In [1]:
```
import numpy as np
import dolfin as dl
from ufl import lhs, rhs, replace
import scipy.sparse as sps
import scipy.sparse.linalg as spla
import scipy.linalg as sla
import matplotlib.pyplot as plt
from pathlib import Path
from nalger_helper_functions import *
import meshio
import sys
from scipy.spatial import KDTree as scipy_KDTree
from scipy.optimize import root_scalar
from localpsf import localpsf_root
from localpsf.newtoncg import newtoncg_ls, cgsteihaug
from nalger_helper_functions import *

import hlibpro_python_wrapper as hpro
from localpsf.product_convolution_kernel import ProductConvolutionKernel
from localpsf.product_convolution_hmatrix import make_hmatrix_from_kernel
from localpsf.derivatives_at_point import StokesDerivativesAtPoint
from localpsf.bilaplacian_regularization import BiLaplacianRegularization
```

Options

In [2]:
```
# mesh_type = 'coarse'
# mesh_type = 'medium'
mesh_type = 'fine'
noise_level=1e-2
relative_prior_correlation_length=0.1
save_plots=True
gamma=1e4 # regularization gamma
m0_constant_value = 1.5 * 7.
rel_correlation_length = 0.1
reg_robin_bc=True
# mtrue_type = 'aces_building'
mtrue_type = 'angel_peak'
# mtrue_type = 'sinusoid'
# mtrue_string = 'm0 - (m0 / 7.)*std::cos((x[0]*x[0]+x[1]*x[1])*pi/(Radius*Radiu
mtrue_string = 'm0 - (m0 / 7.)*std::cos(2.*x[0]*pi/Radius)'
solver_type = 'mumps'
# solver_type = 'default'
# solver_type = 'petsc'
# solver_type = 'umfpack'
# solver_type = 'superlu'
outflow_constant = 1.0e6

num_batches = 5 # number of batches used for Newton solves
tau = 3.0
num_neighbors = 10
hmatrix_tol = 1e-6
newton_rtol = 1e-8
# gamma_morozov = 15428.012899090285 # medium mesh, noise_level=1e-2
# gamma_morozov= 6932.714344119822 # fine mesh, noise_level=1e-2
# gamma_morozov= 6975.784285975133 # fine mesh, noise_level=1e-2
gamma_morozov = None
```

```
forcing_sequence_power = 0.5 # 1.0
num_gn_iter = 5

run_finite_difference_checks = False
check_gauss_newton_hessian = False

all_num_batches = [1, 5, 25] # number of batches used for spectral comparisons
```

In [3]:

```
save_dir = localpsf_root / 'numerical_examples' / 'stokes' / 'data'
save_dir.mkdir(parents=True, exist_ok=True)
save_dir_str = str(save_dir)
```

Mesh

Load unmodified mesh

In [4]:

```
if mesh_type == 'coarse':
    mfile_name = str(localpsf_root) + "/numerical_examples/stokes/meshes/cylinde
    lam = 1e10
elif mesh_type == 'medium':
    mfile_name = str(localpsf_root) + "/numerical_examples/stokes/meshes/cylinde
    lam = 1e11
elif mesh_type == 'fine':
    mfile_name = str(localpsf_root) + "/numerical_examples/stokes/meshes/cylinde
    lam = 1e12
else:
    raise RuntimeError('invalid mesh type '+mesh_type+', valid types are coarse,

mesh = dl.Mesh(mfile_name+".xml")

mesh
```

Out[4]:

Options

Menu Options

☐ Summary

☐ Color

☐ Warp

Boundary submeshes and subdomains

In [5]:

```
class BasalBoundary(dl.SubDomain):
    def inside(me, x, on_boundary):
        return dl.near(x[2], 0.) and on_boundary

class BasalBoundarySub(dl.SubDomain):
    def inside(me, x, on_boundary):
        return dl.near(x[2], 0.)

class TopBoundary(dl.SubDomain):
    def __init__(me, Height):
        me.Height = Height
        dl.SubDomain.__init__(me)
    def inside(me, x, on_boundary):
        return dl.near(x[1], me.Height) and on_boundary

boundary_markers = dl.MeshFunction("size_t", mesh, mfile_name+"_facet_region.xml
boundary_mesh = dl.BoundaryMesh(mesh, "exterior", True)
```

```
basal_mesh3D = dl.SubMesh(boundary_mesh, BasalBoundarySub())

basal_mesh3D
```

Out[5]: