jl
 249   823  7850 Gmsh.jl
 152   518  4423 PlanarPoisson.jl
 630  1998 18742 total
```

Simple example



Weak formulation and finite element approximation

Sobolev space $H_0^1(\Omega)$ consists of those $u \in L_2(\Omega)$ such that $\partial_x u$ and $\partial_y u \in L_2(\Omega)$, with $u = 0$ on Ω.
Weak solution $u \in H_0^1(\Omega)$ satisfies

$$\int_\Omega \nabla u \cdot \nabla v = \int_\Omega 4v \quad \text{for all } v \in H_0^1(\Omega).$$

Weak formulation and finite element approximation

Sobolev space $H_0^1(\Omega)$ consists of those $u \in L_2(\Omega)$ such that $\partial_x u$ and $\partial_y u \in L_2(\Omega)$, with $u = 0$ on Ω.
Weak solution $u \in H_0^1(\Omega)$ satisfies

$$\int_\Omega \nabla u \cdot \nabla v = \int_\Omega 4v \quad \text{for all } v \in H_0^1(\Omega).$$

Approximate Ω by triangulated domain Ω_h.

Finite element space $\mathcal{S}_h$ consists of all continuous, piecewise-linear functions that vanish on $\partial\Omega_h$; thus, $\mathcal{S}_h \subseteq H_0^1(\Omega_h)$.

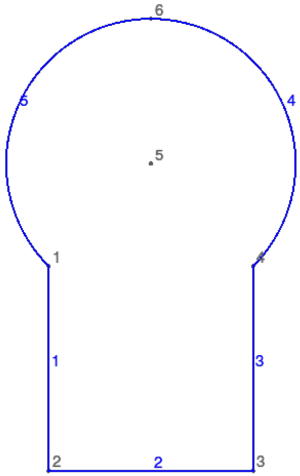
Finite element solution $u_h \in \mathcal{S}_h$ satisfies

$$\int_{\Omega_h} \nabla u_h \cdot \nabla v = \int_{\Omega_h} 4v \quad \text{for all } v \in \mathcal{S}_h.$$

Describe the geometry

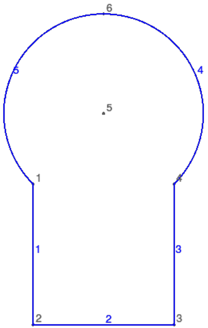
Create file `keyhole.geo` containing.

```
Point(1) = {-1, 0, 0};
Point(2) = {-1,-2, 0};
Point(3) = { 1,-2, 0};
Point(4) = { 1, 0, 0};
Point(5) = { 0, 1, 0};
Point(6) = { 0, 1+Sqrt(2), 0};

Line(1) = {1, 2};
Line(2) = {2, 3};
Line(3) = {3, 4};
Circle(4) = {4, 5, 6};
Circle(5) = {6, 5, 1};
```



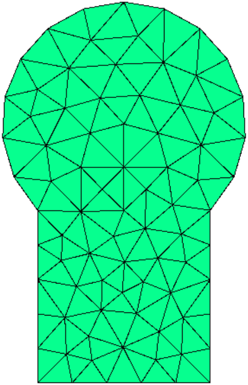
Label the domain and boundary

```
Line Loop(7) = {1, 2, 3, 4, 5};
Plane Surface(1) = { 7 };

Physical Surface("Omega") = { 1 };
Physical Line("Gamma") = { 1, 2, 3, 4, 5 };
```



Triangulate the domain

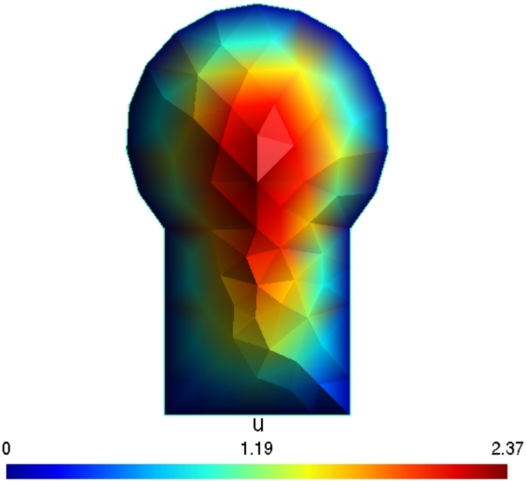
Use Gmsh GUI or CLI to create file `keyhole.msh`.



Solver script

```
using Gmsh
using FEM
using PlanarPoisson
mesh = read_msh_file("keyhole.msh")
essential_bc = [ "Gamma" ]
f(x) = 4.0
vp = VariationalProblem(mesh, essential_bc)
add_bilin_form!(vp,  "Omega", grad_dot_grad!)
add_lin_functnl!(vp, "Omega", source_times_func!, f)
A, b = assembled_linear_system(vp)
ufree = A \ b
u = complete_soln(ufree, vp)
open("keyhole.pos", "w") do fid
    write_format_version(fid)
    save_warp_nodal_scalar_field(u, "u", mesh, fid)
end
```

Visualisation



Geometric element type

In the Gmsh module.

```
immutable GeomType
    gmsh_code :: Integer
    dimen     :: Integer
    nonodes   :: Integer
end

const LINE        = GeomType(1, 1, 2)
const TRIANGLE    = GeomType(2, 2, 3)
const TETRAHEDRON = GeomType(4, 3, 4)

const GETGEOMTYPE = { 1 => LINE,  2 => TRIANGLE,
                      4 => TETRAHEDRON }
```

Mesh data structure

```
immutable Mesh
    coord    :: Array{Float64, 2}
    physdim  :: Dict{String, Integer}
    physnum  :: Dict{String, Integer}
    physname :: Dict{Integer, String}
    elmtype  :: Dict{String, GeomType}
    elms_of  :: Dict{String, Matrix{Integer}}
    nodes_of :: Dict{String, Set{Integer}}
end
```

For example,

$$\begin{aligned}\texttt{mesh.coord[:,n]} &= x,\ y,\ z \text{ coordinates of } n\text{th node},\\ \texttt{mesh.elms\_of["Omega"]} &= \text{connectivity matrix for elements in } \Omega.\end{aligned}$$

FEM data structures

```
immutable DoF
    isfree     :: Vector{Bool}
    freenode   :: Vector{Integer}
    fixednode  :: Vector{Integer}
    node2free  :: Vector{Integer}
    node2fixed :: Vector{Integer}
end

immutable VariationalProblem
    mesh         :: Mesh
    dof          :: DoF
    essential_bc :: Vector{ASCIIString}
    bilin_form   :: Vector{Any}
    lin_functnl  :: Vector{Any}
    ufixed       :: Vector{Float64}
end
```

Inhomogeneous Dirichlet data

```
function assign_bdry_vals!(vp::VariationalProblem,
                           name::String, g::Function)
    if !(name in vp.essential_bc)
        error("$name: not listed in essential_bc")
    end
    for nd in vp.mesh.nodes_of[name]
        i = vp.dof.node2fixed[nd]
        x = vp.mesh.coord[:,nd]
        vp.ufixed[i] = g(x)
    end
end
```

Matrix assembly

```
A = sparse(Int64[], Int64[], Float64[], nofree, nofree)
b = zeros(nofree)
for (name, elm_mat!, coef) in vp.bilin_form
    next = assembled_matrix(name, elm_mat!,
                            coef, mesh, dof)
    A += next[:,1:nofree]
    if nofixed > 0
        b -= next[:,nofree+1:end] * vp.ufixed
    end
end
for (name, elm_vec!, f) in vp.lin_functnl
    next = assembled_vector(name, elm_vec!, f,
                            mesh, dof)
    b += next
end
```

A more complicated example

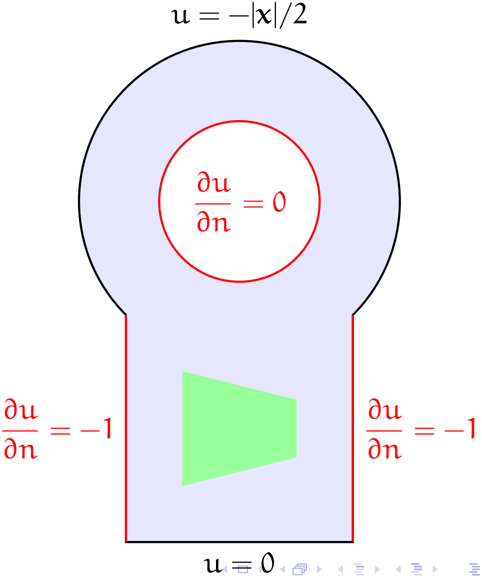
$$-\nabla \cdot (a\nabla u) = f$$

$a = 1$

$f = 1$

$a = 10$

$f = 4$



Weak formulation and finite element approximation

$$\mathcal{S} = \{ v \in H^1(\Omega) : v = -|x|/2 \text{ for } x \in \text{Black} \},$$
$$\mathcal{T} = \{ u \in H^1(\Omega) : v = 0 \text{ for } x \in \text{Black} \}.$$

Weak solution $u \in \mathcal{H}$ satisfies

$$\int_{\text{Blue}} \nabla u \cdot \nabla v + 10 \int_{\text{Green}} \nabla u \cdot \nabla v = \int_{\text{Blue}} v + \int_{\text{Green}} 4v - \int_{\text{Red}} v$$

for all $v \in \mathcal{T}$.

Weak formulation and finite element approximation

$$\mathcal{S} = \{v \in H^1(\Omega) : v = -|x|/2 \text{ for } x \in \text{Black}\},$$
$$\mathcal{T} = \{u \in H^1(\Omega) : v = 0 \text{ for } x \in \text{Black}\}.$$

Weak solution $u \in \mathcal{H}$ satisfies

$$\int_{\text{Blue}} \nabla u \cdot \nabla v + 10 \int_{\text{Green}} \nabla u \cdot \nabla v = \int_{\text{Blue}} v + \int_{\text{Green}} 4v - \int_{\text{Red}} v$$

for all $v \in \mathcal{T}$.

Finite element solution $u_h \in \mathcal{S}_h$ satisfies

$$\int_{\text{Blue}_h} \nabla u_h \cdot \nabla v + 10 \int_{\text{Green}_h} \nabla u_h \cdot \nabla v = \int_{\text{Blue}_h} v + \int_{\text{Green}_h} 4v - \int_{\text{Red}_h} v$$

for all $v \in \mathcal{T}_h$.

Setting up the variational problem

```
g(x) = -hypot(x[1],x[2])/2
assign_bdry_vals!(vp, "North", g)
add_bilin_form!(vp,  "Major",
                grad_dot_grad!, 1.0)
add_lin_functnl!(vp, "Major",
                 source_times_func!, x->1.0)
add_bilin_form!(vp,  "Minor",
                grad_dot_grad!, 10.0)
add_lin_functnl!(vp, "Minor",
                 source_times_func!, x->4.0)
add_lin_functnl!(vp, "East",
                 bdry_source_times_func!, x->-1.0)
add_lin_functnl!(vp, "West",
                 bdry_source_times_func!, x->-1.0)
```

Visualisation

4604 nodes, 9291 triangles.