nb-l22-precontest2d

January 27, 2020

Scientific Computing, TU Berlin, WS 2019/2020, Lecture 22

Jürgen Fuhrmann, WIAS Berlin

0.1 Performance test of Preconditioned CG for 2D problems

0.2 Packages

[1]:
```
using PyPlot
using ExtendableSparse
using SparseArrays
using Printf
using CPUTime
using IncompleteLU
using IterativeSolvers
using LinearAlgebra
using AlgebraicMultigrid
using Triangulate
```

[2]:
```
function compute_edge_matrix(itri, pointlist, trianglelist)
    i1=trianglelist[1,itri];
    i2=trianglelist[2,itri];
    i3=trianglelist[3,itri];

    V11= pointlist[1,i2]- pointlist[1,i1];
    V12= pointlist[1,i3]- pointlist[1,i1];

    V21= pointlist[2,i2]- pointlist[2,i1];
    V22= pointlist[2,i3]- pointlist[2,i1];

    det=V11*V22 - V12*V21;
    vol=0.5*det
    return (V11,V12,V21,V22,vol)
end
```

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

[3]:
```
function  compute_local_stiffness_matrix!(local_matrix,itri,␣
↪pointlist,trianglelist)

    (V11,V12,V21,V22,vol)=compute_edge_matrix(itri, pointlist, trianglelist)

    fac=0.25/vol

    local_matrix[1,1] = fac * (  ( V21-V22 )*( V21-V22 )+( V12-V11 )*( V12-V11␣
↪) );
    local_matrix[2,1] = fac * (  ( V21-V22 )* V22          - ( V12-V11 )*V12 );
    local_matrix[3,1] = fac * ( -( V21-V22 )* V21          + ( V12-V11 )*V11 );

    local_matrix[2,2] =  fac * (  V22*V22 + V12*V12 );
    local_matrix[3,2] =  fac * ( -V22*V21 - V12*V11 );

    local_matrix[3,3] =  fac * ( V21*V21+ V11*V11 );

    local_matrix[1,2] = local_matrix[2,1];
    local_matrix[1,3] = local_matrix[3,1];
    local_matrix[2,3] = local_matrix[3,2];
    return vol
end
```

[3]:
```
compute_local_stiffness_matrix! (generic function with 1 method)
```

0.3 Assembly of matrix (for Dirichlet)

[4]:
```
function  assemble!(matrix, # Global stiffness matrix
                    rhs,    # Right hand side vector
                    frhs::Function, # Source/sink function
                    gbc::Function,  # Boundary condition function
                    pointlist,
                    trianglelist,
                    segmentlist)
    num_nodes_per_cell=3;
    ntri=size(trianglelist,2)
    vol=0.0
    local_stiffness_matrix= [ 0.0 0.0 0.0; 0.0 0.0 0.0; 0.0  0.0  0.0 ]
    local_massmatrix= [ 2.0 1.0 1.0; 1.0 2.0 1.0; 1.0  1.0  2.0 ]
    local_massmatrix./=12.0
    rhs.=0.0

    # Main part
    for itri in 1:ntri
        vol=compute_local_stiffness_matrix!(local_stiffness_matrix,itri,␣
↪pointlist,trianglelist);
```

```
        for i  in 1:num_nodes_per_cell
            for j in 1:num_nodes_per_cell
                k=trianglelist[j,itri]
                x=pointlist[1,k]
                y=pointlist[2,k]
                rhs[trianglelist[i,itri]]+=vol*local_massmatrix[i,j]*frhs(x,y)
                ␣
↪matrix[trianglelist[i,itri],trianglelist[j,itri]]+=local_stiffness_matrix[i,j]
            end
        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
```

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

[5]:
```
function hmax(pointlist,trianglelist)
    num_edges_per_cell=3
    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

    h=0.0
    ntri=size(trianglelist,2)
    for itri=1:ntri
        for iedge=1:num_edges_per_cell
            k=trianglelist[local_edgenodes[1,iedge],itri]
            l=trianglelist[local_edgenodes[2,iedge],itri]
            dx=pointlist[1,k]-pointlist[1,l]
            dy=pointlist[2,k]-pointlist[2,l]
```

```
            h=max(h,dx^2+dy^2)
        end
    end
    return sqrt(h)
end
```

[5]: hmax (generic function with 1 method)

[6]:
```
function precontest(;nref0=0, nref1=0,k=1,l=1, plothist=false, plotscale=false)
    all_n=[]
    all_h=[]

    all_t_asm=[]
    all_t_direct=[]
    all_t_tria=[]

    all_t_cg_rsamg=[]
    all_t_cg_aggamg=[]
    all_t_cg_jacobi=[]
    all_t_cg_ilu0=[]
    all_t_cg_ilu1=[]
    all_t_cg_ilu2=[]

    all_n_cg_rsamg=[]
    all_n_cg_aggamg=[]
    all_n_cg_jacobi=[]
    all_n_cg_ilu0=[]
    all_n_cg_ilu1=[]
    all_n_cg_ilu2=[]

    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)
        s_area=@sprintf("%.20f",area)

        println("grid:")
        CPUtic()
        CPUtic()
        (triout, vorout)=triangulate("pqa$(s_area)qQD", triin)
        t_tria=CPUtoc()
```

```
        println()

        h=hmax(triout.pointlist,triout.trianglelist)
        push!(all_h,h)

        n=size(triout.pointlist,2)
        push!(all_n,n)
        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(n,n)
        rhs=zeros(n)

        # Set matrix
        println("assembly:")
        CPUtic()
        assemble!(matrix,rhs, frhs,gbc,triout.pointlist,triout.trianglelist,␣
↪triout.segmentlist)
        flush!(matrix)
        t_asm=CPUtoc()
        println()

        # Direct solver: UMFPACK
        println("direct:")
        CPUtic()
        sol=matrix\rhs
        t_direct=CPUtoc()
        println()

        # Jacobi solver from ExtendableSparse.jl
        CPUtic()
        println("cg_jacobi:")
        precon_jacobi=JacobiPreconditioner(matrix)
        sol,hist_cg_jacobi=cg(matrix,rhs,Pl=precon_jacobi, tol=1.0e-12,log=true)
        t_cg_jacobi=CPUtoc()
        println("iterations: $(hist_cg_jacobi.iters)\n")

        # ILU(0) solver from ExtendableSparse.jl
        println("cg_ilu0:")
        CPUtic()
        precon_ilu0=ILU0Preconditioner(matrix)
        sol,hist_cg_ilu0=cg(matrix,rhs,Pl=precon_ilu0, tol=1.0e-12,log=true)
```

```
t_cg_ilu0=CPUtoc()
println("iterations: $(hist_cg_ilu0.iters)\n")


# ILUT solver from IncompleteLU.jl
println("cg_ilut,  =0.2")
CPUtic()
precon_ilu1=ilu(matrix.cscmatrix, =0.2)
sol,hist_cg_ilu1=cg(matrix,rhs,Pl=precon_ilu1, tol=1.0e-12,log=true)
t_cg_ilu1=CPUtoc()
println("iterations: $(hist_cg_ilu1.iters)\n")

# ILUT solver from IncompleteLU.jl
println("cg_ilut,  =0.02")
CPUtic()
precon_ilu2=ilu(matrix.cscmatrix, =0.02)
sol,hist_cg_ilu2=cg(matrix,rhs,Pl=precon_ilu2, tol=1.0e-12,log=true)
t_cg_ilu2=CPUtoc()
println("iterations: $(hist_cg_ilu2.iters)\n")



# Smoothed Aggregation from AlgebraicMultigrid.jl
println("cg_aggamg:")
CPUtic()
precon_aggamg=aspreconditioner(smoothed_aggregation(matrix.cscmatrix))
sol,hist_cg_aggamg=cg(matrix,rhs,Pl=precon_aggamg, tol=1.0e-12,log=true)
t_cg_aggamg=CPUtoc()
println("iterations: $(hist_cg_aggamg.iters)\n")

# Ruge-Stüben form AlgebraicMultigrid.jl
println("cg_rsamg:")
CPUtic()
precon_rsamg=aspreconditioner(ruge_stuben(matrix.cscmatrix))
sol, hist_cg_rsamg=cg(matrix,rhs,Pl=precon_rsamg, tol=1.0e-12,log=true)
t_cg_rsamg=CPUtoc()
println("iterations: $(hist_cg_rsamg.iters)\n")



push!(all_t_tria,t_tria)
push!(all_t_asm,t_asm)
push!(all_t_direct,t_direct)


push!(all_t_cg_jacobi,t_cg_jacobi)
push!(all_t_cg_ilu0,t_cg_ilu0)
```

```
        push!(all_t_cg_ilu1,t_cg_ilu1)
        push!(all_t_cg_ilu2,t_cg_ilu2)
        push!(all_t_cg_rsamg,t_cg_rsamg)
        push!(all_t_cg_aggamg,t_cg_aggamg)

        push!(all_n_cg_jacobi,hist_cg_jacobi.iters)
        push!(all_n_cg_ilu0,  hist_cg_ilu0.iters)
        push!(all_n_cg_ilu1,  hist_cg_ilu1.iters)
        push!(all_n_cg_ilu2,  hist_cg_ilu2.iters)
        push!(all_n_cg_rsamg, hist_cg_rsamg.iters)
        push!(all_n_cg_aggamg,hist_cg_aggamg.iters)

        v(hist)=hist.data[:resnorm]
        if plothist
            PyPlot.clf()
            PyPlot.grid()
            PyPlot.xlabel("Iteration number")
            PyPlot.ylabel("residual norm")
            PyPlot.semilogy(v(hist_cg_jacobi), label="cg_jacobi",color=:cyan)
            PyPlot.semilogy(v(hist_cg_ilu0), label="cg_ilu0",    color=:red)
            PyPlot.semilogy(v(hist_cg_ilu1), label="cg_ilu1",    color=:magenta)
            PyPlot.semilogy(v(hist_cg_ilu2), label="cg_ilu2",    color=:blue)
            PyPlot.semilogy(v(hist_cg_rsamg), label="cg_rsamg",  color=:orange)
            PyPlot.semilogy(v(hist_cg_aggamg), label="cg_aggamg",color=:black)
            PyPlot.legend(loc="upper right")
            return
        end

    end
    if nref1>nref0
        PyPlot.clf()
        PyPlot.subplot(121)
        PyPlot.xlabel("unknowns")
        PyPlot.ylabel("CPU time/s")
        # PyPlot.loglog(all_n, all_t_tria, label="tria",color=:
↪yellow,marker="o")
        # PyPlot.loglog(all_n, all_t_asm, label="asm",color=:gray,marker="o")
        PyPlot.loglog(all_n, all_t_direct,       label="direct",color=:
↪green,marker="x",markersize=8)
        PyPlot.loglog(all_n, all_t_cg_jacobi, label="cg_jacobi",color=:
↪cyan,marker="o")
        PyPlot.loglog(all_n, all_t_cg_ilu0    , label="cg_ilu0",color=:
↪red,marker="o")
        PyPlot.loglog(all_n, all_t_cg_ilu1,     label="cg_ilu1",color=:
↪magenta,marker="o")
        PyPlot.loglog(all_n, all_t_cg_ilu2,     label="cg_ilu2",color=:
↪blue,marker="o")
```

```
        PyPlot.loglog(all_n, all_t_cg_aggamg,   label="cg_aggamg",color=:
↪orange,marker="o")
        PyPlot.loglog(all_n, all_t_cg_rsamg,     label="cg_rsamg",color=:
↪black,marker="o")
        PyPlot.loglog(all_n, 1.0e-6.*all_n.^(3/2), label="O(N^(3/2))",color=:
↪red)
        PyPlot.grid()
        PyPlot.legend()
        PyPlot.subplot(122)
        PyPlot.xlabel("unknowns")
        PyPlot.ylabel("iterations")
        PyPlot.loglog(all_n, all_n_cg_jacobi, label="cg_jacobi",color=:
↪cyan,marker="o")
        PyPlot.loglog(all_n, all_n_cg_ilu0    , label="cg_ilu0",color=:
↪red,marker="o")
        PyPlot.loglog(all_n, all_n_cg_ilu1,     label="cg_ilu1",color=:
↪magenta,marker="o")
        PyPlot.loglog(all_n, all_n_cg_ilu2,     label="cg_ilu2",color=:
↪blue,marker="o")
        PyPlot.loglog(all_n, all_n_cg_aggamg,   label="cg_aggamg",color=:
↪orange,marker="o")
        PyPlot.loglog(all_n, all_n_cg_rsamg,     label="cg_rsamg",color=:
↪black,marker="o")
        PyPlot.loglog(all_n, 1.0e1.*all_n.^(1/2), label="O(N^(1/2))",color=:red)
        PyPlot.loglog(all_n, 1.0*all_n.^(1/4), label="O(N^(1/4))",color=:green)
        PyPlot.grid()
        PyPlot.legend()
    end
end
```

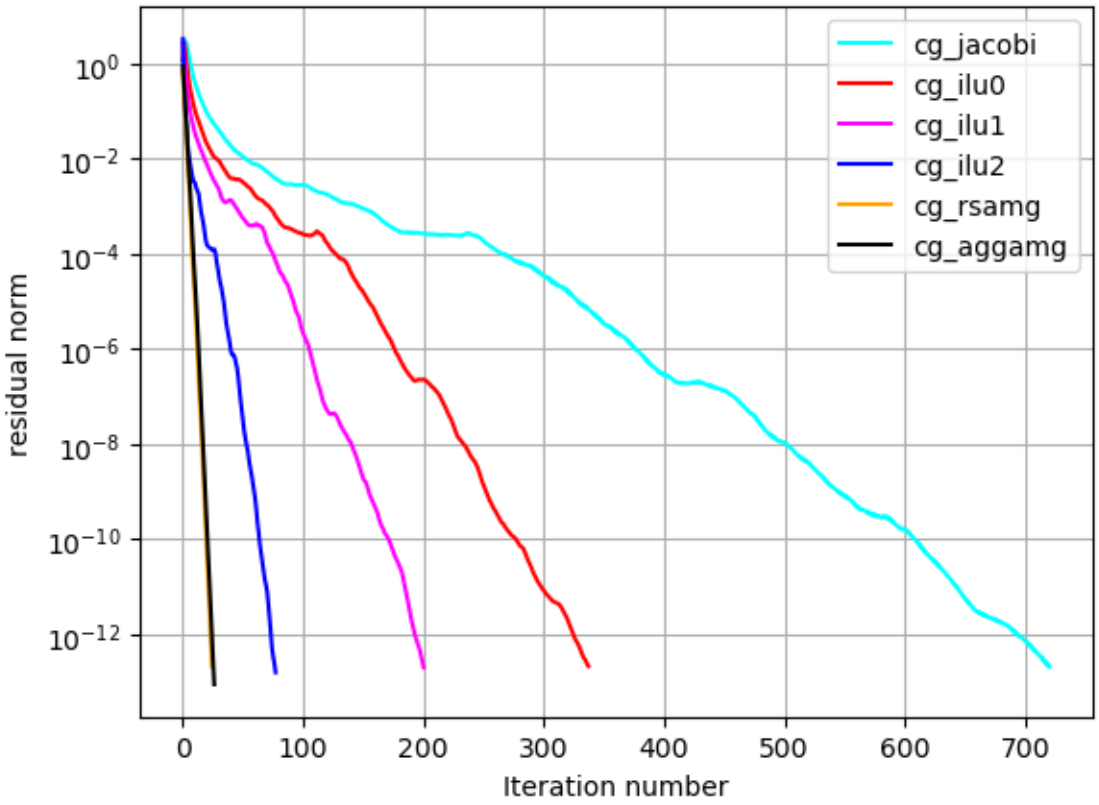
[6]: precontest (generic function with 1 method)

0.4 Comparative plot of convetgence history

[7]:
```
precontest(nref0=5,plothist=true)
```

```
grid:
elapsed CPU time: 0.019955 seconds

assembly:
elapsed CPU time: 0.071709 seconds

direct:
elapsed CPU time: 0.132753 seconds

cg_jacobi:
```

```
elapsed CPU time: 0.252324 seconds
iterations: 721

cg_ilu0:
elapsed CPU time: 0.265684 seconds
iterations: 338

cg_ilut, =0.2
elapsed CPU time: 0.338743 seconds
iterations: 201

cg_ilut, =0.02
elapsed CPU time: 0.148962 seconds
iterations: 78

cg_aggamg:
elapsed CPU time: 0.955798 seconds
iterations: 27

cg_rsamg:
elapsed CPU time:
```



```
0.530197 seconds
iterations: 25
```

0.5 Comparative plot of convergence rates

[8]:
```
precontest(nref0=0,nref1=6, plotscale=true)
```

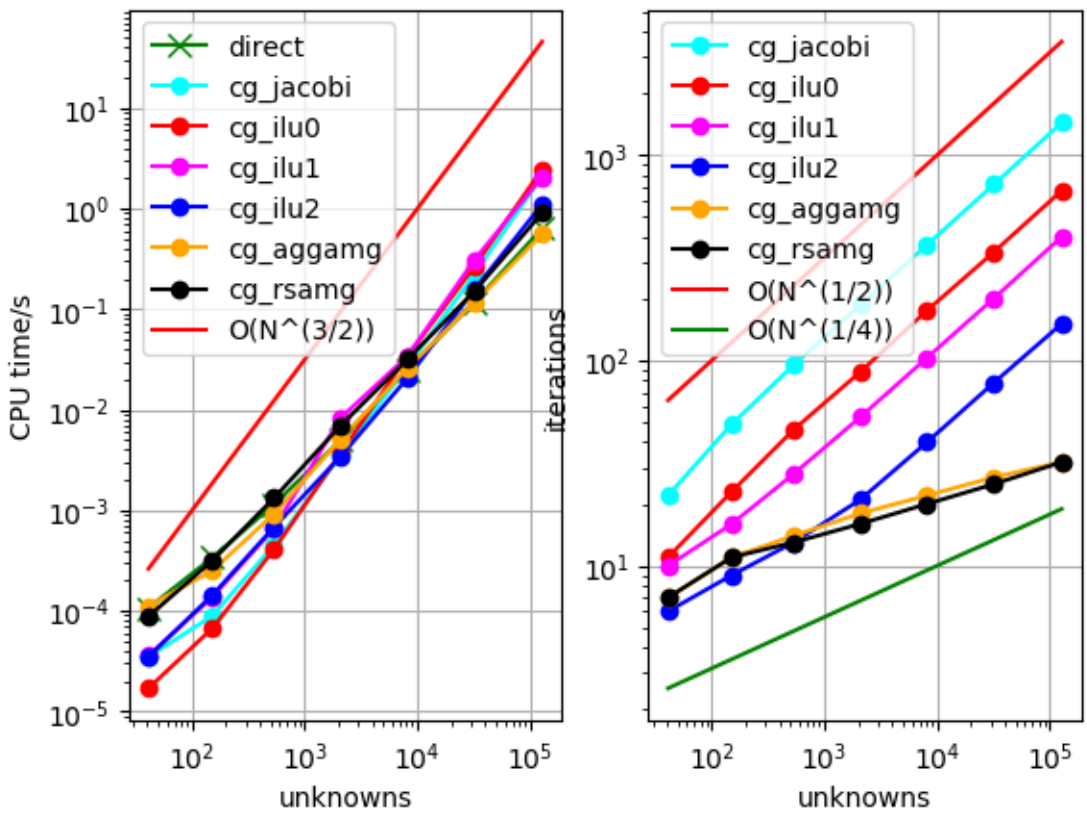
```
grid:
elapsed CPU time: 9.0e-5 seconds

assembly:
elapsed CPU time: 7.2e-5 seconds

direct:
elapsed CPU time: 0.000107 seconds

cg_jacobi:
elapsed CPU time: 3.5e-5 seconds
iterations: 22

cg_ilu0:
elapsed CPU time: 1.7e-5 seconds
iterations: 11

cg_ilut, =0.2
elapsed CPU time: 3.6e-5 seconds
iterations: 10

cg_ilut, =0.02
elapsed CPU time: 3.5e-5 seconds
iterations: 6

cg_aggamg:
elapsed CPU time: 0.000112 seconds
iterations: 7

cg_rsamg:
elapsed CPU time: 8.8e-5 seconds
iterations: 7

grid:
elapsed CPU time: 0.00015 seconds

assembly:
elapsed CPU time: 0.000235 seconds

direct:
```

```
elapsed CPU time: 0.000338 seconds

cg_jacobi:
elapsed CPU time: 8.9e-5 seconds
iterations: 49

cg_ilu0:
elapsed CPU time: 6.7e-5 seconds
iterations: 23

cg_ilut, =0.2
elapsed CPU time: 0.000137 seconds
iterations: 16

cg_ilut, =0.02
elapsed CPU time: 0.000143 seconds
iterations: 9

cg_aggamg:
elapsed CPU time: 0.000251 seconds
iterations: 11

cg_rsamg:
elapsed CPU time: 0.000316 seconds
iterations: 11

grid:
elapsed CPU time: 0.000381 seconds

assembly:
elapsed CPU time: 0.000801 seconds

direct:
elapsed CPU time: 0.001147 seconds

cg_jacobi:
elapsed CPU time: 0.000452 seconds
iterations: 95

cg_ilu0:
elapsed CPU time: 0.000409 seconds
iterations: 46

cg_ilut, =0.2
elapsed CPU time: 0.000662 seconds
iterations: 28

cg_ilut, =0.02
```

```
elapsed CPU time: 0.000682 seconds
iterations: 13

cg_aggamg:
elapsed CPU time: 0.000922 seconds
iterations: 14

cg_rsamg:
elapsed CPU time: 0.001325 seconds
iterations: 13

grid:
elapsed CPU time: 0.001393 seconds

assembly:
elapsed CPU time: 0.003024 seconds

direct:
elapsed CPU time: 0.004981 seconds

cg_jacobi:
elapsed CPU time: 0.003528 seconds
iterations: 186

cg_ilu0:
elapsed CPU time: 0.003713 seconds
iterations: 88

cg_ilut, =0.2
elapsed CPU time: 0.008318 seconds
iterations: 53

cg_ilut, =0.02
elapsed CPU time: 0.003438 seconds
iterations: 21

cg_aggamg:
elapsed CPU time: 0.005118 seconds
iterations: 18

cg_rsamg:
elapsed CPU time: 0.006967 seconds
iterations: 16

grid:
elapsed CPU time: 0.005063 seconds

assembly:
```

```
elapsed CPU time: 0.012137 seconds

direct:
elapsed CPU time: 0.025094 seconds

cg_jacobi:
elapsed CPU time: 0.027095 seconds
iterations: 366

cg_ilu0:
elapsed CPU time: 0.034065 seconds
iterations: 175

cg_ilut, =0.2
elapsed CPU time: 0.034003 seconds
iterations: 103

cg_ilut, =0.02
elapsed CPU time: 0.020643 seconds
iterations: 40

cg_aggamg:
elapsed CPU time: 0.025858 seconds
iterations: 22

cg_rsamg:
elapsed CPU time: 0.031607 seconds
iterations: 20

grid:
elapsed CPU time: 0.019866 seconds

assembly:
elapsed CPU time: 0.048944 seconds

direct:
elapsed CPU time: 0.117949 seconds

cg_jacobi:
elapsed CPU time: 0.221679 seconds
iterations: 721

cg_ilu0:
elapsed CPU time: 0.265732 seconds
iterations: 338

cg_ilut, =0.2
elapsed CPU time: 0.30942 seconds
```

```
iterations: 201

cg_ilut, =0.02
elapsed CPU time: 0.157063 seconds
iterations: 78

cg_aggamg:
elapsed CPU time: 0.116641 seconds
iterations: 27

cg_rsamg:
elapsed CPU time: 0.150854 seconds
iterations: 25

grid:
elapsed CPU time: 0.083859 seconds

assembly:
elapsed CPU time: 0.194668 seconds

direct:
elapsed CPU time: 0.632325 seconds

cg_jacobi:
elapsed CPU time: 2.107732 seconds
iterations: 1440

cg_ilu0:
elapsed CPU time: 2.461818 seconds
iterations: 672

cg_ilut, =0.2
elapsed CPU time: 2.057781 seconds
iterations: 400

cg_ilut, =0.02
elapsed CPU time: 1.09832 seconds
iterations: 151

cg_aggamg:
elapsed CPU time: 0.554959 seconds
iterations: 32

cg_rsamg:
elapsed CPU time:
```




```
0.903939 seconds
iterations: 32
```

[8]: PyObject <matplotlib.legend.Legend object at 0x7f4ec341def0>

This notebook was generated using Literate.jl.