ngsPETSc

Release 0.0.1

Umberto Zerbinati

May 14, 2024

CONTENTS:

CHAPTER

ONE

NGSPETSC

ngsPETSc is a PETSc interface for NGSolve.

CHAPTER

TWO

INSTALLATION

To install ngsPETSc you need to clone the GitHub repository, and then you can install it using pip.

```
git clone https://github.com/UZerbinati/ngsPETSc.git
cd ngsPETSc
pip install .
```

You can also build PETSc, SLEPc and NGSolve from source and then install ngsPETSc. First we install all the needed package using apt and pip or an equivalent package manager.

```
apt-get update
apt-get -y install git build-essential cmake python3 python3-distutils python3-tk␣
↪libpython3-dev libxmu-dev tk-dev tcl-dev g++ libglu1-mesa-dev liblapacke-dev libblas-
↪dev liblapack-dev
pip install numpy cython pytest pytest-mpi
```

We now install PETSc from scratch in a home installation folder, with OpenMPI, HYPRE, Metis, MUMPS, SuprLU, Scalapack and eigen.

```
git clone https://gitlab.com/petsc/petsc.git
cd petsc
python configure --download-chaco \
--download-cmake \
--download-eigen \
--download-openmpi \
--download-hypre \
--download-metis \
--download-parmetis \
--download-ml \
--download-mumps \
--download-scalapack \
--download-superlu_dist \
--with-c2html=0 \
--with-cxx-dialect=C++11 \
--with-debugging=0 \
--download-fblaslapack=1 \
--with-fortran-bindings=0 \
--with-shared-libraries=1 \
--with-petsc4py=1 \
```

To build PETSc you need to run the Makefile as suggested at the end of the configuration script. We now need to set in the `.bashrc` (for MacOS user `.bash_profile`) file the `PETSC_DIR`, `PETSC_ARCH` system variable as they appear

when we finish build PETSc. You also need to add to your PYTHONPATH the PYTHONPATH appear when we finished building PETSc. We also suggest adding the following line to you .bashrc the following lines:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PETSC_DIR/$PETSC_ARCH/lib
export PATH=$PATH:$PETSC_DIR/$PETSC_ARCH/bin
```

We now install SLEPc from source once again in the home installation folder

```
git clone https://gitlab.com/slepc/slepc.git
cd slepc
python configure --download-blopex --with-slepc4py=1
```

To build SLEPc you need to run the Makefile as suggested at the end of the configuration script. We now need to set in the .bashrc (for MacOS user .bash_profile) file the SLEPC_DIR system variable as they appear when we finish build PETSc. You also need to add to your PYTHONPATH the PYTHONPATH appear when we finished building SLEPc. We now build mpi4py form source (once again in our home installation folder) in order to have a mpi4py installation that uses PETSc's local MPI installation.

```
git clone https://github.com/mpi4py/mpi4py.git
cd mpi4py
pip install .
```

Now we are left building NGSolve from sources, once again in the home installation directory.

```
export BASEDIR=$PWD/ngsuite
mkdir -p $BASEDIR
cd $BASEDIR
git clone https://github.com/NGSolve/ngsolve.git ngsolve-src
cd $BASEDIR/ngsolve-src
git submodule update --init --recursive
mkdir $BASEDIR/ngsolve-build
mkdir $BASEDIR/ngsolve-install
cd $BASEDIR/ngsolve-build
cmake -DCMAKE_INSTALL_PREFIX=${BASEDIR}/ngsolve-install ${BASEDIR}/ngsolve-src -DUSE_
↪MPI=ON
make
make install
```

You should add to your .bashrc the BASEDIR system variable:

```
echo "export $BASEDIR=${BASEDIR}" >> ~/.bashrc
```

We suggest you adding following lines to your .bashrc:

```
export NETGENDIR="${BASEDIR}/ngsolve-install/bin"
export PATH=$NETGENDIR:$PATH
export PYTHONPATH=$PYTHONPATH:$NETGENDIR/../`python3 -c "from distutils.sysconfig import␣
↪get_python_lib; print(get_python_lib(1,0,''))"`
```

We are now fianlly ready to install ngsPETSc:

```
git clone https://github.com/UZerbinati/ngsPETSc.git
cd ngsPETSc
NGSPETSC_NO_INSTALL_REQUIRED=ON pip install .
```

CHAPTER

THREE

AUTHORS

Stefano Zampini, Umberto Zerbinati

CHAPTER

FOUR

LICENSE

The package is released under the MIT License.

CHAPTER

FIVE

API

CHAPTER

SIX

HOW TO USE NGSPETSC

This page was generated from notebooks/Vectors and Matrices.ipynb.

6.1 PETSc Vec and PETSc Mat

This tutorial will be focused on how to use the PETSc `KSP` class to solve the linear systems that are obtained from a finite element discretization of a partial differential equation (PDE). In particular, we will show how to use the `VectorMapping` class to map PETSc `Vec` to NGSolve vectors and vice versa and the `Matrix` class to create a PETSc `Mat` from an NGSolve `BilinearForm`.

We begin initializing the cluster to the test parallel implementation in a Jupyter notebook, you can do this also using the command line, i.e. `ipcluster start -engines=MPI -n 4`.

[1]:
```
from ipyparallel import Cluster
c = await Cluster().start_and_connect(n=1, activate=True)
```

```
Starting 1 engines with <class 'ipyparallel.cluster.launcher.LocalEngineSetLauncher'>
```

```
  0%|          | 0/1 [00:00<?, ?engine/s]
```

Let's test if the cluster has been initialized correctly by checking the size of the `COMM_WORLD`.

[2]:
```
%%px
from mpi4py.MPI import COMM_WORLD
COMM_WORLD.Get_size()
```

Out[0:1]: 1

First we need to construct the distributed mesh that will be used to define the finite element space that will be used to discretize the PDE here considered.

[3]:
```
%%px
from ngsolve import Mesh
from netgen.geom2d import unit_square
import netgen.meshing as ngm

if COMM_WORLD.rank == 0:
    mesh = Mesh(unit_square.GenerateMesh(maxh=0.2).Distribute(COMM_WORLD))
else:
    mesh = Mesh(ngm.Mesh.Receive(COMM_WORLD))
```

We now proceed constructing a linear polynomial finite element space, with H^1 conformity, and discretize the mass matrix that represent the L^2 scalar product in the discrete context. We create a mass matrix to initialize a NGSolve vector corresponding a `GridFunction` defined on the finite element space here considered.

[4]:
```
%%px
from ngsolve import H1, BilinearForm, dx
fes = H1(mesh, order=1, dirichlet="left|right|top|bottom")
u,v = fes.TnT()
m = BilinearForm(u*v*dx).Assemble()
M = m.mat
ngsVec = M.CreateColVector()
```

We are now ready to create a `VectorMapping` that we will first use to construct PETSc `Vec` corresponding to the `ngsVec` just initialized. The only information that the `VectorMapping` class needs is the finite element space corresponding to the vector associated to the `GridFunction` we aim to map, this because the NGSolve `FESpace` class contains information about the way the degrees of freedom are distributed and which degrees of freedom are not constrained by the boundary conditions.

[5]:
```
%%px
from ngsPETSc import VectorMapping
Map = VectorMapping(fes)
petscVec = Map.petscVec(ngsVec)
print("Vector type is {} and it has size {}.".format(petscVec.type,petscVec.size))
```

```
[stdout:0] Vector type is seq and it has size 17.
```

We now use the `Matrix` class to create a PETSc `Mat` from a NGSolve `BilinearForm`. Once the `Matrix` class has been set up, it is possible to access the corresponding PETSc `Mat` object as `Matrix().mat`. By default, if the communicator world is larger than one `mat` is initialized as a PETSc `mpiaij` which is the default sparse parallel matrix in PETSc, while if the communicator world is one than `mat` is initialized as a PETSc `seqaij` which is the default serial matrix in PETSc. We can also spy inside the matrix using the `Matrix().view()` method.

[7]:
```
%%px
from ngsPETSc import Matrix
M = Matrix(m.mat, fes)
print("Matrix type is {} and it has size {}.".format(M.mat.type,M.mat.size))
M.view()
```

```
[stdout:0] Matrix type is seqaij and it has size (17, 17).
Mat Object: 1 MPI process
  type: seqaij
row 0: (0, 0.0224274)  (1, 0.00348275)  (10, 0.00389906)  (12, 0.00344571)
row 1: (0, 0.00348275)  (1, 0.0198054)  (2, 0.00329403)  (11, 0.00318985)  (12, 0.
↪00335232)
row 2: (1, 0.00329403)  (2, 0.0213869)  (3, 0.00364805)  (11, 0.00320442)
row 3: (2, 0.00364805)  (3, 0.0187914)  (4, 0.00276012)  (11, 0.0029006)  (15, 0.
↪00252874)
row 4: (3, 0.00276012)  (4, 0.0161198)  (5, 0.00273449)  (13, 0.0025502)  (15, 0.
↪00239586)
row 5: (4, 0.00273449)  (5, 0.0170459)  (6, 0.00262495)  (13, 0.00256258)
row 6: (5, 0.00262495)  (6, 0.0125646)  (7, 0.00238756)  (13, 0.00238142)
row 7: (6, 0.00238756)  (7, 0.0176799)  (8, 0.00325374)  (13, 0.00285307)  (16, 0.
↪00338652)
row 8: (7, 0.00325374)  (8, 0.0200592)  (9, 0.00312461)  (14, 0.00350041)  (16, 0.
```

(continues on next page)

(continued from previous page)

```
↪00368234)
row 9: (8, 0.00312461)  (9, 0.0189321)  (14, 0.00362765)
row 10: (0, 0.00389906)  (10, 0.0211063)  (12, 0.00340267)  (14, 0.00383436)
row 11: (1, 0.00318985)  (2, 0.00320442)  (3, 0.0029006)  (11, 0.0187766)  (12, 0.
↪0034216)  (15, 0.0027623)  (16, 0.00329787)
row 12: (0, 0.00344571)  (1, 0.00335232)  (10, 0.00340267)  (11, 0.0034216)  (12, 0.
↪0210307)  (14, 0.00364803)  (16, 0.00376036)
row 13: (4, 0.0025502)  (5, 0.00256258)  (6, 0.00238142)  (7, 0.00285307)  (13, 0.
↪0159034)  (15, 0.00253607)  (16, 0.0030201)
row 14: (8, 0.00350041)  (9, 0.00362765)  (10, 0.00383436)  (12, 0.00364803)  (14, 0.
↪0226675)  (16, 0.00387176)
row 15: (3, 0.00252874)  (4, 0.00239586)  (11, 0.0027623)  (13, 0.00253607)  (15, 0.
↪0130458)  (16, 0.00282287)
row 16: (7, 0.00338652)  (8, 0.00368234)  (11, 0.00329787)  (12, 0.00376036)  (13, 0.
↪0030201)  (14, 0.00387176)  (15, 0.00282287)  (16, 0.0238418)
```

There are other matrices format that are wrapped some of which are device dependent, to mention a few: - `dense`, store and operate on the matrix in dense format, - `cusparse`, store and operate on the matrix on NVIDIA GPU device in CUDA sparse format, - `aijmkl`, store and operate on the matrix in Intel MKL format.

[9]:
```
%%px
M = Matrix(m.mat, fes, matType="dense")
```

6.1.1 Example (Precondition Inverse Iteration)

We here implement the Precondition INVerse ITeration (PINVIT) developed by Knyazef and Neymeyr, more detail here, using PETSc. In particular, we will use the PINVIT scheme to compute the eigenvalue of the Laplacian, i.e. we are looking for $\lambda \in \mathbb{R}$ such that it exits $u \in H_0^1(\Omega)$ that verifies following equation for any $v \in H_0^1(\Omega)$

$$\int_\Omega \nabla u \cdot \nabla v \, d\vec{x} = \lambda \int_\Omega uv \, d\vec{x}.$$

We solve this specific problem by looking for the eigenvalue of the generalised eigenproblem $A\vec{u}_h = \lambda M \vec{u}_h$ where A and M are the finite element discretisation respectively of the stiffness matrix corresponding to the Laplacian and the mass matrix corresponding to the L^2 inner product. We begin constructing the finite element discretisation for A and M.

[10]:
```
%%px
from ngsolve import grad, Preconditioner, GridFunction
a = BilinearForm(fes)
a += grad(u)*grad(v)*dx
a.Assemble()
u = GridFunction(fes)
```

The heart of the PINVIT scheme there is an iteration similar idea to the Rayleigh quotient iteration for a generalised eigenvalue problem, more detail can be found in Nick Trefethen's Numerical Linear Algebra, Lecture 27:

$$\vec{u}_h^{(n+1)} = \omega_1^{(n)} \vec{u}_h^{(n)} + \omega_2^{(n)} \vec{\omega}_h^{(n)}, \qquad \vec{\omega}_h^{(n)} = P^{-1}(A\vec{u}_h^{(n)} - \rho_n M \vec{u}_h^{(n)}),$$

where P^{-1} is an approximate inverse of the stifness matrix A and ρ_n is the Rayleigh quotient corresponding to $\vec{u}_h^{(n)}$,

i.e.

$$\rho_n = \frac{(\vec{u}_h^{(n)}, A\vec{u}_h^{(n)})}{(\vec{u}_h^{(n)}, M\vec{u}_h^{(n)})}.$$

Instrumental in order to obtain a converged PINVIT scheme is our choice of α_n, but we will postpone this discuss and first implement the previous itration for a fixed choice of $\omega_i^{(n)}$.

[11]:
```
%%px
def stepChoice(Asc,Msc,w,u0):
    return (0.5,0.5)
```

We begin constructing a PETSc `Mat` object corresponding to A and M using the ngsPETSc `Matrix` class. We then construct a `VectorMapping` to convert NGSolve `GridFunction` to PETSc `Vec`.

[12]:
```
%%px
A = Matrix(a.mat, fes)
M = Matrix(m.mat, fes)
Map = VectorMapping(fes)
```

We then construct a PETSc `PC` object used to create an approximate inverse of A, in particular we will be interested in using a preconditioner build using HYPRE.

[13]:
```
%%px
from petsc4py import PETSc
pc = PETSc.PC()
pc.create(PETSc.COMM_WORLD)
pc.setOperators(A.mat)
pc.setType(PETSc.PC.Type.HYPRE)
pc.setUp()
```

We now implement the iteration itself, starting from a PETSc `Vec` that we create from a PETSc `Mat` to be sure it has the correct size, and that we then set to have random entries.

[14]:
```
%%px
from math import pi
itMax = 10
u0 = A.mat.createVecLeft()
w = A.mat.createVecLeft()
u0.setRandom()
for it in range(itMax):
        Au0 = u0.duplicate(); A.mat.mult(u0,Au0)
        Mu0 = u0.duplicate(); M.mat.mult(u0,Mu0)
        rho = Au0.dot(u0)/Mu0.dot(u0)
        print("[{}] Eigenvalue estimate: {}".format(it,rho/(pi**2)))
        u = Au0+rho*Mu0
        pc.apply(u,w)
        alpha = stepChoice(A.mat,M.mat,w,u0)
        u0 = alpha[0]*u0+alpha[1]*w
```

```
[stdout:0] [0] Eigenvalue estimate: 6.4389641604089185
[1] Eigenvalue estimate: 3.343928625687638
[2] Eigenvalue estimate: 2.6416489541644186
[3] Eigenvalue estimate: 2.3821542621098093
[4] Eigenvalue estimate: 2.2665070692423788
```

(continues on next page)

(continued from previous page)

```
[5] Eigenvalue estimate: 2.210409247274843
[6] Eigenvalue estimate: 2.1819357667646018
[7] Eigenvalue estimate: 2.167055475874936
[8] Eigenvalue estimate: 2.1590946930855504
[9] Eigenvalue estimate: 2.1547369780738284
```

We now need to discuss how to choose the step size ω_i and we do this by solving the optimization problem,

$$\vec{u}_h^{(n+1)} = \underset{\vec{v} \in <\vec{u}_h^n, \vec{\omega}_h^{(n)}>}{arg\ min} \frac{(\vec{u}_h^{(n+1)}, A\vec{u}_h^{(n+1)})}{(\vec{u}_h^{(n+1)}, M\vec{u}_h^{(n+1)})}$$

and we do solving a small generalised eigenvalue problem, i.e.

$$\begin{bmatrix} \vec{u}_h^{(n)} \cdot A\vec{u}_h^{(n)} & \vec{u_h}^{(n)} \cdot A\vec{\omega}_h^{(n)} \\ \vec{\omega}_h^{(n)} \cdot A\vec{u}_h^{(n)} & \vec{\omega}_h^{(n)} \cdot A\vec{\omega}_h^{(n)} \end{bmatrix} = \omega \begin{bmatrix} \vec{u}_h^{(n)} \cdot M\vec{u}_h^{(n)} & \vec{u_h}^{(n)} \cdot M\vec{\omega}_h^{(n)} \\ \vec{\omega}_h^{(n)} \cdot M\vec{u}_h^{(n)} & \vec{\omega}_h^{(n)} \cdot M\vec{\omega}_h^{(n)} \end{bmatrix}.$$

[15]:
```
%%px
import numpy as np
from scipy.linalg import eigh
def stepChoice(Asc,Msc,w,u0):
    Au0 = u0.duplicate(); Asc.mult(u0,Au0)
    Mu0 = u0.duplicate(); Msc.mult(u0,Mu0)
    Aw = w.duplicate(); Asc.mult(w,Aw)
    Mw = w.duplicate(); Msc.mult(w,Mw)
    smallA = np.array([[u0.dot(Au0),u0.dot(Aw)],[w.dot(Au0),w.dot(Aw)]])
    smallM = np.array([[u0.dot(Mu0),u0.dot(Mw)],[w.dot(Mu0),w.dot(Mw)]])
    _, evec = eigh(a=smallA, b=smallM)
    return (float(evec[0,0]),float(evec[1,0]))

itMax = 10
u0 = A.mat.createVecLeft()
w = A.mat.createVecLeft()
u0.setRandom()
for it in range(itMax):
        Au0 = u0.duplicate(); A.mat.mult(u0,Au0)
        Mu0 = u0.duplicate(); M.mat.mult(u0,Mu0)
        rho = Au0.dot(u0)/Mu0.dot(u0)
        print("[{}] Eigenvalue estimate: {}".format(it,rho/(pi**2)))
        u = Au0+rho*Mu0
        pc.apply(u,w)
        alpha = stepChoice(A.mat,M.mat,w,u0)
        u0 = alpha[0]*u0+alpha[1]*w
```

```
[stdout:0] [0] Eigenvalue estimate: 6.4389641604089185
[1] Eigenvalue estimate: 2.182148561544114
[2] Eigenvalue estimate: 2.149490978038022
[3] Eigenvalue estimate: 2.1482074870710552
[4] Eigenvalue estimate: 2.148165460157942
[5] Eigenvalue estimate: 2.1481654570280604
[6] Eigenvalue estimate: 2.148165457028059
```

(continues on next page)

(continued from previous page)

```
[7] Eigenvalue estimate: 2.1481654570280577
[8] Eigenvalue estimate: 2.148165457028059
[9] Eigenvalue estimate: 2.148165457028058
```

This page was generated from notebooks/Meshes.ipynb.

6.2 PETSc DMPlex

In this tutorial we will have an in-depth look in to the way we transform NGSolve/Netgen `Mesh` to a PETSc DMPlex. In particular we will show to create a PETSc `DMPlex` from an NGSolve/Netgen and vice-versa, using the `MeshMapping` class. We will also show how to use PETSc `DMPlexTransform` to construct purely quad mesh and Alfeld split mesh.

[1]:
```
from ipyparallel import Cluster
c = await Cluster().start_and_connect(n=1, activate=True)
```

```
Starting 1 engines with <class 'ipyparallel.cluster.launcher.LocalEngineSetLauncher'>
```

```
  0%|          | 0/1 [00:00<?, ?engine/s]
```

Let's test if the cluster has been initialized correctly by checking the size of the `COMM_WORLD`.

[2]:
```
%%px
from mpi4py.MPI import COMM_WORLD
COMM_WORLD.Get_size()
```

Out[0:1]: 1

First we need to construct the distributed mesh that will be interested in dealing with in our map to a PETSc `DMPlex`.

[3]:
```
%%px
from ngsolve import Mesh
from netgen.geom2d import unit_square

if COMM_WORLD.rank == 0:
    mesh = Mesh(unit_square.GenerateMesh(maxh=0.2).Distribute(COMM_WORLD))
else:
    mesh = Mesh(ngm.Mesh.Receive(COMM_WORLD))
```

We can now convert this matrix in a PETSc `DMPlex`, to do this we will initialize a `MeshMapping` class and then access the mapped PETSc `DMPlex` as the attribute `petscPlex`.

[4]:
```
%%px
from ngsPETSc import MeshMapping
Map = MeshMapping(mesh)
Map.petscPlex.view()
```

```
[stdout:0] DM Object: Default 1 MPI process
  type: plex
Default in 2 dimensions:
  Number of 0-cells per rank: 37
```

(continues on next page)

(continued from previous page)

```
  Number of 1-cells per rank: 88
  Number of 2-cells per rank: 52
Labels:
  celltype: 3 strata with value/size (0 (37), 3 (52), 1 (88))
  depth: 3 strata with value/size (0 (37), 1 (88), 2 (52))
  Face Sets: 4 strata with value/size (1 (5), 2 (5), 3 (5), 4 (5))
```

We can use any PETSc `DMPlex` function that is wrapped in petsc4py on the `Map.petscPlex` object. For example, we can apply any PETSc `DMPlexTransform`. We will now apply different PETSc `DMPlexTransform` and check using the `view` method that we have the mesh was transformed correctly. We begin with the PETSc `DMPlexTransformType.REFINEREGULAR` which will create split a triangle in four subs triangles connecting the middle points of each vertex of the triangle to create a new triangle in the center.

[5]:
```
%%px
from petsc4py import PETSc
tr = PETSc.DMPlexTransform().create(comm=PETSc.COMM_WORLD)
tr.setType(PETSc.DMPlexTransformType.REFINEREGULAR)
tr.setDM(Map.petscPlex)
tr.setUp()
newplex = tr.apply(Map.petscPlex)
newplex.view()
```

```
[stdout:0] DM Object: 1 MPI process
  type: plex
DM_0x55baeedcda70_1 in 2 dimensions:
  Number of 0-cells per rank: 125
  Number of 1-cells per rank: 332
  Number of 2-cells per rank: 208
Labels:
  celltype: 3 strata with value/size (1 (332), 3 (208), 0 (125))
  depth: 3 strata with value/size (0 (125), 1 (332), 2 (208))
  Face Sets: 4 strata with value/size (1 (15), 2 (15), 3 (15), 4 (15))
```

We can easily verify that the number of `2-cells` elements, i.e. triangles in the mesh has quadrupled. We can also create a new `MeshMapping` class to convert the new PETSc `DMPlex` into a Netgen `Mesh` and visualize it.

[6]:
```
%%px
from ngsolve import Mesh
Map = MeshMapping(newplex)
from ngsolve.webgui import Draw
Draw(Mesh(Map.ngMesh))
```

```
[output:0]
```

```
WebGuiWidget(layout=Layout(height='50vh', width='100%'), value={'gui_settings': {},
↪'ngsolve_version': '6.2.23...
```

```
Out[0:5]: BaseWebGuiScene
```

We can experiment also with other PETSc `DMPlexTransformation` for example the `REFINETOBOX` transformation which will split each triangle in the mesh into three quadrilateral by joining the midpoints of each edge. This will allow to obtain a purely quadrilateral mesh from a Netgen mesh.

[7]:
```
%%px
tr = PETSc.DMPlexTransform().create(comm=PETSc.COMM_WORLD)
tr.setType(PETSc.DMPlexTransformType.REFINETOBOX)
tr.setDM(Map.petscPlex)
tr.setUp()
newplex = tr.apply(Map.petscPlex)
Map = MeshMapping(newplex)
Draw(Mesh(Map.ngMesh))
```

```
[output:0]
```

```
WebGuiWidget(layout=Layout(height='50vh', width='100%'), value={'gui_settings': {},
↪'ngsolve_version': '6.2.23...
```

Out[0:6]: BaseWebGuiScene

6.2.1 Example (Alfeld Splittings and Scott-Vogelious)

In this example we would like to show that the finite element pair $P^2 - P^1_{disc}$ known as the low-order Scott-Vogelious pair, which is known to verify the Brezzi-Babuska condition for the Stokes problem on Alfeld split mesh, can be easily implemented in NGSolve using a `PETSc DMPlexTransformation`. First we construct a mesh and use the `MeshMapping` class to obtain an PETSc `DMPlex` that we proceed to split to obtain an Alfeld refinement.

[8]:
```
%%px
from netgen.geom2d import SplineGeometry
if COMM_WORLD.rank == 0:
    geo = SplineGeometry()
    geo.AddRectangle( (0, 0), (2, 0.41), bcs = ("wall", "outlet", "wall", "inlet"))
    geo.AddCircle ( (0.2, 0.2), r=0.05, leftdomain=0, rightdomain=1, bc="cyl")
    mesh = Mesh( geo.GenerateMesh(maxh=0.05))
    mesh.Curve(3)
else:
    mesh = Mesh(ngm.Mesh.Receive(COMM_WORLD))
Map = MeshMapping(mesh)
tr = PETSc.DMPlexTransform().create(comm=PETSc.COMM_WORLD)
tr.setType(PETSc.DMPlexTransformType.REFINEALFELD)
tr.setDM(Map.petscPlex)
tr.setUp()
newplex = tr.apply(Map.petscPlex)
mesh = Mesh(MeshMapping(newplex).ngMesh)
Draw(mesh)
```

```
[output:0]
```

```
WebGuiWidget(layout=Layout(height='50vh', width='100%'), value={'gui_settings': {},
↪'ngsolve_version': '6.2.23...
```

Out[0:7]: BaseWebGuiScene

We now proceed constructing the finite element space we are interested in and distressing the Stokes equation in variational form, i.e. find $(\vec{u}_h, p_h) \in [P^2(\mathcal{T}_h)]^2 \times P^1_{disc}(\mathcal{T}_h)$ such that for any $(\vec{v}_h, q_h) \in [P^2(\mathcal{T}_h)]^2 \times P^1_{disc}(\mathcal{T}_h)$ the following equations hold,

$$(\nabla \vec{u}_h, \nabla \vec{v}_h) - (\nabla \cdot \vec{v}_h, p_h) = (\vec{f}, \vec{v}_h)$$
$$(\nabla \cdot \vec{u}_h, 1_h) = 0$$

[9]:
```
%%px
from ngsolve import VectorH1, L2, H1, BilinearForm, InnerProduct, GridFunction
from ngsolve import grad, div, x,y, dx, CoefficientFunction, Norm, SetVisualization, TRIG
V = VectorH1(mesh, order=2,dirichlet=[4,1,3,5,6,7,8])
Q = L2(mesh, order=1)
X = V*Q

u,p = X.TrialFunction()
v,q = X.TestFunction()

a = BilinearForm(X)
a += (InnerProduct(grad(u),grad(v))+div(u)*q-div(v)*p)*dx
a.Assemble()

gfu = GridFunction(X)
uin = CoefficientFunction( (1.5*4*y*(0.41-y)/(0.41*0.41), 0) )
gfu.components[0].Set(uin, definedon=mesh.Boundaries([3]))

res = gfu.vec.CreateVector()
res.data = -a.mat * gfu.vec
inv = a.mat.Inverse(freedofs=X.FreeDofs(), inverse="umfpack")
gfu.vec.data += inv * res
Draw(Norm(gfu.components[0]), mesh, "|vel|")
SetVisualization(max=2)
```

```
[stdout:0] warning: Boundaries( [int list] ) is deprecated, pls generate Region
```

```
[output:0]
```

```
WebGuiWidget(layout=Layout(height='50vh', width='100%'), value={'gui_settings': {},
↪'ngsolve_version': '6.2.23...
```

This page was generated from notebooks/Krylov Solver and Preconditioners.ipynb.

6.3 PETSc KSP and PETSc PC

In this tutorial we will have an in-depth look in to the way we can use a PETSc KSP object to solve a linear system arising from the discretization of a linear partial differential equation. In particular, we will focus our attention the usual Poisson problem in variational form, i.e. find $u_h \in P^k(\mathcal{T}_h)$ such for any $v_h \in P^k(\mathcal{T}_h)$ the following equation is verified,

$$(\nabla u_h, \nabla v_h) = (f, v_h), \quad \text{for a certain data } f \in [P^k(\mathcal{T}_h)]^*.$$

[1]:
```
from ipyparallel import Cluster
c = await Cluster().start_and_connect(n=1, activate=True)
```

```
Starting 1 engines with <class 'ipyparallel.cluster.launcher.LocalEngineSetLauncher'>
```

```
  0%|          | 0/1 [00:00<?, ?engine/s]
```

Let's test if the cluster has been initialized correctly by checking the size of the `COMM_WORLD`-

[2]:
```
%%px
from mpi4py.MPI import COMM_WORLD
COMM_WORLD.Get_size()
```

```
Out[0:1]: 1
```

First we need to construct the distributed mesh that will be interested in dealing with, then define the `BilinearForm` corresponding to the Poisson problem and the load vector `f` we are interested in solving for, which is a NGSolve `LinearForm` object.

[3]:
```
%%px
from ngsolve import Mesh, BilinearForm, LinearForm, H1
from ngsolve import x, y, dx, grad
from netgen.geom2d import unit_square

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
```

We now initialize a new `KrylovSolver` which wraps a PETSc `KSP` object and can be used to solve the linear system $A\vec{u}_h = \vec{f}_h$, obtained from the above discretization. In particular, it is possible to control the setup of the PETSc `KSP` solver using the `solverParameters` dictionary which is equivalent to passing PETSc command line options. More detail on available options and solvers can be found in the PETSc KSP Manual. Once the solver has been initialized it is possible to obtain a solution of the linear system in terms of `GridFunction` using the `solve` method.

[4]:
```
%%px
from ngsPETSc import KrylovSolver
from ngsolve.webgui import  Draw
solver = KrylovSolver(a,fes, solverParameters={'ksp_type': 'cg', 'pc_type': 'lu',
                                'pc_factor_mat_solver_type': 'mumps'})
gfu = solver.solve(f)
Draw(gfu,mesh, "solution")
solver.ksp.view()
```

```
[output:0]
```

```
WebGuiWidget(layout=Layout(height='500px', width='100%'), value={'gui_settings': {},
↪'ngsolve_version': '6.2.2...
```

```
[stdout:0] KSP Object: 1 MPI process
  type: cg
  maximum iterations=10000, initial guess is zero
  tolerances:  relative=1e-05, absolute=1e-50, divergence=10000.
  left preconditioning
  using PRECONDITIONED norm type for convergence test
```

(continues on next page)

(continued from previous page)

```
PC Object: 1 MPI process
  type: lu
    out-of-place factorization
    tolerance for zero pivot 2.22045e-14
    matrix ordering: external
    factor fill ratio given 0., needed 0.      Factored matrix follows:
        Mat Object: 1 MPI process
          type: mumps
          rows=205, cols=205
          package used to perform factorization: mumps
          total: nonzeros=7603, allocated nonzeros=7603
            MUMPS run parameters:
              Use -ksp_view ::ascii_info_detail to display information for all processes
              RINFOG(1) (global estimated flops for the elimination after analysis):␣
↪153913.
              RINFOG(2) (global estimated flops for the assembly after factorization):␣
↪1916.
              RINFOG(3) (global estimated flops for the elimination after factorization):
↪ 153913.
              (RINFOG(12) RINFOG(13))*2^INFOG(34) (determinant): (0.,0.)*(2^0)          ␣
↪   INFOG(3) (estimated real workspace for factors on all processors after analysis):␣
↪7603
              INFOG(4) (estimated integer workspace for factors on all processors after␣
↪analysis): 998
              INFOG(5) (estimated maximum front size in the complete tree): 35
              INFOG(6) (number of nodes in the complete tree): 14
              INFOG(7) (ordering option effectively used after analysis): 2
              INFOG(8) (structural symmetry in percent of the permuted matrix after␣
↪analysis): 100
              INFOG(9) (total real/complex workspace to store the matrix factors after␣
↪factorization): 7603
              INFOG(10) (total integer space store the matrix factors after␣
↪factorization): 998
              INFOG(11) (order of largest frontal matrix after factorization): 35
              INFOG(12) (number of off-diagonal pivots): 0
              INFOG(13) (number of delayed pivots after factorization): 0
              INFOG(14) (number of memory compress after factorization): 0
              INFOG(15) (number of steps of iterative refinement after solution): 0
              INFOG(16) (estimated size (in MB) of all MUMPS internal data for␣
↪factorization after analysis: value on the most memory consuming processor): 0
              INFOG(17) (estimated size of all MUMPS internal data for factorization␣
↪after analysis: sum over all processors): 0
              INFOG(18) (size of all MUMPS internal data allocated during factorization:␣
↪value on the most memory consuming processor): 0
              INFOG(19) (size of all MUMPS internal data allocated during factorization:␣
↪sum over all processors): 0
              INFOG(20) (estimated number of entries in the factors): 7603
              INFOG(21) (size in MB of memory effectively used during factorization -␣
↪value on the most memory consuming processor): 0
              INFOG(22) (size in MB of memory effectively used during factorization -␣
↪sum over all processors): 0
              INFOG(23) (after analysis: value of ICNTL(6) effectively used): 0
```

(continues on next page)

(continued from previous page)

```
              INFOG(24) (after analysis: value of ICNTL(12) effectively used): 1
              INFOG(25) (after factorization: number of pivots modified by static␣
↪pivoting): 0
              INFOG(28) (after factorization: number of null pivots encountered): 0
              INFOG(29) (after factorization: effective number of entries in the factors␣
↪(sum over all processors)): 7603
              INFOG(30, 31) (after solution: size in Mbytes of memory used during␣
↪solution phase): 0, 0
              INFOG(32) (after analysis: type of analysis done): 1
              INFOG(33) (value used for ICNTL(8)): 7
              INFOG(34) (exponent of the determinant if determinant is requested): 0
              INFOG(35) (after factorization: number of entries taking into account BLR␣
↪factor compression - sum over all processors): 7603
              INFOG(36) (after analysis: estimated size of all MUMPS internal data for␣
↪running BLR in-core - value on the most memory consuming processor): 0
              INFOG(37) (after analysis: estimated size of all MUMPS internal data for␣
↪running BLR in-core - sum over all processors): 0
              INFOG(38) (after analysis: estimated size of all MUMPS internal data for␣
↪running BLR out-of-core - value on the most memory consuming processor): 0
              INFOG(39) (after analysis: estimated size of all MUMPS internal data for␣
↪running BLR out-of-core - sum over all processors): 0
  linear system matrix = precond matrix:
  Mat Object: 1 MPI process
    type: seqaij
    rows=205, cols=205
    total: nonzeros=2863, allocated nonzeros=2863
    total number of mallocs used during MatSetValues calls=0
      using I-node routines: found 137 nodes, limit used is 5
```

In the previous example we have solved using a CG method and a LU factorization preconditioned computed using MUMPS. We can access all this information from the using the `view` of the `solver.ksp` object. We can also solve using more exotic methods and preconditioners, such as a biconjugate gradient method using HYPRE for preconditioning.

[5]:
```
%%px
solver = KrylovSolver(a,fes, solverParameters={'ksp_type': 'bcgs', 'pc_type': 'hypre'})
gfu = solver.solve(f)
Draw(gfu,mesh, "solution")
solver.ksp.view()
```

```
[output:0]
```

```
WebGuiWidget(layout=Layout(height='500px', width='100%'), value={'gui_settings': {},
↪'ngsolve_version': '6.2.2...
```

```
[stdout:0] KSP Object: 1 MPI process
  type: bcgs
  maximum iterations=10000, initial guess is zero
  tolerances:  relative=1e-05, absolute=1e-50, divergence=10000.
  left preconditioning
  using PRECONDITIONED norm type for convergence test
PC Object: 1 MPI process
  type: hypre
    HYPRE BoomerAMG preconditioning
```

(continues on next page)

(continued from previous page)

```
      Cycle type V
      Maximum number of levels 25
      Maximum number of iterations PER hypre call 1
      Convergence tolerance PER hypre call 0.
      Threshold for strong coupling 0.25
      Interpolation truncation factor 0.
      Interpolation: max elements per row 0
      Number of levels of aggressive coarsening 0
      Number of paths for aggressive coarsening 1
      Maximum row sums 0.9
      Sweeps down         1
      Sweeps up           1
      Sweeps on coarse    1
      Relax down          symmetric-SOR/Jacobi
      Relax up            symmetric-SOR/Jacobi
      Relax on coarse     Gaussian-elimination
      Relax weight  (all)      1.
      Outer relax weight (all) 1.
      Maximum size of coarsest grid 9
      Minimum size of coarsest grid 1
      Using CF-relaxation
      Not using more complex smoothers.
      Measure type        local
      Coarsen type        Falgout
      Interpolation type  classical
      SpGEMM type         hypre
  linear system matrix = precond matrix:
  Mat Object: 1 MPI process
    type: seqaij
    rows=205, cols=205
    total: nonzeros=2863, allocated nonzeros=2863
    total number of mallocs used during MatSetValues calls=0
      using I-node routines: found 137 nodes, limit used is 5
```

It is also possible to import any PETSc PC object and used it NGSolve using the PETScPC NGSolve `Preconditioner`. This is very useful if one is interested in using NGSolve linear algebra functionality with "foreign" preconditioner.

[6]:
```
%%px
from ngsPETSc import pc
from ngsolve import Preconditioner, GridFunction
from ngsolve.solvers import CG
pre = Preconditioner(a, "PETScPC", pc_type="hypre")
gfu = GridFunction(fes)
gfu.vec.data = CG(a.mat, rhs=f.vec, pre=pre, printrates=mesh.comm.rank==0)
Draw(gfu,mesh, "solution")
```

```
[stdout:0] CG iteration 1, residual = 2.3515313068754367
CG iteration 2, residual = 0.11703951230766897
CG iteration 3, residual = 0.02532238998559266
CG iteration 4, residual = 0.002762280521164809
CG iteration 5, residual = 0.00036992984473413113
CG iteration 6, residual = 5.0193813921945675e-05
```

(continues on next page)

(continued from previous page)

```
CG iteration 7, residual = 8.048185966060108e-06
CG iteration 8, residual = 1.1206592184815855e-06
CG iteration 9, residual = 1.3930271656429977e-07
CG iteration 10, residual = 2.358932103568244e-08
CG iteration 11, residual = 3.2889334523253205e-09
CG iteration 12, residual = 4.688724686311323e-10
CG iteration 13, residual = 6.51694553700458e-11
CG iteration 14, residual = 9.11213172981437e-12
CG iteration 15, residual = 7.541816903801115e-13
```

```
[stderr:0] WARNING: kwarg 'pc_type' is an undocumented flags option for class <class
↪'ngsolve.comp.Preconditioner'>, maybe there is a typo?
```

```
[output:0]
```

```
WebGuiWidget(layout=Layout(height='500px', width='100%'), value={'gui_settings': {},
↪'ngsolve_version': '6.2.2...
```

Out[0:5]: BaseWebGuiScene

This page was generated from notebooks/Eigenvalue Problems.ipynb.

6.4 SLEPc EPS

In this tutorial we will have an in-depth look in to the way we can use a SLEPc EPS object to solve eigenvalue problem arising from the discretization of a linear partial differential equation. We will focus our attention the Maxwell eigenvalue problem in variational form, i.e. find $u_h \in P^k(\mathcal{T}_h)$ such for any $v_h \in P^k(\mathcal{T}_h)$ the following equation is verified,

$$(\nabla \vec{u}_h, \nabla \vec{v}_h) = \lambda(\vec{u}_h, \vec{v}_h), \quad \text{for a certain data } f \in [P^k(\mathcal{T}_h)]^*.$$

[1]:
```
from ipyparallel import Cluster
c = await Cluster().start_and_connect(n=1, activate=True)
```

```
Starting 1 engines with <class 'ipyparallel.cluster.launcher.LocalEngineSetLauncher'>
```

```
  0%|          | 0/1 [00:00<?, ?engine/s]
```

Let's test if the cluster has been initialized correctly by checking the size of the COMM_WORLD-

[2]:
```
%%px
from mpi4py.MPI import COMM_WORLD
COMM_WORLD.Get_size()
```

Out[0:1]: 1

First we need to construct the distributed mesh over which we will construct the finite element spaces we are interested in.

[3]:
```
%%px
from ngsolve import Mesh, BilinearForm, LinearForm, H1
from ngsolve import x, y, dx, grad
from netgen.occ import Box, OCCGeometry
from ngsolve.webgui import *

if COMM_WORLD.rank == 0:
    cube1 = Box( (-1,-1,-1), (1,1,1) )
    cube2 = Box( (0,0,0), (2,2,2) )
    cube2.edges.hpref=1
    fichera = cube1-cube2
    mesh = Mesh(OCCGeometry(fichera).GenerateMesh(maxh=0.2).Distribute(COMM_WORLD))
else:
    mesh = Mesh(ngm.Mesh.Receive(COMM_WORLD))
mesh.RefineHP(levels=2, factor=0.2)
Draw(mesh)
```

```
[0:execute]
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[2], line 11
      9     cube2.edges.hpref=1
     10     fichera = cube1-cube2
---> 11     mesh =␣
↪Mesh(OCCGeometry(fichera).GenerateMesh(maxh=0.2).Distribute(COMM_WORLD))
     12 else:
     13     mesh = Mesh(ngm.Mesh.Receive(COMM_WORLD))

TypeError: Distribute(): incompatible function arguments. The following argument types␣
↪are supported:
    1. (self: netgen.libngpy._meshing.Mesh, comm: netgen.libngpy._meshing.MPI_Comm) ->␣
↪netgen.libngpy._meshing.Mesh

Invoked with: <netgen.libngpy._meshing.Mesh object at 0x7fa7240dd1f0>, <mpi4py.MPI.
↪Intracomm object at 0x7fa7446dc5a0>
```

```
1 errors
```

[7]:
```
%%px
fes = H1(mesh, order=2,dirichlet=".*")
u,v = fes.TnT()
a = BilinearForm(grad(u)*grad(v)*dx, symmetric=True)
a.Assemble()
m = BilinearForm(-1*u*v*dx, symmetric=True)
m.Assemble()
```

```
Out[0:3]: <ngsolve.comp.BilinearForm at 0x7f211368e970>
```

[8]:
```
%%px
from ngsPETSc import EigenSolver
solver = EigenSolver((m, a), fes, 40, solverParameters={"eps_type":"lobpcg", "st_type":
↪"precond"})
solver.solve()
print ("Eigenvalues")
```

(continues on next page)

(continued from previous page)

```
for i in range(40):
    print(solver.eigenValue(i))
eigenMode, _ = solver.eigenFunction(0)
Draw(eigenMode,clipping={"y":0, "z":-1})
```

```
%px:   0%|          | 0/1 [00:00<?, ?tasks/s]
```

```
[stdout:0] Eigenvalues
(10.541301534822994+0j)
(16.60493282003399+0j)
(16.60554869090414+0j)
(20.534265668499938+0j)
(23.654986807854243+0j)
(23.65622585324029+0j)
(29.683558607578036+0j)
(30.684386447184774+0j)
(30.690456785524947+0j)
(31.67906098862113+0j)
(35.07209294568837+0j)
(37.47513564911229+0j)
(37.4853478878991+0j)
(40.79529556143462+0j)
(40.80726452215837+0j)
(40.89475051082556+0j)
(45.40513095726472+0j)
(47.0459953779459+0j)
(47.05341831122042+0j)
(50.80507494901877+0j)
(50.82443592673758+0j)
(50.93998033036734+0j)
(52.506664535386115+0j)
(52.52058315670419+0j)
(53.706029221875944+0j)
(53.97735254600284+0j)
(53.99877738529949+0j)
(54.0430262930373+0j)
(59.747774735822524+0j)
(59.77322066390943+0j)
(59.793365618440596+0j)
(62.43132544748816+0j)
(62.44608533405657+0j)
(62.522586446686745+0j)
(67.18997618666373+0j)
(67.22744306152411+0j)
(67.86334994447714+0j)
(70.60178200122165+0j)
(70.62839551412465+0j)
(70.9498983748746+0j)
```

```
[output:0]
```

```
WebGuiWidget(layout=Layout(height='500px', width='100%'), value={'gui_settings': {},
↪'ngsolve_version': '6.2.2...
```

```
Out[0:4]: BaseWebGuiScene
```

[10]:
```
%%px
eigenModes, _ = solver.eigenFunctions(range(10))
Draw(eigenModes,clipping={"x":0, "z":-1})
```

```
%px:   0%|          | 0/1 [00:00<?, ?tasks/s]
```

```
[output:0]
```

```
WebGuiWidget(layout=Layout(height='500px', width='100%'), value={'gui_settings': {},
↪'ngsolve_version': '6.2.2...
```

```
Out[0:6]: BaseWebGuiScene
```

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search