

ngsPETSc: NGS meets PETSc



Mathematical
Institute

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

A decorative geometric pattern consisting of white outlines of various polygons, including squares, rectangles, and parallelograms, arranged in a complex, overlapping structure that resembles a honeycomb or a crystalline lattice. The pattern is set against a dark blue background.

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 Hypr 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** 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.



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, $H(\text{div})$ Raviart-Thomas and Brezzi-Douglas-Marini elements, and $H(\text{curl})$ 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, **ngsTrefftz** for Trefftz 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

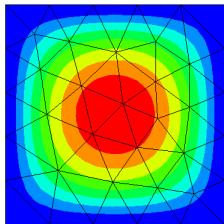
```
1  from ngsolve import *
2  import netgen.gui
3  import netgen.meshing as ngm
4  from mpi4py.MPI import COMM_WORLD
5
6  if COMM_WORLD.rank == 0:
7      mesh = Mesh(unit_square.GenerateMesh(maxh=0.2) .
8          Distribute(COMM_WORLD))
9  else:
10     mesh = Mesh(ngm.Mesh.Receive(COMM_WORLD))
11
12  fes = H1(mesh, order=3, dirichlet="left|right|top|
13     bottom")
14  u,v = fes.TnT()
15  a = BilinearForm(grad(u)*grad(v)*dx).Assemble()
16  f = LinearForm(fes)
17  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,
7                       solverParameters=opts)
8 gfu = solver.solve(f)
9 Draw(gfu,mesh, "solution")
```



Solution of Poisson problem
computed with MUMPS



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

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

```
1  opts = {'ksp_type': 'richardson',  
2         'ksp_richardson_scale': 1.0,  
3         'pc_type': 'jacobi'}  
4  solver = KrylovSolver(a,fes, solverParameters=opts)  
5  gfu = solver.solve(f)  
6  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**, ...



An NGSolve Example – Linear Elasticity

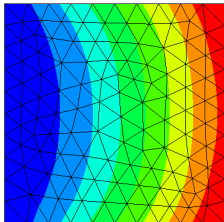
```
1  E, nu = 210, 0.2
2  mu  = E / 2 / (1+nu)
3  lam = E * nu / ((1+nu)*(1-2*nu))
4
5  def Stress(strain):
6      return 2*mu*strain + lam*Trace(strain)*Id(2)
7
8  fes = VectorH1(mesh, order=1, dirichlet="left")
9  u,v = fes.TnT()
10
11  a = BilinearForm(InnerProduct(Stress(Sym(Grad(u))),
12      Sym(Grad(v))))*dx)
13  a.Assemble()
14
15  force = CF( (0,1) )
16  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.

```
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 linear 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



- 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,
6   printrates=True)
7 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)



- 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,
6   printrates=True)
7 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.

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

```
1 blocks = VertexPatchBlocks(mesh, fes)
2 blockjac = a.mat.CreateBlockSmoother(blocks)
3 gfu.vec.data = CG(a.mat, rhs=f.vec, pre=blockjac,
4   printrates=True)
5 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.

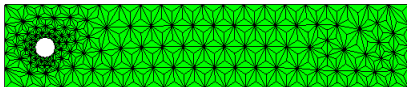
```
1  from ngsPETSc import pc
2  smoother = Preconditioner(aDG, "PETScPC", pc_type="
    sor")
3  transform = fes.ConvertL2Operator(fesDG)
4  preDG = transform @ pre.mat @ transform.T +
    smoother.mat
5  gfuDG = GridFunction(fesDG)
6  gfuDG.vec.data = CG(aDG.mat, rhs=fDG.vec, pre=preDG
    , printrates=True)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

Saddle Point Problems

An NGSolve Example – Stokes flow



```
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"))
```



Fieldsplit Schur preconditioner – Mass matrix

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:

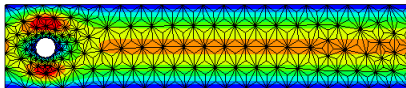
$$\begin{bmatrix} \hat{A}^{-1} & 0 \\ 0 & -\nu \hat{M}^{-1} \end{bmatrix}$$

where M is the mass matrix.

PETSc PC – NGSolve Fieldsplit



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

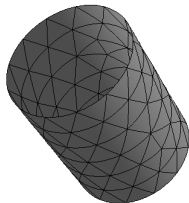


PETSc SNES

An NGSolve Example – Naghdi Shell



```
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

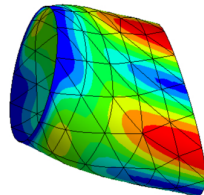


```
1 nsurf = specialcf.normal(3)
2 thickness = 0.1
3 Ptau = Id(3) - OuterProduct(nsurf,nsurf)
4 Ftau = grad(u).Trace() + Ptau
5 Ctautau = Ftau.trans * Ftau
6 Etautau = 0.5*(Ctautau - Ptau)
7 eps_beta = Sym(Ptau*grad(beta).Trace())
8 gradu = grad(u).Trace()
9 ngradu = gradu.trans*nsurf
10 gfn = nsurf
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
8 ,solverParameters=opts)
9 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 $H(\text{div})$, $H(\text{curl})$, H^1 and H^2 .
- ▶ Provides access to **PETSc** linear solvers and non-linear solvers.

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.

SLEP_c EPS

An NGSolve Example – Mass conserving scheme

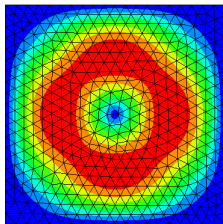


```
1 V1 = HDiv(mesh,order=order,dirichlet="rect",RT=True)
2 V2 = HCurlDiv(mesh, order=order, GGBubbles=True)
3 Q = L2(mesh, order=order, lowest_order_wb=el_int)
4 W = L2(mesh, order=order)
5 N = ng.FESpace("number", mesh)
6 V = V1 * V2 * Q * N * W
7 u, sigma, p, lam, w = V.TrialFunction()
8 v, tau, q, mu, r = V.TestFunction()
9 n = ng.specialcf.normal(mesh.dim)
10 h = ng.specialcf.mesh_size
11 a = ng.BilinearForm(V,eliminate_internal=el_int)
12 a += 1/(2*nu)*InnerProduct(dev(sigma),dev(tau))*dx
13 a += ((div(sigma) * v + div(tau) * u)) * dx
14 a += -(((sigma * n) * n) * (v * n) + ((tau * n) * n) *
15         (u * n)) * dx(element_boundary=True)
16 a += (div(u) * q + div(v) * p) * dx
17 a += (lam * q + mu * p) * dx
18 a += (w * skw(tau) + skw(sigma) * r) * dx
19 m = ng.BilinearForm(V)
20 m += -1.*InnerProduct(u,v)*dx
```




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

```
1 from ngsPETSc import EigenSolver
2 opts={"eps_type":"arnoldi",
3       "st_type":"sinvert",
4       "pc_type": "lu",
5       "pc_factor_mat_solver_type": "
  mumps"}
6 solver = EigenSolver((m, a), V, 10,
  solverParameters=opts)
7 solver.solve()
8 print ("Eigenvalues")
9 for i in range(10):
10     print(solver.eigenValue(i))
11 eigenMode, _ = solver.eigenFunction
    (0)
```



First eigenfunctions of the Stokes eigenvalue problem

Conclusions

- ▶ We plan to extend the interface to include time-stepping capabilities from **PETSc TS**.
- ▶ We plan to experiment with **HPDDM**.
- ▶ We plan to use **PETSc** as linear algebra backend in **NGSolve** to ensure cross-architecture compatibility and GPU acceleration.

- 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 !