nb-l15-fv

December 19, 2019

Scientific Computing, TU Berlin, WS 2019/2020, Lecture 15

Jürgen Fuhrmann, WIAS Berlin

0.1 Implementation of the finite volume method

1 Packages to be used

[1]:
```
using Triangulate
using PyPlot
using ExtendableSparse
using SparseArrays
using Printf
```

1.1 Plot function on grid

[2]:
```
function plot(Plotter::Module,u,pointlist, trianglelist)
    cmap="coolwarm"
    levels=10
    t=transpose(trianglelist.-1)
    x=view(pointlist,1,:)
    y=view(pointlist,2,:)
    ax=Plotter.matplotlib.pyplot.gca()
    ax.set_aspect(1)
    #  tricontour/tricontourf takes triangulation as argument
    Plotter.tricontourf(x,y,t,u,levels=levels,cmap=cmap)
    PyPlot.colorbar(shrink=0.5)
    Plotter.tricontour(x,y,t,u,levels=levels,colors="k")
end
```

[2]:
```
plot (generic function with 1 method)
```

1.2 Plot grid and function on grid

[3]:
```
function plotpair(Plotter::Module,u,triout)
    if ispyplot(Plotter)
        PyPlot=Plotter
```

```
        PyPlot.clf()
        PyPlot.subplot(121)
        PyPlot.title("Grid")
        Triangulate.plot(PyPlot,triout)
        PyPlot.subplot(122)
        PyPlot.title("Solution")
        plot(PyPlot,u,triout.pointlist, triout.trianglelist)
    end

end
```

[3]: plotpair (generic function with 1 method)

1.3 Triangle form factors

[4]:
```
function trifactors!(npar::Vector, epar::Vector, itri::Int, pointlist,␣
↪trianglelist)
    i1=trianglelist[1,itri]
    i2=trianglelist[2,itri]
    i3=trianglelist[3,itri]


    # Fill matrix of edge vectors
    V11= pointlist[1,i2]- pointlist[1,i1]
    V21= pointlist[2,i2]- pointlist[2,i1]

    V12= pointlist[1,i3]- pointlist[1,i1]
    V22= pointlist[2,i3]- pointlist[2,i1]

    V13= pointlist[1,i3]- pointlist[1,i2]
    V23= pointlist[2,i3]- pointlist[2,i2]

    # Compute determinant
    det=V11*V22 - V12*V21
    vol=0.5*det
    ivol = 1.0/vol

    # squares of edge lengths
    dd1=V13*V13+V23*V23 # l32
    dd2=V12*V12+V22*V22 # l31
    dd3=V11*V11+V21*V21 # l21


    # contributions to \sigma_kl/h_kl
    epar[1]= (dd2+dd3-dd1)*0.125*ivol
    epar[2]= (dd3+dd1-dd2)*0.125*ivol
    epar[3]= (dd1+dd2-dd3)*0.125*ivol
```

```
    # contributions to \omega_k
    npar[1]= (epar[3]*dd3+epar[2]*dd2)*0.25
    npar[2]= (epar[1]*dd1+epar[3]*dd3)*0.25
    npar[3]= (epar[2]*dd2+epar[1]*dd1)*0.25
end
```

[4]: trifactors! (generic function with 1 method)

1.4 Boundary form factors

```
function bfacefactors!(npar::Vector,ibface::Int, pointlist, segmentlist)
    i1=segmentlist[1,ibface]
    i2=segmentlist[2,ibface]
    dx=pointlist[1,i1]-pointlist[1,i2]
    dy=pointlist[2,i1]-pointlist[2,i2]
    d=0.5*sqrt(dx*dx+dy*dy)
    npar[1]=d
    npar[2]=d
end
```

[5]: bfacefactors! (generic function with 1 method)

1.5 Assembly of matrix (for Dirichlet)

```
function assemble!(matrix, # System matrix
                   rhs,    # Right hand side vector
                   frhs::Function, # Source/sink function
                   gbc::Function,  # Boundary condition function
                   pointlist,
                   trianglelist,
                   segmentlist)
    num_nodes_per_cell=3;
    num_edges_per_cell=3;

    # Array which describing nodes per edge locally
    local_edgenodes=zeros(Int32,2,3)
    local_edgenodes[1,1]=2
    local_edgenodes[2,1]=3

    local_edgenodes[1,2]=3
    local_edgenodes[2,2]=1

    local_edgenodes[1,3]=1
    local_edgenodes[2,3]=2
```

```
    # Data for form factors
    epar=zeros(num_nodes_per_cell)
    npar=zeros(num_edges_per_cell)

    # Set right hand side to zero
    rhs.=0.0

    # Loop over all triangles
    ntri=size(trianglelist,2)
    for itri=1:ntri
        trifactors!(npar,epar,itri,pointlist,trianglelist)
        # Assemble nodal contributions to right hand side
        for k_local=1:num_nodes_per_cell
            k=trianglelist[k_local,itri]
            x=pointlist[1,k]
            y=pointlist[2,k]
            rhs[k]+=frhs(x,y)*npar[k_local]
        end
        # Assemble edge contributions to matrix
        for iedge=1:num_edges_per_cell
            k=trianglelist[local_edgenodes[1,iedge],itri]
            l=trianglelist[local_edgenodes[2,iedge],itri]
            matrix[k,k]+=epar[iedge]
            matrix[l,k]-=epar[iedge]
            matrix[k,l]-=epar[iedge]
            matrix[l,l]+=epar[iedge]
        end
    end

    # Assemble penalty terms for Dirichlet boundary conditions
    penalty=1.0e30
    nbface=size(segmentlist,2)
    for ibface=1:nbface
        for i=1:2
            k=segmentlist[i,ibface]
            matrix[k,k]+=penalty
            x=pointlist[1,k]
            y=pointlist[2,k]
            rhs[k]+=penalty*gbc(x,y)
        end
    end
end
```

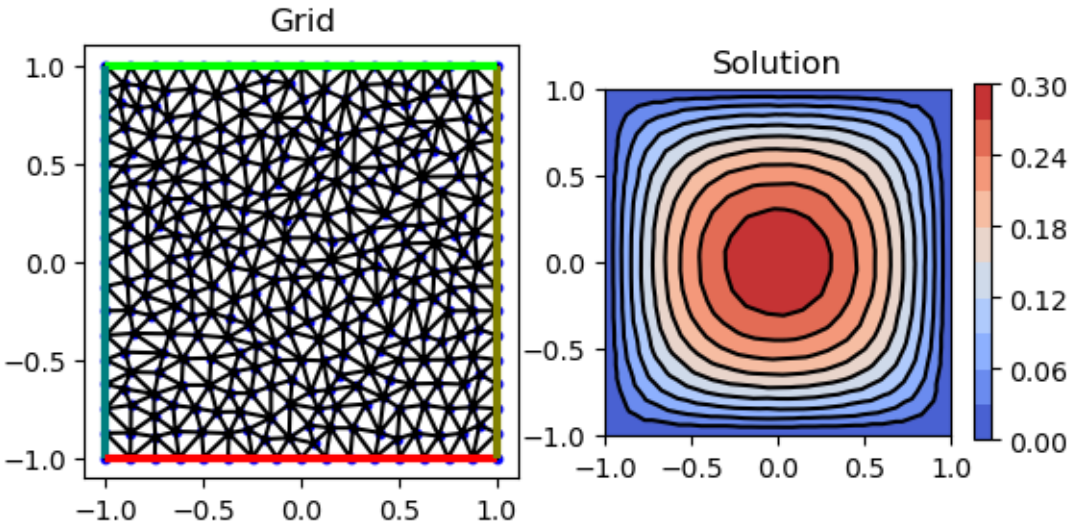
[6]: assemble! (generic function with 1 method)

1.6 Run things

[7]:
```
function example1(;plotgrid=false)
    triin=TriangulateIO()
    triin.pointlist=Matrix{Cdouble}([-1.0 -1.0; 1.0 -1.0 ; 1.0 1.0 ; -1.0 1.0␣
↪]')
    triin.segmentlist=Matrix{Cint}([1 2 ; 2 3 ; 3 4 ; 4 1 ]')
    triin.segmentmarkerlist=Vector{Int32}([1, 2, 3, 4])
    (triout, vorout)=triangulate("pqa0.01qQD", triin)

    if plotgrid
        plotpair(PyPlot,triin,triout,voronoi=vorout)
        return
    end

    n=size(triout.pointlist,2)
    frhs(x,y)=1
    gbc(x,y)=0
    matrix=spzeros(n,n)
    rhs=zeros(n)
    assemble!(matrix,rhs, frhs,gbc,triout.pointlist,triout.trianglelist, triout.
↪segmentlist)
    sol=matrix\rhs
    plotpair(PyPlot,sol,triout)
end
```

[7]:
```
example1 (generic function with 1 method)
```

[8]:
```
example1()
```



[8]: PyObject <matplotlib.tri.tricontour.TriContourSet object at 0x7f2b22e43be0>

1.7 Calculate norms of solution

[9]:
```
function norms(u,pointlist,trianglelist)
    local_edgenodes=zeros(Int32,2,3)
    local_edgenodes[1,1]=2
    local_edgenodes[2,1]=3

    local_edgenodes[1,2]=3
    local_edgenodes[2,2]=1

    local_edgenodes[1,3]=1
    local_edgenodes[2,3]=2
    num_nodes_per_cell=3;
    num_edges_per_cell=3;
    epar=zeros(num_nodes_per_cell)
    npar=zeros(num_edges_per_cell)
    l2norm=0.0
    h1norm=0.0
    ntri=size(trianglelist,2)
    for itri=1:ntri
        trifactors!(npar,epar,itri,pointlist,trianglelist)
        for k_local=1:num_nodes_per_cell
            k=trianglelist[k_local,itri]
            x=pointlist[1,k]
            y=pointlist[2,k]
            l2norm+=u[k]^2*npar[k_local]
        end
        for iedge=1:num_edges_per_cell
            k=trianglelist[local_edgenodes[1,iedge],itri]
            l=trianglelist[local_edgenodes[2,iedge],itri]
            h1norm+=(u[k]-u[l])^2*epar[iedge]
        end
    end
    return (sqrt(l2norm),sqrt(h1norm));
end
```

[9]: norms (generic function with 1 method)

1.8 Convergence test with exact solution

[10]:
```
function example2(;plotgrid=false,nref0=0, nref1=0,k=1,l=1)
    allh=[]
    alll2=[]
    allh1=[]
    if nref0>nref1
```

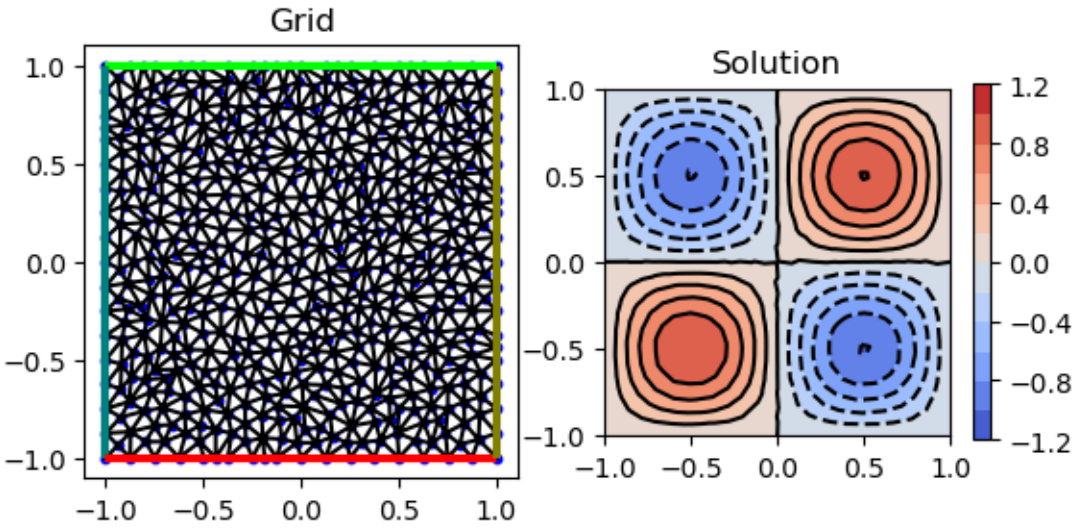
```
        nref1=nref0
    end
    for iref=nref0:nref1
        triin=TriangulateIO()
        triin.pointlist=Matrix{Cdouble}([-1.0 -1.0; 1.0 -1.0 ; 1.0 1.0 ; -1.0 1.
↪0 ]')
        triin.segmentlist=Matrix{Cint}([1 2 ; 2 3 ; 3 4 ; 4 1 ]')
        triin.segmentmarkerlist=Vector{Int32}([1, 2, 3, 4])
        area=0.1*2.0^(-2*iref)
        h=sqrt(area)
        s_area=@sprintf("%.20f",area)
        (triout, vorout)=triangulate("pqa$(s_area)qQD", triin)

        if plotgrid
            plotpair(PyPlot,triin,triout,voronoi=vorout)
            return
        end

        n=size(triout.pointlist,2)
        fexact(x,y)=sin(k*pi*x)*sin(l*pi*y);
        uexact=zeros(n)
        for i=1:n
            uexact[i]=fexact(triout.pointlist[1,i],triout.pointlist[2,i])
        end
        frhs(x,y)=(k^2+l^2)*pi^2*fexact(x,y);
        gbc(x,y)=0
        # Use ExtendableSparse for faster matrix construction
        matrix=ExtendableSparseMatrix{Float64,Int64}(n,n)
        rhs=zeros(n)
        assemble!(matrix,rhs, frhs,gbc,triout.pointlist,triout.trianglelist,␣
↪triout.segmentlist)
        flush!(matrix)
        sol=matrix.cscmatrix\rhs
        plotpair(PyPlot,sol,triout)
        (l2norm,h1norm)=norms(uexact-sol,triout.pointlist,triout.trianglelist)
        @show l2norm,h1norm
        push!(allh,h)
        push!(allh1,h1norm)
        push!(alll2,l2norm)
    end
    if nref1>nref0
        PyPlot.clf()
        PyPlot.grid()
        PyPlot.xlabel("h")
        PyPlot.ylabel("error")
        PyPlot.loglog(allh, alll2, label="l2",color=:green,marker="o")
        PyPlot.loglog(allh, allh.^2, label="O(h^2)",color=:green)
```

```
        PyPlot.loglog(allh, allh1, label="h1",color=:red,marker="o")
        PyPlot.loglog(allh, allh, label="O(h)",color=:red)
        PyPlot.legend()
    end
end
```

[10]: example2 (generic function with 1 method)

Run example

[11]:
```
example2(nref0=2)
```




```
(l2norm, h1norm) = (0.012935904285071756, 0.07960745529988267)
```

This notebook was generated using Literate.jl.