ngsPETSc: NGS 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

PDESoft, 1st of July 2024, Cambridge

UNIVERSITY OF OXFORD
Mathematical Institute

Oxford Mathematics

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** representation, 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.

Netgen/NGSolve

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 $H(curl)$ Nédélec 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 time-stepper capabilities of **TS**.
- Construct **DMPLEX** from **NETGEN** meshes.

PETSc KSP

An NGsolve Example – Poisson

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

mesh = Mesh(unit_square.GenerateMesh(maxh=0.2, comm=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 – 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**, **...**

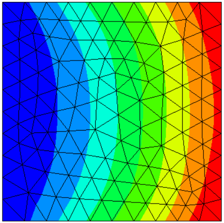
An NGsolve Example – Linear Elasticity

```
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.

```
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.

```
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.

```
from ngsPETSc.pc import *
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.pc import *
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)
dofs = BitArray(fes.ndof); dofs[:] = True
opts={"pc_type": "asm",
      "sub_ksp_type":"preonly", "sub_pc_type":"lu"}
blockjac = ASMPreconditioner(a.mat, dofs, blocks=
 blocks, solverParameters=opts)
```

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 = PETScPreconditioner(a.mat, vertexdofs, solverParameters={"pc_type": "hypre"})
pretwo = preCoarse.mat + blockjac
gfu.vec.data = CG(a.mat, rhs=f.vec, pre=pretwo, printrates=True)
```

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.

```
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)
```

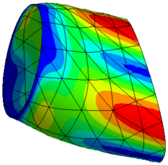
PETSc SNES

PETSc SNES

```
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)
```

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

```
    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

ngsPETSc – Firedrake

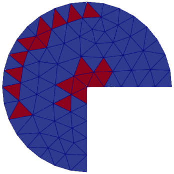
ngsPETSc provides new capabilities to **Firedrake** such as:

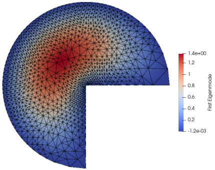
- Access to all Netgen generated linear meshes and high order meshes.
- Mesh refinement via splits, such as Alfeld and Powell-Sabin splits (even on curved geometries).
- Adaptive mesh refinement capabilities, that conform to the geometry.
- High order mesh hierarchies for multigrid solvers.

Mesh Refinement – Adaptive Mesh Refinement

UNIVERSITY OF OXFORD
Mathematical Institute

```
if comm.rank == 0:
    ngmsh = geo.GenerateMesh(maxh=0.2)
    labels=sum([ngmsh.GetBCIDs(label)
   for label in ["line","curve"]], [])
else:
    ngmsh=netgen.libngpy._meshing.Mesh
   (2)
    labels = None
msh = Mesh(ngmsh)
labels = comm.bcast(labels, root=0)
for i in range(max_iterations):
    lam, uh, V = Solve(msh,labels)
    mark = Mark(msh, uh, lam)
    msh = msh.Refine(mark)
    File("VTK/PacManAdp.pvd").write(uh,
   mark)
assert(abs(lam-exact)<1e-2)
```

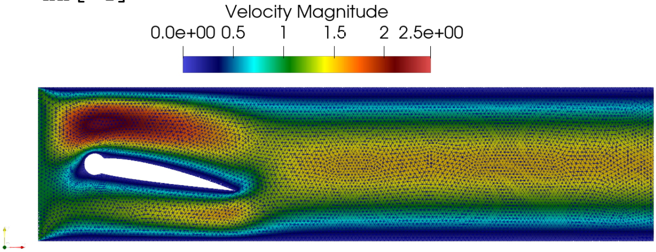




Multigrid on curved meshes

ngsPETSc allows us to create a hierarchy of curved meshes for multigrid solvers.

```
ngmesh = geo.GenerateMesh(maxh=0.1)
mesh = Mesh(ngmesh)
nh = MeshHierarchy(mesh, 2, netgen_flags={"degree":
    [1, 2, 3]})
mesh = nh[-1]
```

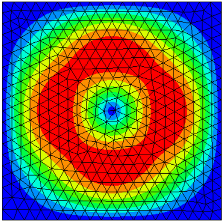


SLEPc EPS

SLEPc ESP

- We easily solve the eigenvalue problem associated to the Stokes formulation using ngsPETSc EigenSolver.

```python
from ngsPETSc import EigenSolver
opts={"eps_type":"arnoldi",
      "st_type":"sinvert",
      "pc_type": "lu",
      "pc_factor_mat_solver_type": "mumps"}
solver = EigenSolver((m, a), V, 10, solverParameters=opts)
solver.solve()
print ("Eigenvalues")
for i in range(10):
    print(solver.eigenValue(i))
eigenMode, _ = solver.eigenFunction(0)
```



First eigenfunctions of the Stokes eigenvalue problem

Conclusions

Future developments

- Integrate domain decomposition methods via **HPDDM**.
- Use **PETSc** as linear algebra backend in **NGSolve** to ensure cross-architecture compatibility and GPU acceleration.
- Wrap also **SLEPc PEP** to solve polynomial eigenvalue problems.

Thank You For Your Attention!