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

Overview

- 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/NGSolve

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 – NETGEN/NGSolve

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

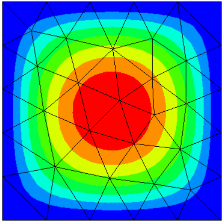
An NGsolve Example – Poisson

```python
from ngsolve import *
import netgen.gui
import netgen.meshing as ngm
from mpi4py.MPI import COMM_WORLD

if COMM_WORLD.rank == 0:
    mesh = Mesh(unit_square.GenerateMesh(maxh=0.2).Distribute(COMM_WORLD))
else:
    mesh = Mesh(ngm.Mesh.Receive(COMM_WORLD))
fes = H1(mesh, order=3, dirichlet="left|right|top|bottom")
u,v = fes.TnT()
a = BilinearForm(grad(u)*grad(v)*dx).Assemble()
f = LinearForm(fes)
f += 32 * (y*(1-y)+x*(1-x)) * v * dx
```

PETSc KSP – Direct solve with MUMPS

- We can perform a direct solve using MUMPS.

```
from ngsPETSc import KrylovSolver
opts = {'ksp_type': 'preonly',
        'pc_type': 'lu',
        'pc_factor_mat_solver_type': 'mumps'}
solver = KrylovSolver(a,fes, solverParameters=opts)
gfu = solver.solve(f)
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.

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

```
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.

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

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

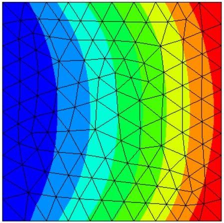
An NGsolve Example – Linear Elasticity

```python
E, nu = 210, 0.2
mu  = E / 2 / (1+nu)
lam = E * nu / ((1+nu)*(1-2*nu))

def Stress(strain):
   return 2*mu*strain + lam*Trace(strain)*Id(2)

fes = VectorH1(mesh, order=1, dirichlet="left")
u,v = fes.TnT()

a = BilinearForm(InnerProduct(Stress(Sym(Grad(u))),
  Sym(Grad(v)))*dx)
a.Assemble()

force = CF( (0,1) )
f = LinearForm(force*v*ds("right")).Assemble()
```

PETSc KSP – Near Nullspace

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

```python
from ngsPETSc import KrylovSolver,
  NullSpace
rbms = []
for val in [(1,0), (0,1), (-y,x)]:
    rbm = GridFunction(fes)
    rbm.Set(CF(val))
    rbms.append(rbm.vec)
nullspace = NullSpace(fes, rbms,
 near=True)
opts = {'ksp_type': 'cg',
        'pc_type': 'gamg'}
```



Solution of lienar elasticity fixing SO(3) to be in the near nullspace.

```python
solver = KrylovSolver(a,fes, solverParameters=opts,
  nullspace=nullspace)
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.

```python
from ngsPETSc import pc
from ngsolve.krylovspace import CG
pre = Preconditioner(a, "PETScPC", pc_type="hypre")
gfu = GridFunction(fes)
gfu.vec.data = CG(a.mat, rhs=f.vec, pre=pre.mat,
 printrates=True)
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.

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

PETSc PC – Vertex Patch

- 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.

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

```
blocks = VertexPatchBlocks(mesh, fes)
blocksmooth = a.mat.CreateBlockSmoother(blocks)
gfu.vec.data = CG(a.mat, rhs=f.vec, pre=blockjac,
 printrates=True)
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.

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

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

An NGsolve Example – Discontinuous Galerkin

```
fesDG = L2(mesh, order=3, dgjumps=True)
u,v = fesDG.TnT()
aDG = BilinearForm(fesDG)
jump_u = u-u.Other(); jump_v = v-v.Other()
n = specialcf.normal(2)
mean_dudn = 0.5*n * (grad(u)+grad(u.Other()))
mean_dvdn = 0.5*n * (grad(v)+grad(v.Other()))
alpha = 4
h = specialcf.mesh_size
aDG = BilinearForm(fesDG)
aDG += grad(u)*grad(v) * dx
aDG += alpha*3**2/h*jump_u*jump_v * dx(skeleton=True)
aDG += alpha*3**2/h*u*v * ds(skeleton=True)
aDG += (-mean_dudn*jump_v -mean_dvdn*jump_u)*dx(skeleton=True)
aDG += (-n*grad(u)*v-n*grad(v)*u)*ds(skeleton=True)
fDG = LinearForm(fesDG)
fDG += 1*v * dx
aDG.Assemble()
```

PETSc PC – Auxiliary Space Preconditioner

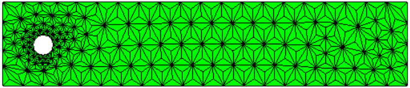
- We can now use the PETSc PC assembled for the conforming Poisson problem as an auxiliary space preconditioner for the DG discretisation. In particular, we will use as smoother a PETSc PC of type SOR.

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

Saddle Point Problems

An NGsolve Example – Stokes flow

```
V = VectorH1(mesh, order=4, dirichlet="wall|inlet|cyl")
Q = H1(mesh, order=3)
u,v = V.TnT(); p,q = Q.TnT()
a = BilinearForm(InnerProduct(Grad(u),Grad(v))*dx+1e1*div(u)*div(v)*dx)
a.Assemble()
b = BilinearForm(div(u)*q*dx).Assemble()
gfu = GridFunction(V, name="u")
gfp = GridFunction(Q, name="p")
uin = CoefficientFunction( (1.5*4*y*(0.41-y)/(0.41*0.41), 0) )
gfu.Set(uin, definedon=mesh.Boundaries("inlet"))
```



Fieldsplit Schur preconditioner – Mass matrix

UNIVERSITY OF OXFORD
Mathematical Institute

It is well known that a field split preconditioner can be used to solve saddle point problems, i.e.

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

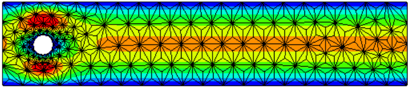
Thanks to the inf-sup condition we can prove that the Schur complement is spectrally equivalent to the mass matrix, hence we can use as preconditioner:

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

where M is the mass matrix.

PETSc PC – NGSolve Fieldsplit

```
m = BilinearForm(p*q*dx)
K = BlockMatrix( [ [a.mat, b.mat.T], [b.mat, None] ] )
from ngsPETSc import pc
pre = Preconditioner(a, "PETScPC", pc_type="hypre")
mp = Preconditioner(m, "bddc")
m.Assemble()
C = BlockMatrix( [ [pre.mat, None], [None, mp.mat] ] )
rhs = BlockVector ( [f.vec, g.vec] )
sol = BlockVector( [gfu.vec, gfp.vec] )
solvers.MinRes (mat=K, pre=C, rhs=rhs, sol=sol,
                printrates=True, initialize=False,
```



Navier-Stokes flow

A more interesting example is the Navier-Stokes flow, which is a non-linear problem. In particular, we will consider the problem of finding $(\boldsymbol{u}, p) \in H^1(\Omega)^2 \times L^2(\Omega)$ such that

$$\int_\Omega \partial_t \boldsymbol{u} \cdot \boldsymbol{v} + \int_\Omega (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} \cdot \boldsymbol{v} + \nu \int_\Omega \nabla \boldsymbol{u} : \nabla \boldsymbol{v} - \int_\Omega p \nabla \cdot \boldsymbol{v} = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v}$$

$$- \int_\Omega q \nabla \cdot \boldsymbol{u} = 0$$

for all $(\boldsymbol{v}, q) \in H^1(\Omega)^2 \times L^2(\Omega)$.

Navier-Stokes flow – Augmented Lagrangian

We consider an augmented Lagrangian formulation for the discrete problem, i.e. find $(\boldsymbol{u}_h, p_h) \in V_h \times Q_h$ such that

$$
\begin{aligned}
(\partial_t \boldsymbol{u}, \boldsymbol{v})_0 + (\boldsymbol{u} \cdot \nabla \boldsymbol{u}, \boldsymbol{v})_0 + \nu(\nabla \boldsymbol{u}, \nabla \boldsymbol{v})_0 & \\
- (p, \nabla \cdot \boldsymbol{v})_0 + \gamma(\nabla \cdot \boldsymbol{u}, \nabla \cdot \boldsymbol{v})_0 &= (\boldsymbol{f}, \boldsymbol{v})_0
\end{aligned}
$$

and verifying the weak divergence free constraint $(\nabla \cdot \boldsymbol{u}, q)_0 = 0$, for all $(\boldsymbol{v}, q) \in V_h \times Q_h$.

Navier-Stokes flow – Fieldsplit Schour preconditioner

The linearized version of the Navier-Stokes equations can be written in matrix form as

$$\begin{bmatrix} A+\gamma B^T W B & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ p \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ 0 \end{bmatrix}$$

We choose as preconditioner the **fieldsplit Schur preconditioner**, i.e.

$$\begin{bmatrix} I & -\hat{A}_\gamma^{-1} B^T \\ 0 & I \end{bmatrix} \begin{bmatrix} \hat{A}_\gamma^{-1} & 0 \\ 0 & \hat{S}_\gamma^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -BA_\gamma^{-1} & I \end{bmatrix}$$

$$A_\gamma = A + \gamma B^T W B \qquad S_\gamma = -BA_\gamma^{-1} B^T.$$

Navier-Stokes flow – Augmented Lagrangian

- The augmented Lagrangian term helps enforce the divergence-free constraint, and makes the scheme pressure robust.
- We can use as preconditioner

$$\begin{bmatrix} \hat{A}_\gamma^{-1} & 0 \\ 0 & -(\nu + \gamma) M^{-1} \end{bmatrix}$$

- How do we compute $\hat{A}_\gamma^{-1}$ efficiently ? Can we adopt a multigrid approach ?

Robust relaxation via FEEC – Hood–Taylor

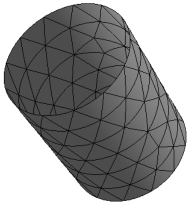
$$
\begin{tikzcd}
0 \arrow[r] & \left[H^2(\Omega)\right]^2 \arrow[r, "\nabla\times"] \arrow[d] & \left[H^1_0(\Omega)\right]^2 \arrow[r, "\nabla\cdot"] \arrow[d] & L^2(\Omega) \arrow[r] \arrow[d] & 0 \\
0 \arrow[r] & \left[\mathbb{P}^5(\mathcal{T}_h)\right]^2 \arrow[r, "\nabla\times"] & \left[\mathbb{P}^4(\mathcal{T}_h)\right]^2 \arrow[r, "\nabla\cdot"] & \mathbb{P}^3_{disc}(\mathcal{T}_h) \arrow[r] & 0
\end{tikzcd}
$$

- Thanks to the FEEC complex we can construct a robust relaxation and prolongation operator for the Hood–Taylor element.

PETSc SNES

An NGsolve Example – Naghdi Shell

```python
geo = CSGeometry()
cyl   = Cylinder(Pnt(0,0,0),Pnt(1,0,0),0.4).bc("cyl")
left  = Plane(Pnt(0,0,0), Vec(-1,0,0))
right = Plane(Pnt(1,0,0), Vec(1,0,0))
finitecyl = cyl * left * right
geo.AddSurface(cyl, finitecyl)
geo.NameEdge(cyl,left, "left")
geo.NameEdge(cyl,right, "right")
if MPI.COMM_WORLD.rank == 0:
    mesh = Mesh(geo.GenerateMesh(maxh=0.3).Distribute(MPI.COMM_WORLD))
else:
    mesh = Mesh(ngm.Mesh.Receive(MPI.COMM_WORLD))
mesh.Curve(order)
```



Naghdi shell undeformed geometry.

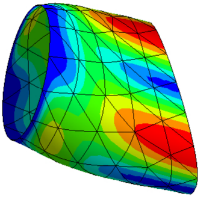
An NGsolve Example – Naghdi Shell

```
nsurf = specialcf.normal(3)
thickness = 0.1
Ptau = Id(3) - OuterProduct(nsurf,nsurf)
Ftau = grad(u).Trace() + Ptau
Ctautau = Ftau.trans * Ftau
Etautau = 0.5*(Ctautau - Ptau)
eps_beta = Sym(Ptau*grad(beta).Trace())
gradu = grad(u).Trace()
ngradu = gradu.trans*nsurf
gfn = nsurf
a = BilinearForm(fes, symmetric=True)
a += Variation( thickness*InnerProduct(Etautau,
    Etautau)*ds )
a += Variation( 0.5*thickness**3*InnerProduct(eps_beta
    -Sym(gradu.trans*grad(gfn)),eps_beta-Sym(gradu.
    trans*grad(gfn)))*ds )
a += Variation( thickness*(ngradu-beta)*(ngradu-beta)*
    ds )
factor = Parameter(0.0)
a += Variation( -thickness*factor*y*u[1]*ds )
```

PETSc SNES

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

```
factor.Set (1.5*(loadstep+1))
opts = {"snes_type": "newtonls",
"snes_max_it": 10,
"snes_monitor": "",
"ksp_monitor": "",
"pc_type": "lu"}
solver = NonLinearSolver(fes, a=a,solverParameters=opts)
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.

ngsPETSc – Firedrake

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

NGSolve User Meeting

UNIVERSITY OF OXFORD
Mathematical Institute

- 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 !