ngsPETSc: NETGEN meets PETSc

P. E. Farrell*, S. Zampini†, U. Zerbinati*

* *Mathematical Institute*
University of Oxford

† *Extreme Computing Research Center*
King Abdullah University of Science and Technology

PETSc User Meeting, 23rd of May 2024, Cologne

UNIVERSITY OF
OXFORD

Mathematical
Institute

Oxford
Mathematics

- ▶ We will see how to use **PETSc KSP** to solve linear systems in **NGSolve**. We will also how to impose a near nullspace.

- ▶ We will see how to use **PETSc PC** as a preconditioners building block inside **NGSolve**. In particular, we will see how to use Hypre in a vertex patch preconditioner and as an auxiliary space preconditioner.

- ▶ We will see how to use **PETSc SNES** to solve non-linear problems in **NGSolve**. In particular, we will see how to solve the Naghdi shell problem.

All codes are available on Github:
https://github.com/UZerbinati/PETSc24

Netgen is an advancing front 2D/3D-mesh generator, with many interesting features.

- ▶ The geometry we intend to mesh can be described by **Constructive Solid Geometry** (CSG), in particular we can use **Opencascade** to describe our geometry.
- ▶ It is able to construct **isoparametric meshes** reppresentation, which conform to the geometry.
- ▶ A wide variety of mesh splits are available also for curved geometries, such as Alfeld splits and Powell-Sabin splits.
- ▶ High flexibility in the mesh generation and mesh refinement.

NGSolve is a high-performance multiphysics finite element software with an extremely flexible Python interface.

▶ Wide range of finite elements available, including and not limited to hierarchical H^1 elements, $H(div)$ Raviart-Thomas and Brezzi-Douglas-Marini elements, and $H(curl)$ Nedelec elements.

▶ The variational formulation can be easily defined using an analogous language to the unified form language (UFL).

▶ Many extensions are available, including **ngsxfem** for unfitted finite element discretizations, **ngsTreffetz** for Treffetz methods and **ngsTents** for spacetime tents schemes.

ngsPETSc is an interface between NETGEN/NGSolve and
PETSc. In particular, **ngsPETSc** provides new capabilities to
NETGEN/**NGSolve** such as:

▶ Access to all linear solver capabilities of **KSP**.

▶ Access to all preconditioning capabilities of **PC**.

▶ Access to all non-linear solver capabilities of **SNES**.

▶ Access to all mesh refinement capabilities of **DMPLEX**.

# PETSc KSP

An NGsolve Example – Poisson

```
1   from ngsolve import *
2   import netgen.gui
3   import netgen.meshing as ngm
4   from mpi4py.MPI import COMM_WORLD
5
6   if COMM_WORLD.rank == 0:
7       mesh = Mesh(unit_square.GenerateMesh(maxh=0.2).
    Distribute(COMM_WORLD))
8   else:
9       mesh = Mesh(ngm.Mesh.Receive(COMM_WORLD))
10  fes = H1(mesh, order=3, dirichlet="left|right|top|
    bottom")
11  u,v = fes.TnT()
12  a = BilinearForm(grad(u)*grad(v)*dx).Assemble()
13  f = LinearForm(fes)
14  f += 32 * (y*(1-y)+x*(1-x)) * v * dx
```

PETSc KSP – Direct solve with MUMPS

▶ We can perform a direct solve using MUMPS.

```
1
2    from ngsPETSc import KrylovSolver
3    opts = {'ksp_type': 'preonly',
4            'pc_type': 'lu',
5            'pc_factor_mat_solver_type
     ': 'mumps'}
6    solver = KrylovSolver(a,fes,
     solverParameters=opts)
7    gfu = solver.solve(f)
8    Draw(gfu,mesh, "solution")
```
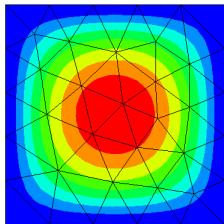

Solution of Poisson problem
computed with MUMPS

PETSc KSP – Iterative Jacobi method

▶ We can use a wide variety of iterative solvers, for example, the Jacobi method, i.e.

$$x^{(k+1)} = D^{-1}(b - (A - D)x^{(k)}).$$

```python
opts = {'ksp_type': 'richardson',
        'ksp_richardson_scale':  1.0,
        'pc_type': 'jacobi'}
solver = KrylovSolver(a,fes, solverParameters=opts)
gfu = solver.solve(f)
Draw(gfu,mesh, "solution")
```

▶ Analogously we can implement the Gauss-Seidel method.

PETSc KSP – Galerkin Algebraic MultiGrid (GAMG)

▶ Inside of a classical iterative method such as conjugate gradient, we can play with different preconditioners such as PETSc GAMG.

```
1    opts = {'ksp_type': 'cg',
2            'pc_type': 'gamg'}
3    solver = KrylovSolver(a,fes, solverParameters=opts)
4    gfu = solver.solve(f)
```

▶ As we will see in a moment we have a wide variety of preconditioners available, such as: **Hypre (AMG)**, **BDDC**, **...**

```
1    E, nu = 210, 0.2
2    mu  = E / 2 / (1+nu)
3    lam = E * nu / ((1+nu)*(1-2*nu))
4
5    def Stress(strain):
6        return 2*mu*strain + lam*Trace(strain)*Id(2)
7
8    fes = VectorH1(mesh, order=1, dirichlet="left")
9    u,v = fes.TnT()
10
11   a = BilinearForm(InnerProduct(Stress(Sym(Grad(u))),
      Sym(Grad(v)))*dx)
12   a.Assemble()
13
14   force = CF( (0,1) )
15   f = LinearForm(force*v*ds("right")).Assemble()
```

▶ We can pass a near nullspace to a **KrylovSolver**, informing the solver that there is a near nullspace.

```
1   from ngsPETSc import KrylovSolver,
      NullSpace
2   rbms = []
3   for val in [(1,0), (0,1), (-y,x)]:
4       rbm = GridFunction(fes)
5       rbm.Set(CF(val))
6       rbms.append(rbm.vec)
7   nullspace = NullSpace(fes, rbms,
     near=True)
8   opts = {'ksp_type': 'cg',
9               'pc_type': 'gamg'}
```
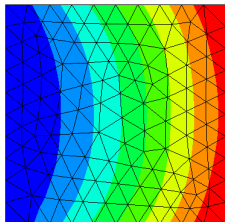
Solution of lienar elasticity fixing SO(3) to be in the near nullspace.

```
1   solver = KrylovSolver(a,fes, solverParameters=opts,
      nullspace=nullspace)
2   gfu = solver.solve(f)
```

PETSc PC

PETSc PC – Hypre

▶ We can use PETSc preconditioners as normal preconditioners in NGSolve, for example we can wrap a PETSc PC of type Hypre in NGSolve and use it inside NGSolve Krylov solvers.

```
1    from ngsPETSc import pc
2    from ngsolve.krylovspace import CG
3    pre = Preconditioner(a, "PETScPC", pc_type="hypre")
4    gfu = GridFunction(fes)
5    gfu.vec.data = CG(a.mat, rhs=f.vec, pre=pre.mat,
     printrates=True)
6    Draw(gfu)
```

Degrees of Freedom (p=1)	7329	1837569
PETSc PC (HYPRE)	22 (5.19e-13)	31 (6.82e-13)
NGSolve Geometric MultiGrid	14 (4.08e-13)	16 (1.30e-12)

PETSc PC – Hypre

▶ We can use PETSc preconditioners as normal preconditioners in NGSolve, for example we can wrap a PETSc PC of type Hypre in NGSolve and use it inside NGSolve Krylov solvers.

```
1   from ngsPETSc import pc
2   from ngsolve.krylovspace import CG
3   pre = Preconditioner(a, "PETScPC", pc_type="hypre")
4   gfu = GridFunction(fes)
5   gfu.vec.data = CG(a.mat, rhs=f.vec, pre=pre.mat,
    printrates=True)
6   Draw(gfu)
```

Degrees of Freedom (p=3)	64993	259009
PETSc PC (HYPRE)	40 (6.48e-13)	69 (2.53e-13)
NGSolve Geometric MultiGrid	19 (8.89e-13)	19 (7.78e-13)

▶ We can use PETSc preconditioner as one of the building blocks of a more complex preconditioner. For example, we can use it as a two-level additive Schwarz preconditioner. In this case, we will use as fine space correction, the inverse of the local matrices associated with the patch of a vertex, i.e.

$$\mathcal{P} = \sum_{i=1}^{n} I_i A_i^{-1} I_i^T.$$

```
1    blocks = VertexPatchBlocks ( mesh , fes )
2    blockjac = a . mat . CreateBlockSmoother ( blocks )
3    gfu . vec . data = CG ( a . mat , rhs = f . vec , pre = blockjac ,
     printrates = True )
4    Draw ( gfu )
```

PETSc PC – Two level additive Schwarz

► We can also use the PETSc PC inside a two-level additive Schwarz preconditioner. In particular, we will use a PETSc PC of type HYPRE to do a coarse grid correction on the vertex degree of freedom.

$$\mathcal{P} = I_H A_H^{-1} I_H^T + \sum_{i=1}^{n} I_i A_i^{-1} I_i^T.$$

```
1    vertexdofs = VertexDofs(mesh, fes)
2    preCoarse = Preconditioner(a, "PETScPC", pc_type="
     hypre", restrictedTo=vertexdofs)
3    pretwo = preCoarse.mat + blockjac
4    gfu.vec.data = CG(a.mat, rhs=f.vec, pre=pretwo,
     printrates=True)
```

An NGsolve Example – Discontinuous Galerkin

```
1    fesDG = L2(mesh, order=3, dgjumps=True)
2    u,v = fesDG.TnT()
3    aDG = BilinearForm(fesDG)
4    jump_u = u-u.Other(); jump_v = v-v.Other()
5    n = specialcf.normal(2)
6    mean_dudn = 0.5*n * (grad(u)+grad(u.Other()))
7    mean_dvdn = 0.5*n * (grad(v)+grad(v.Other()))
8    alpha = 4
9    h = specialcf.mesh_size
10   aDG = BilinearForm(fesDG)
11   aDG += grad(u)*grad(v) * dx
12   aDG += alpha*3**2/h*jump_u*jump_v * dx(skeleton=
     True)
13   aDG += alpha*3**2/h*u*v * ds(skeleton=True)
14   aDG += (-mean_dudn*jump_v -mean_dvdn*jump_u)*dx(
     skeleton=True)
15   aDG += (-n*grad(u)*v-n*grad(v)*u)*ds(skeleton=True)
16   fDG = LinearForm(fesDG)
17   fDG += 1*v * dx
18   aDG.Assemble()
```

PETSc PC – Auxiliary Space Preconditioner

▶ We can now use the PETSc PC assembled for the conforming Poisson problem as an auxiliary space preconditioner for the DG discretisation. In particular, we will use as smoother a PETSc PC of type SOR.

```
1    from ngsPETSc import pc
2    smoother = Preconditioner(aDG, "PETScPC", pc_type="
     sor")
3    transform = fes.ConvertL2Operator(fesDG)
4    preDG = transform @ pre.mat @ transform.T +
     smoother.mat
5    gfuDG = GridFunction(fesDG)
6    gfuDG.vec.data = CG(aDG.mat, rhs=fDG.vec, pre=preDG
     , printrates=True)
```

Saddle Point Problems

```
1    V = VectorH1 ( mesh , order =4 , dirichlet =" wall | inlet |
     cyl ")
2    Q = H1 ( mesh , order =3)
3    u , v = V . TnT () ; p , q = Q . TnT ()
4    a = BilinearForm ( InnerProduct ( Grad ( u ) , Grad ( v ) ) * dx +1
     e1 * div ( u ) * div ( v ) * dx )
5    a . Assemble ()
6    b = BilinearForm ( div ( u ) * q * dx ) . Assemble ()
7    gfu = GridFunction (V , name =" u ")
8    gfp = GridFunction (Q , name =" p ")
9    uin = CoefficientFunction ( (1.5*4* y *(0.41 - y )
     /(0.41*0.41) , 0) )
10   gfu . Set ( uin , definedon = mesh . Boundaries (" inlet ") )
```
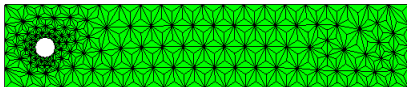
It is well known that a field split preconditioner can be used to solve saddle point problems, i.e.

$$\begin{bmatrix} \hat{A}^{-1} & 0 \\ 0 & -B\hat{A}^{-1}B^T \end{bmatrix}$$

Thanks to the inf-sup condition we can prove that the Schur complement is spectrally equivalent to the mass matrix, hence we can use as preconditioner:

$$\begin{bmatrix} \hat{A}^{-1} & 0 \\ 0 & -\nu\hat{M}^{-1} \end{bmatrix}$$

where M is the mass matrix.

```
1    m = BilinearForm(p*q*dx)
2    K = BlockMatrix( [ [a.mat, b.mat.T], [b.mat, None]
     ] )
3    from ngsPETSc import pc
4    pre = Preconditioner(a, "PETScPC", pc_type="hypre")
5    mp = Preconditioner(m, "bddc")
6    m.Assemble()
7    C = BlockMatrix( [ [pre.mat, None], [None, mp.mat]
     ] )
8    rhs = BlockVector ( [f.vec, g.vec] )
9    sol = BlockVector ( [gfu.vec, gfp.vec] )
10   solvers.MinRes (mat=K, pre=C, rhs=rhs, sol=sol,
11                     printrates=True, initialize=False,
```
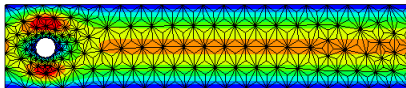
PETSc SNES

An NGsolve Example – Naghdi Shell

```python
1  geo = CSGeometry ()
2  cyl   = Cylinder ( Pnt (0 ,0 ,0) , Pnt
       (1 ,0 ,0) ,0.4) . bc ( "cyl" )
3  left  = Plane ( Pnt (0 ,0 ,0) , Vec ( -1 ,0 ,0)
       )
4  right = Plane ( Pnt (1 ,0 ,0) , Vec (1 ,0 ,0) )
5  finitecyl = cyl * left * right
6  geo . AddSurface ( cyl , finitecyl )
7  geo . NameEdge ( cyl , left , "left" )
8  geo . NameEdge ( cyl , right , "right" )
9  if MPI . COMM_WORLD . rank == 0:
10     mesh = Mesh ( geo . GenerateMesh ( maxh
       =0.3) . Distribute ( MPI . COMM_WORLD ))
11 else :
12     mesh = Mesh ( ngm . Mesh . Receive ( MPI .
       COMM_WORLD ))
13 mesh . Curve ( order )
```
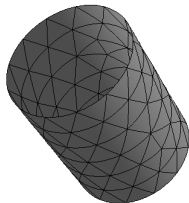
Naghdi shell undeformed
geometry.

An NGsolve Example – Naghdi Shell


```
1 nsurf = specialcf.normal(3)
2 thickness = 0.1
3 Ptau = Id(3) - OuterProduct(nsurf,nsurf)
4 Ftau = grad(u).Trace() + Ptau
5 Ctautau = Ftau.trans * Ftau
6 Etautau = 0.5*(Ctautau - Ptau)
7 eps_beta = Sym(Ptau*grad(beta).Trace())
8 gradu = grad(u).Trace()
9 ngradu = gradu.trans*nsurf
10 gfn = nsurf
11 a = BilinearForm(fes, symmetric=True)
12 a += Variation( thickness*InnerProduct(Etautau,
      Etautau)*ds )
13 a += Variation( 0.5*thickness**3*InnerProduct(eps_beta
      -Sym(gradu.trans*grad(gfn)),eps_beta-Sym(gradu.
      trans*grad(gfn)))*ds )
14 a += Variation( thickness*(ngradu-beta)*(ngradu-beta)*
      ds )
15 factor = Parameter(0.0)
16 a += Variation( -thickness*factor*y*u[1]*ds )
```

▶ We can use PETSc SNES to solve the non-linear Naghdi shell problem.

```
1    factor.Set (1.5*(loadstep+1))
2    opts = {"snes_type": "newtonls",
3    "snes_max_it": 10,
4    "snes_monitor": "",
5    "ksp_monitor": "",
6    "pc_type": "lu"}
7    solver = NonLinearSolver(fes, a=a
     ,solverParameters=opts)
8    gfu = solver.solve(gfu)
```
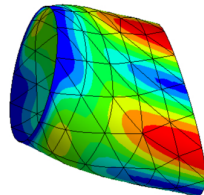
Solution of Naghdi shell problem.

PETSc DMPLEX

Firedrake is an automated system for the solution of partial differential equations using the finite element method (FEM).

▶ Variational formulation can be easily defined using the **UFL** language.

▶ Wide class of finite elements are available, including $H(div)$, $H(curl)$, H^1 and H^2.

▶ Provides access to **PETSc** linear solvers and non-linear solvers.

University of OXFORD

Mathematical Institute

ngsPETSc provides new capabilities to **Firedrake** such as:

▶ Access to all Netgen generated linear meshes and high order meshes.

▶ Splits for macro elements, such as Alfeld splits and Powell-Sabin splits (even on curved geometries).

▶ Adaptive mesh refinement capabilities, that conform to the geometry.

▶ High order mesh hierarchies for multigrid solvers.

Conclusions

▶ Come to the 5th NGSolve User Meeting that will be held between the **17th of June** and the **19th of June** at **TU Wien**.

Usally there are beers !