```
# User NumBAT mesh implementation file


import matplotlib.patches as mplpatches
import numpy as np

from usermesh import UserGeometryBase

nmtoum = 0.001   # template radii are in nm but matplotlib plots are in microns


def _process_one_and_two_incls(params):
    nelts = 0
    gmshfile = ''

    if 'slab_b_x' in params:
        raise ValueError(
                f"NumBAT doesn't understand your geometry: with shape {params['inc_shape']}, I did not e
xpect values for slab_b.")

    if 'slab_a_x' in params:
        raise ValueError(
            f"NumBAT doesn't understand your geometry: with shape {params['inc_shape']}, I did not expect
values for slab_a.")

    if 'inc_a_x' in params:
        if 'coat_y' not in params and 'inc_b_x' not in params:  # One inclusion, no
 coating
            gmshfile = 'oneincl'   # used to be just '1'
            nelts = 2          # bkg, core (mat_a)


        elif 'coat_y' not in params and 'inc_b_x' in params:  # Two inclusions, no
coating
            gmshfile = 'twoincl'   # used to be just '2'
            nelts = 3          # bkg, core 1 (mat_a), core 2 (mat_b)


        # Two inclusions, with coating # TODO:implement
        elif 'coat_y' in params and 'inc_b_x' in params:
            raise NotImplementedError(
                'Have not implemented 2 coated inclusions.')

        elif 'coat_y' in params and 'inc_b_x' not in params:  # One inclusion, with
 coating # TODO:implement
            raise NotImplementedError(
                'Have not implemented 1 coated inclusions.')

        else:
            raise ValueError("NumBAT doesn't understand your geometry.")
    else:
        raise ValueError('must have at least one nonzero inclusion.')

    return gmshfile, nelts

def _process_one_and_two_incls_subs(msh_template, umb):
        # TODO: these are crazy small defaults
    subs = [('dx_in_nm = 100;', 'dx_in_nm = %f;', umb.get_param('domain_x'))]
    subs.append(('dy_in_nm = 50;', 'dy_in_nm = %f;', umb.get_param('domain_y')))
    subs.append(('a1 = 20;', 'a1 = %f;', umb.get_param('inc_a_x')))
    subs.append(('a1y = 10;', 'a1y = %f;', umb.get_param('inc_a_y')))
```

```

    subs.append(('lc = 0.1;', 'lc = %f;', umb.get_param('lc_bkg')))
    subs.append(('lc_refine_1 = lc/1;', 'lc_refine_1 = lc/%f;', umb.get_param('lc_refine_1')))
    subs.append(('lc_refine_2 = lc/1;', 'lc_refine_2 = lc/%f;', umb.get_param('lc_refine_2')))

    if msh_template in ['twoincl', '2', '2_on_s', '2_on_2s']:
        subs.append(('a2 = 10;', 'a2 = %f;', umb.get_param('inc_b_x')))
        subs.append(('a2y = 20;', 'a2y = %f;', umb.get_param('inc_b_y')))
        subs.append(('sep = 10;', 'sep = %f;', umb.get_param('two_inc_sep')))

        # geo = geo.replace('lc_refine_3 = lc/1;', 'lc_refine_3 = lc/%f;' % umb.
lc_refine_3)
    if msh_template == '2':
        subs.append(('yoff = -5;', 'yoff = %f;', umb.get_param('incs_y_offset')))

    if msh_template in ['1_on_slab', '1_on_2slabs', '1_on_slab', '2_on_2slabs']:
        subs.append(('slab_width = dx_in_nm;',
                     'slab_width = %f;', umb.get_param('slab_a_x')))
        subs.append(
            ('slab_height = 10;', 'slab_height = %f;', umb.get_param('.slab_a_y')))
        subs.append(
            ('lc_refine_3 = lc/1;', 'lc_refine_3 = lc/%f;', umb.get_param('lc_refine_3')))
        subs.append(
            ('lc_refine_4 = lc/1;', 'lc_refine_4 = lc/%f;', umb.get_param('lc_refine_4')))

    if msh_template in ['1_on_2slabs', '2_on_2slabs']:
        subs.append(('slab2_width = dx_in_nm;',
                     'slab2_width = %f;', umb.get_param('slab_b_x')))
        subs.append(
            ('slab2_height = 5;', 'slab2_height = %f;', umb.get_param('slab_b_y')))
        subs.append(
            ('lc_refine_3 = lc/1;', 'lc_refine_3 = lc/%f;',umb.get_param('.lc_refine_3')))
        subs.append(
            ('lc_refine_4 = lc/1;', 'lc_refine_4 = lc/%f;', umb.get_param('lc_refine_4')))


    return subs

class Circular(UserGeometryBase):

    def init_geometry(self):
        gmshfile, nelts = _process_one_and_two_incls(self._d_params)
        desc = '''A NumBAT geometry template for a circular waveguide.'''
        self.set_properties('circular', nelts, True, desc)
        self._gmsh_template_filename = gmshfile  # special case where Circular a
nd Rectangular share common gmshfile, so geom name and geom file are different


    def apply_parameters(self):

        subs = _process_one_and_two_incls_subs(self._gmsh_template_filename, sel
f)
        subs.append(('rect = 1;', 'rect = 0;', ''))  # apply circularness

        return subs

    def draw_mpl_frame(self, ax):

        rad = self.get_param('inc_a_x') * 0.5

        circ = mplpatches.Circle((0, 0), rad*nmtoum, facecolor=None, fill=False,
 edgecolor='gray',
                                    linewidth=.75)
        ax.add_patch(circ)


class Rectangular(UserGeometryBase):

    def init_geometry(self):
        gmshfile, nelts = _process_one_and_two_incls(self._d_params)
        desc = '''A NumBAT geometry template for a rectangular waveguide.'''
        self.set_properties('rectangular', nelts, False, desc)
        self._gmsh_template_filename = gmshfile # special case where Circular and Rectangular share common gmshfile, so geom name and geom file are different


    def apply_parameters(self):
        subs = _process_one_and_two_incls_subs(self._gmsh_template_filename, self)


        return subs


    def draw_mpl_frame(self, ax):

        wid = self.get_param('inc_a_x') * nmtoum
        hgt = self.get_param('inc_a_y') * nmtoum

        ax.add_patch(mplpatches.Rectangle((-wid/2, -hgt/2), wid, hgt,
                      facecolor=None, fill=False, edgecolor='gray', linewidth=.75))

    def check_dimensions(self):
        dom_x = self.get_param('domain_x')
        dom_y = self.get_param('domain_y')
        wid = self.get_param('inc_a_x')
        hgt = self.get_param('inc_a_y')

        msg= ''

        if wid >= dom_x: msg += 'Waveguide width (inc_a_x) is larger than domain width (domain_x).\n'
        if hgt >= dom_y: msg += 'Waveguide height (inc_a_y) is larger than domain height (domain_y).\n'

        dims_ok = not len(msg)
        return dims_ok, msg



class TwoIncl(UserGeometryBase):

    def init_geometry(self):
        gmshfile, nelts = _process_one_and_two_incls(self._d_params)
        desc = '''A NumBAT geometry template for a double inclusion waveguide.'''
        self.set_properties('twoincl', nelts, True, desc)
        self._gmsh_template_filename = gmshfile  # special case where Circular and Rectangular share common gmshfile, so geom name and geom file are different

    def apply_parameters(self):
```

```
        subs = _process_one_and_two_incls_subs(self._gmsh_template_filename, self)

        return subs

    def draw_mpl_frame(self, ax):

        widl = self.get_param('inc_a_x') * nmtoum
        hgtl = self.get_param('inc_a_y') * nmtoum
        widr = self.get_param('inc_b_x') * nmtoum
        hgtr = self.get_param('inc_b_y') * nmtoum
        sep  = self.get_param('two_inc_sep') * nmtoum
        yoff  = self.get_param('yoff') * nmtoum

        shape = self.get_param('inc_shape')

        if shape == 'circular':
            ax.add_patch(mplpatches.Circle( (-sep/2, 0), widl,
                facecolor=None, fill=False, edgecolor='gray', linewidth=.75))

            ax.add_patch(mplpatches.Circle( (sep/2, yoff), widr,
                facecolor=None, fill=False, edgecolor='gray', linewidth=.75))

        else:
            ax.add_patch(mplpatches.Rectangle( (-sep/2-widl/2, -hgtl/2), widl, hgtl,
                facecolor=None, fill=False, edgecolor='gray', linewidth=.75))

            ax.add_patch(mplpatches.Rectangle( (sep/2-widr/2, yoff-hgtr/2), widr, hgtr,
                facecolor=None, fill=False, edgecolor='gray', linewidth=.75))

class TwoInclVert(UserGeometryBase):

    def init_geometry(self):
        #gmshfile, nelts = _process_one_and_two_incls(self._d_params)
        desc = '''A NumBAT geometry template for a rectangular double inclusion waveguide arranged vertically.'''
        self.set_properties('twoinclvert', 3, True, desc)

        self.set_required_parameters(['inc_a_w', 'inc_a_h', 'inc_b_w', 'inc_b_h',
            'inc_sep_x', 'inc_sep_y'],  num_mats=3)
        self.set_allowed_parameters(['lc_bkg', 'lc_refine_1', 'lc_refine_2'],  num_allowed_mats=3)

    def apply_parameters(self):

        subs = [('dx_in_nm = 100.0;',    'dx_in_nm = %f;',       self.get_param('domain_x')),
                ('dy_in_nm = 50.0;',     'dy_in_nm = %f;',       self.get_param('domain_y')),
                ('inc_a_w = 20.0;',      'inc_a_w = %f;',        self.get_param('inc_a_w')),
                ('inc_a_h = 10.0;',      'inc_a_h = %f;',        self.get_param('inc_a_h')),
                ('inc_b_w = 20.0;',      'inc_b_w = %f;',        self.get_param('inc_b_w')),
                ('inc_b_h = 10.0;',      'inc_b_h = %f;',        self.get_param('inc_b_h')),
                ('inc_sep_x = 5.0;',     'inc_sep_x = %f;',      self.get_param('inc_sep_x')),
                ('inc_sep_y = 15.0;',    'inc_sep_y = %f;',      self.get_param('inc_sep_y')
)),
                ('lc_bkg = 0.05;',          'lc_bkg = %f;',               self.get_param('lc
_bkg')),
                ('lc_refine_1 = lc/2.0;', 'lc_refine_1 = lc/%f;', self.get_param('lc_refine_1')),
                ('lc_refine_2 = lc/2.0;', 'lc_refine_2 = lc/%f;', self.get_param('lc_refine_2'))
                ]


        return subs

    def check_dimensions(self):
        '''The first box must be higher. Which means yoff must be positive.'''

        dom_x = self.get_param('domain_x')
        dom_y = self.get_param('domain_y')
        inc_a_w = self.get_param('inc_a_w')
        inc_a_h = self.get_param('inc_a_h')
        inc_b_w = self.get_param('inc_b_w')
        inc_b_h = self.get_param('inc_b_h')
        inc_sep_x = self.get_param('inc_sep_x')
        inc_sep_y = self.get_param('inc_sep_y')

        # waveguides can't touch so either they are fully separated in y, or in
x
        # either way the upper one must be higher

        msg= ''

        if (inc_a_w + inc_sep_x)/2 > dom_x/2 or (-inc_a_w + inc_sep_x)/2 < -dom_
x/2:
            msg+='Waveguide 1 is falling outside the x-domain (Check parameters: inc_a_w, inc_sep_x, doma
in_x). \n'
        if (inc_a_h + inc_sep_y)/2 > dom_y/2 or (-inc_a_h + inc_sep_y)/2 < -dom_
y/2:
            msg+='Waveguide 1 is falling outside the x-domain (Check parameters: inc_a_h, inc_sep_y, domai
n_y). \n'

        if (inc_b_w + inc_sep_x)/2 > dom_x/2 or (-inc_b_w + inc_sep_x)/2 < -dom_
x/2:
            msg+='Waveguide 1 is falling outside the x-domain (Check parameters: inc_b_w, inc_sep_x, doma
in_x). \n'
        if (inc_b_h + inc_sep_y)/2 > dom_y/2 or (-inc_b_h + inc_sep_y)/2 < -dom_
y/2:
            msg+='Waveguide 1 is falling outside the x-domain (Check parameters: inc_b_h, inc_sep_y, domai
n_y). \n'


        yoverlap = inc_sep_y - (inc_a_h+inc_b_h)/2
        minysep = inc_sep_y - max(inc_a_h,inc_b_h)/2

        xoverlap = abs(inc_sep_x) - (inc_a_w+inc_b_w)/2
        if inc_sep_y <= 0 or minysep<=0: msg += 'Vertical separation of the two guides must be
positive (Check parameter: inc_sep_y) \n'

        if yoverlap <= 0 and xoverlap <=0:
            msg+= 'The two waveguides are overlapping (Check parameters: inc_a_w, inc_a_h, inc_b_w, inc_
b_h, inc_sep_x, inc_sep_y). \n'

        dims_ok = not len(msg)

        return dims_ok, msg
```

```
    def draw_mpl_frame(self, ax):

        widu = self.get_param('inc_a_w') * nmtoum
        hgtu = self.get_param('inc_a_h') * nmtoum
        widl = self.get_param('inc_b_w') * nmtoum
        hgtl = self.get_param('inc_b_h') * nmtoum
        xsep  = self.get_param('inc_sep_x') * nmtoum
        ysep  = self.get_param('inc_sep_y') * nmtoum

        xu = -xsep/2 -widu/2
        yu = ysep/2-hgtu/2
        xl = xsep/2-widl/2
        yl = -ysep/2-hgtu/2

        ax.add_patch(mplpatches.Rectangle( (xu,yu), widu, hgtu,
            facecolor=None, fill=False, edgecolor='gray', linewidth=.75))

        ax.add_patch(mplpatches.Rectangle( (xl,yl), widl, hgtl,
            facecolor=None, fill=False, edgecolor='gray', linewidth=.75))


class Triangular(UserGeometryBase):

    def init_geometry(self):
        desc = '''A NumBAT geometry template for a triangular waveguide.'''
        self.set_properties('triangular', 2, False, desc)
        self.set_required_parameters(['base_width', 'peak_xoff', 'peak_height'],  num_m
ats=1)
        self.set_allowed_parameters(['lc_bkg', 'lc_refine_1','lc_refine_2'], num_allowed
_mats=2)
        self.set_parameter_help(
            {
                'base_width'  : "length of base of triangle along x-axis",
                'peak_xoff'  :  "horizontal offset of peak along x-axis from left vertex",
                'peak_height': "perpendicular height of triangle",
                'material_bkg': "background material",
                'material_a': "material of triangular core",
                }
            )

    def apply_parameters(self):
        subs = [('dx_in_nm = 1000.0;',     'dx_in_nm = %f;',        self.get_param('domain
_x')),
                ('dy_in_nm = 1000.0;',     'dy_in_nm = %f;',        self.get_param('domain
_y')),
                ('base_width = 600.0;',    'base_width = %f;',      self.get_param('base_width
')),
                ('peak_xoff = 200.0;',     'peak_xoff = %f;',       self.get_param('peak_xoff
')),
                ('peak_height = 400.0;',   'peak_height = %f;',     self.get_param('peak_height
')),
                ('lc = 0.1;',              'lc = %f;',              self.get_param('lc_bk
g')),
                ('lc_refine_1 = lc/1.0;', 'lc_refine_1 = lc/%f;', self.get_param('lc_refine_1')),
                ('lc_refine_2 = lc/3.0;', 'lc_refine_2 = lc/%f;', self.get_param('lc_refine_2'))
                ]

        return subs

    def check_dimensions(self):
        dom_x = self.get_param('domain_x')
        dom_y = self.get_param('domain_y')
        base_wid = self.get_param('base_width')
        peak_xoff = self.get_param('peak_xoff')
        peak_height = self.get_param('peak_height')

        peak_locx = -base_wid/2 + peak_xoff

        msg= ''

        if base_wid >= dom_x: msg += 'Waveguide base width (base_width) is larger than the domain
width (domain_x).\n'

        if peak_locx < -dom_x/2 or peak_locx > dom_x/2: msg += 'Waveguide peak is outs
ide the x-domain (domain_x).\n'
        if peak_height > dom_y/2 or peak_height< -dom_y/2: msg += 'Waveguide height
(peak_height) is too large for the domain height (domain_y).\n'

        dims_ok = not len(msg)
        return dims_ok, msg

    def draw_mpl_frame(self, ax):
        wid = self.get_param('base_width') * nmtoum
        xoff = self.get_param('peak_xoff') * nmtoum
        hgt = self.get_param('peak_height') * nmtoum
        vertices = np.array([[-wid/2, -hgt/2], [-wid/2+xoff, hgt/2], [wid/2,-hgt
/2]])

        ax.add_patch(mplpatches.Polygon(vertices,
            facecolor=None, fill=False, edgecolor='gray', linewidth=.75))



def make_onion_subs(umb):
    subs = [('dx_in_nm = 2000;', 'dx_in_nm = %f;', umb.get_param('domain_x'))]
    subs.append(('dy_in_nm = 2000;', 'dy_in_nm = %f;', umb.get_param('domain_y')))
    subs.append(('a1 = 100;', 'a1 = %f;', umb.get_param('inc_a_x')))
    subs.append(('a2 = 100;', 'a2 = %f;', umb.get_param('inc_b_x')))
    subs.append(('a3 = 100;', 'a3 = %f;', umb.get_param('inc_c_x')))
    subs.append(('a4 = 100;', 'a4 = %f;', umb.get_param('inc_d_x')))
    subs.append(('a5 = 100;', 'a5 = %f;', umb.get_param('inc_e_x')))
    subs.append(('a6 = 100;', 'a6 = %f;', umb.get_param('inc_f_x')))
    subs.append(('a7 = 100;', 'a7 = %f;', umb.get_param('inc_g_x')))
    subs.append(('a8 = 100;', 'a8 = %f;', umb.get_param('inc_h_x')))
    subs.append(('a9 = 100;', 'a9 = %f;', umb.get_param('inc_i_x')))
    subs.append(('a10 = 100;', 'a10 = %f;', umb.get_param('inc_j_x')))
    subs.append(('a11 = 100;', 'a11 = %f;', umb.get_param('inc_k_x')))
    subs.append(('a12 = 100;', 'a12 = %f;', umb.get_param('inc_l_x')))
    subs.append(('a13 = 100;', 'a13 = %f;', umb.get_param('inc_m_x')))
    subs.append(('a14 = 100;', 'a14 = %f;', umb.get_param('inc_n_x')))
    subs.append(('a15 = 100;', 'a15 = %f;', umb.get_param('inc_o_x')))
    subs.append(('lc = 0.1;', 'lc = %f;', umb.get_param('lc_bkg')))
    subs.append(
        ('lc_refine_1 = lc/1;', 'lc_refine_1 = lc/%f;', umb.get_param('lc_refine_1')))
    subs.append(
        ('lc_refine_2 = lc/1;', 'lc_refine_2 = lc/%f;', umb.get_param('lc_refine_2')))

    return subs


def draw_onion_frame(ax, umb):

    layers = ('inc_a_x', 'inc_b_x', 'inc_c_x', 'inc_d_x', 'inc_e_x',
```

```
                                    'inc_f_x', 'inc_g_x', 'inc_h_x', 'inc_i_x', 'inc_j_x'
,
                                    'inc_k_x', 'inc_l_x', 'inc_m_x', 'inc_n_x', 'inc_o_x
')


    rad = 0
    for sl in layers:
        lwid = umb.get_param(sl)
        if lwid is not None:
            if sl == 'inc_a_x':
                rad += lwid/2  # inc_a_x is diameter
            else:
                rad += lwid
            ax.add_patch( mplpatches.Circle((0, 0), rad*nmtoum,
                             facecolor=None, fill=False, edgecolor='gray', linewi
dth=.75))

class Onion(UserGeometryBase):
    def init_geometry(self):
        desc = '''A NumBAT geometry template for a many-layer circular waveguide in a square domain.''
'

        self.set_properties('onion', 16, True, desc)
        self.set_required_parameters(['inc_a_x'],  num_mats=2)
        self.set_allowed_parameters(['lc_refine_1','lc_refine_2',
                                     'inc_b_x', 'inc_c_x', 'inc_d_x', 'inc_e_x',
                                    'inc_f_x', 'inc_g_x', 'inc_h_x', 'inc_i_x', 'inc_j_x'
,
                                    'inc_k_x', 'inc_l_x', 'inc_m_x', 'inc_n_x', 'inc_o_x
'],
                                    num_allowed_mats=16)

    def apply_parameters(self):
        subs = make_onion_subs(self)
        return subs

    def draw_mpl_frame(self, ax): draw_onion_frame(ax, self)




class Onion1(UserGeometryBase):
    def init_geometry(self):
        desc = '''A NumBAT geometry template for a one-layer circular waveguide in a square domain.'''
        nt=2

        self.set_properties('onion1', nt, True, desc)

        self.set_required_parameters(['inc_a_x'],  num_mats=nt)
        self.set_allowed_parameters(['lc_bkg', 'lc_refine_2'],  num_allowed_mats=nt)
        self.set_parameter_help(
                { 'inc_a_x': "diameter of central cylinder",
                  'material_a': "material of central cylinder",
                  'lc_bkg': "mesh spacing on outer boundary",
                  'lc_refine_2': "mesh refinement on cylinder 1",
                  }
                )

    def apply_parameters(self):
        subs = make_onion_subs(self)
        return subs
    def check_dimensions(self):
        dom_x = self.get_param('domain_x')
        dom_y = self.get_param('domain_y')
        diam_a = self.get_param('inc_a_x')

        msg= ''

        if diam_a >= dom_x: msg += 'Waveguide cylinder a (inc_a_x) has diameter larger than domain
width (domain_x).\n'
        if diam_a >= dom_y: msg += 'Waveguide cylinder a (inc_a_x) has diameter larger than domain
height (domain_y).\n'

        dims_ok = not len(msg)
        return dims_ok, msg

    def draw_mpl_frame(self, ax): draw_onion_frame(ax, self)



class Onion2(UserGeometryBase):
    def init_geometry(self):
        desc = '''A NumBAT geometry template for a two-layer circular waveguide in a square domain.'''

        nt = 3
        self.set_properties('onion2', nt, True, desc)

        self.set_required_parameters(['inc_a_x', 'inc_b_x'],  num_mats=nt)
        self.set_allowed_parameters(['lc_bkg', 'lc_refine_2'],  num_allowed_mats=nt)

        self.set_parameter_help(
                { 'inc_a_x' : "diameter of central (a) cylinder",
                  'inc_b_x' : "annular radius of second (b) ring",
                  'material_a' : "material of central (a) cylinder",
                  'material_a' : "material of second (b) ring",
                  'lc_bkg' : "mesh spacing on outer boundary",
                  'lc_refine_2' : "mesh refinement on cylinders",
                  }
                )

    def apply_parameters(self):
        subs = make_onion_subs(self)
        return subs

    def check_dimensions(self):
        dom_x = self.get_param('domain_x')
        dom_y = self.get_param('domain_y')
        rad_a = self.get_param('inc_a_x')/2.0
        rad_ann_b = self.get_param('inc_b_x')

        msg= ''

        diam_outer = 2*(rad_a+rad_ann_b)

        if diam_outer >= dom_x: msg += 'Outer cylinder has total diameter larger than domain width
(domain_x).\n'
        if diam_outer >= dom_y: msg += 'Outer cylinder has total diameter larger than domain height
(domain_y).\n'

        dims_ok = not len(msg)
        return dims_ok, msg
```

```
    def draw_mpl_frame(self, ax): draw_onion_frame(ax, self)



class Onion3(UserGeometryBase):
    def init_geometry(self):
        desc = '''A NumBAT geometry template for a three-layer circular waveguide in a square domain.''
'
        nt=4
        self.set_properties('onion3', nt, True, desc)

        self.set_required_parameters(['inc_a_x', 'inc_b_x', 'inc_c_x'],  num_mats=nt)
        self.set_allowed_parameters(['lc_bkg', 'lc_refine_2'],  num_allowed_mats=nt)

        self.set_parameter_help(
                { 'inc_a_x' : "diameter of central (a) cylinder",
                  'inc_b_x' : "annular radius of second (b) ring",
                  'inc_c_x' : "annular radius of third (c) ring",
                  'material_a' : "material of central (a) cylinder",
                  'material_b' : "material of second (b) ring",
                  'material_c' : "material of third (c) ring",
                  'lc_bkg' : "mesh spacing on outer boundary",
                  'lc_refine_2' : "mesh refinement on cylinders",
                  }
                )


    def apply_parameters(self):
        subs = make_onion_subs(self)
        return subs

    def check_dimensions(self):
        dom_x = self.get_param('domain_x')
        dom_y = self.get_param('domain_y')
        rad_a = self.get_param('inc_a_x')/2.0
        rad_ann_b = self.get_param('inc_b_x')
        rad_ann_c = self.get_param('inc_c_x')

        msg= ''

        diam_outer = 2*(rad_a+rad_ann_b+rad_ann_c)

        if diam_outer >= dom_x: msg += 'Outer cylinder has total diameter larger than domain width
(domain_x).\n'
        if diam_outer >= dom_y: msg += 'Outer cylinder has total diameter larger than domain height
(domain_y).\n'
        dims_ok = not len(msg)
        return dims_ok, msg


    def draw_mpl_frame(self, ax): draw_onion_frame(ax, self)



class CircOnion(UserGeometryBase):
    def init_geometry(self):
        desc = '''A NumBAT geometry template for a many-layer circular waveguide in a circular domain.'
''
        self.set_properties('circ_onion', 16, True, desc)

    def apply_parameters(self):
        subs = make_onion_subs(self)
        return subs

    def draw_mpl_frame(self, ax): draw_onion_frame(ax, self)

class CircOnion1(UserGeometryBase):
    def init_geometry(self):
        desc = '''A NumBAT geometry template for a one-layer circular waveguide in a circular domain.''
'
        self.set_properties('circ_onion1', 2, True, desc)

    def apply_parameters(self):
        subs = make_onion_subs(self)
        return subs

    def draw_mpl_frame(self, ax): draw_onion_frame(ax, self)

class CircOnion2(UserGeometryBase):
    def init_geometry(self):
        desc = '''A NumBAT geometry template for a two-layer circular waveguide in a circular domain.''
'
        self.set_properties('circ_onion2', 3, True, desc)

    def apply_parameters(self):
        subs = make_onion_subs(self)
        return subs

    def draw_mpl_frame(self, ax): draw_onion_frame(ax, self)

class CircOnion3(UserGeometryBase):
    def init_geometry(self):
        desc = '''A NumBAT geometry template for a three-layer circular waveguide in a circular domain.'
''
        self.set_properties('circ_onion3', 4, True, desc)

        self.set_required_parameters(['inc_a_x', 'inc_b_x', 'inc_c_x'],  num_mats=nt)
        self.set_allowed_parameters(['lc_bkg', 'lc_refine_2'],  num_allowed_mats=nt)

        self.set_parameter_help(
                { 'inc_a_x': "diameter of central (a) cylinder",
                  'inc_b_x': "annular radius of second (b) ring",
                  'inc_c_x': "annular radius of third (c) ring",
                  'material_a': "material of central (a) cylinder",
                  'material_b': "material of second (b) ring",
                  'material_c': "material of third (c) ring",
                  'lc_bkg': "mesh spacing on outer boundary",
                  'lc_refine_2': "mesh refinement on cylinders",
                  }
                )


    def apply_parameters(self):
        subs = make_onion_subs(self)
        return subs

    def check_dimensions(self):
        dom_x = self.get_param('domain_x')
        dom_y = self.get_param('domain_y')
        rad_a = self.get_param('inc_a_x')/2.0
        rad_ann_b = self.get_param('inc_b_x')
        rad_ann_c = self.get_param('inc_c_x')

        msg= ''
```

```

        diam_outer = 2*(rad_a+rad_ann_b+rad_ann_c)

        if diam_outer >= dom_x: msg += 'Outer cylinder has total diameter larger than domain width
(domain_x).\n'
        if diam_outer >= dom_y: msg += 'Outer cylinder has total diameter larger than domain height
(domain_y).\n'
        dims_ok = not len(msg)
        return dims_ok, msg


    def draw_mpl_frame(self, ax): draw_onion_frame(ax, self)




class Pedestal(UserGeometryBase):
    def init_geometry(self):
        desc = '''A NumBAT geometry template for a pedestal-type waveguide.'''
        self.set_properties('pedestal', 4, False, desc)



    def apply_parameters(self):
        # msh_name = self.get_param('_make_mesh_name(self._mesh_name,
        #                                  (self.get_param('domain_x, self.get_par
am('domain_y,
        #                                   self.get_param('inc_a_x, self.get_para
m('inc_a_y,
        #                                   self.get_param('pillar_x, self.get_par
am('pillar_y,
        #                                   self.get_param('slab_a_x, self.get_par
am('slab_a_y))

        subs = [('dx_in_nm = 100;', 'dx_in_nm = %f;', self.get_param('domain_x'))]
        subs.append(('dy_in_nm = 50;', 'dy_in_nm = %f;', self.get_param('domain_y')))
        subs.append(('a1 = 20;', 'a1 = %f;', self.get_param('inc_a_x')))
        subs.append(('a1y = 10;', 'a1y = %f;', self.get_param('inc_a_y')))
        subs.append(('a1top = 15;', 'a1top = %f;', self.get_param('inc_b_x')))
        subs.append(('slabx = 80;', 'slabx = %f;', self.get_param('slab_a_x')))
        subs.append(('slaby = 10;', 'slaby = %f;', self.get_param('slab_a_y')))
        subs.append(('slabxtop = 60;', 'slabxtop = %f;', self.get_param('slab_b_x')))
        subs.append(('px = 2;', 'px = %f;', self.get_param('pillar_x')))
        subs.append(('py = 5;', 'py = %f;', self.get_param('pillar_y')))
        subs.append(('lc = 0.1;', 'lc = %f;', self.get_param('lc_bkg')))
        subs.append(
            ('lc_refine_1 = lc/1;', 'lc_refine_1 = lc/%f;', self.get_param('lc_refine_1')))
        subs.append(
            ('lc_refine_2 = lc/1;', 'lc_refine_2 = lc/%f;', self.get_param('lc_refine_2')))

        return subs


class TrapezoidalRib(UserGeometryBase):


    def init_geometry(self):
        desc = '''A NumBAT geometry template for a trapezoidal_rib waveguide.

    Geometric parameters are:
```

```
    # inc_a_x  – width of the top of the rib
    # inc_a_y  – height of the top of the rib
    # slab_a_x – width of the middle of the rib
    # slab_a_y – height of the buried part of the rib

   Materials are:
    # material_bkg – background
    # material_a   – rib
    # material_b   – substrate

   Grid parameters are:
    # lc         – grid points arounds boundary as fraction of domain_x
    # lc_refine1  – refined density along upper rib
    # lc_refine2  – refined density along buried rib

    # Adjust so that the bottom of the emerging rib takes its grid from the buried part
   '''
        self.set_properties('trapezoidal_rib', 4, False, desc)


    def apply_parameters(self):
        # msh_name = self.get_param('_make_mesh_name(self._mesh_name,
        #                                  (self.get_param('domain_x, self.get_pa
ram('inc_a_x,
        #                                   self.get_param('inc_a_y, self.get_par
am('slab_a_x, self.get_param('slab_a_y))

        subs = [     ('dx_in_nm = 4000.0;', 'dx_in_nm = %f;',  self.get_param('domain_x')
)]
        subs.append(('dy_in_nm = 2000.0;', 'dy_in_nm = %f;', self.get_param('domain_y'))
)
        subs.append(('top_rib_width = 600.0;',       'top_rib_width = %f;',     self.get_param
('inc_a_x')))
        subs.append(('mid_rib_width = 900.0;',       'mid_rib_width = %f;',     self.get_param
('slab_a_x')))
        subs.append(('bottom_rib_width = 1800.0;', 'bottom_rib_width = %f;', self.get_param(
'slab_b_x')))
        subs.append(('rib_height = 500.0;',          'rib_height = %f;',        self.get_param
('inc_a_y')))
        subs.append(('slab_thickness = 300.0;',      'slab_thickness = %f;',    self.get_param('
slab_a_y')))

        subs.append(('lc = 0.020000;',              "lc = %f;", self.get_param('lc_bkg')))
        subs.append(('lc_refine_1 = lc/10.0;', "lc_refine_1 = lc/%f;", self.get_param('lc_refine_
1')))
        subs.append(('lc_refine_2 = lc/5.0;',   "lc_refine_2 = lc/%f;", self.get_param('lc_refine_
2')))

        return subs

    def check_dimensions(self):

        dom_x = self.get_param('domain_x')
        dom_y = self.get_param('domain_y')
        rib_wbot = self.get_param('slab_b_x')

        # TODO: more  checks

        msg = ''

        if rib_wbot >= dom_x :
            msg = 'Slab width (slab_b_x) is wider than domain width (domain_x).\n'
```

```

        dims_ok = not len(msg)
        return dims_ok, msg


class Rib(UserGeometryBase):

    def init_geometry(self):
        if self._d_materials['c'].is_vacuum():  # TODO: perhaps a better test is
 whether bkg = mat_c
            nt = 3
        else:
            nt = 4

        nt=3 # including bkg
        desc = '''A NumBAT geometry template for a rib waveguide. '''

        self.set_properties('rib', nt, False, desc)

        self.set_required_parameters(['rib_w', 'rib_h', 'slab_w', 'slab_h'],  num_mat
s=nt)
        self.set_allowed_parameters(['lc_bkg', 'lc_refine_1','lc_refine_2'],  num_allowe
d_mats=nt)
        self.set_parameter_help(
                {
                'rib_w': "width of raised rib region",
                'rib_h': "height of raised rib region",
                'slab_w': "width of slab substrate region",
                'slab_h': "height of slab substrate region",
                'material_bkg': "background material",
                'material_a': "material of triangular core",
                    }
                )


    def apply_parameters(self):
        # msh_name = self.get_param('_make_mesh_name(self._mesh_name,
        #                                  (self.get_param('domain_x, self.get_pa
ram('domain_y, self.get_param('inc_a_x, self.get_param('inc_a_y, self.get_param(
'slab_a_x, self.get_param('slab_a_y')))

        subs = [('dx_in_nm = 100;', 'dx_in_nm = %f;', self.get_param('domain_x'))]
        subs.append(('dy_in_nm = 50;', 'dy_in_nm = %f;', self.get_param('domain_y')))
        subs.append(('a1 = 20;', 'a1 = %f;', self.get_param('rib_w')))
        subs.append(('a1y = 10;', 'a1y = %f;', self.get_param('rib_h')))
        subs.append(('slabx = 80;', 'slabx = %f;', self.get_param('slab_w')))
        subs.append(('slaby = 10;', 'slaby = %f;', self.get_param('slab_h')))
        subs.append(('lc = 0.1;', 'lc = %f;', self.get_param('lc_bkg')))
        subs.append( ('lc_refine_1 = lc/1;', 'lc_refine_1 = lc/%f;', self.get_param('lc_refine_1
')))
        subs.append( ('lc_refine_2 = lc/1;', 'lc_refine_2 = lc/%f;', self.get_param('lc_refine_2
')))

        return subs

    def draw_mpl_frame(self, ax):
        rib_w = self.get_param('rib_w')*nmtoum
        rib_h = self.get_param('rib_h')*nmtoum
        slab_w = self.get_param('slab_w')*nmtoum
        slab_h = self.get_param('slab_h')*nmtoum

        vertices = np.array([
            [-slab_w/2, -slab_h],
            [-slab_w/2, 0],
             [-rib_w/2, 0],
             [-rib_w/2, rib_h],
             [rib_w/2, rib_h],
             [rib_w/2, 0],
             [slab_w/2, 0],
             [slab_w/2, -slab_h],
             [-slab_w/2, -slab_h]])

        ax.add_patch(mplpatches.Polygon(vertices,
            facecolor=None, fill=False, edgecolor='gray', linewidth=.75))

class RibCoated(UserGeometryBase):

    def init_geometry(self):

        desc = '''A NumBAT geometry template for a coated rib waveguide. '''

        self.set_properties('rib_coated', 4, False, desc)

        nt=4
        self.set_required_parameters(['rib_w', 'rib_h', 'slab_w', 'slab_h', 'coat_w',
'coat_h'],  num_mats=nt)
        self.set_allowed_parameters(['lc_bkg', 'lc_refine_1', 'lc_refine_2', 'lc_refine_3'
],  num_allowed_mats=nt)
        self.set_parameter_help(
                {
                'rib_w' : "width of raised rib region",
                'rib_h' : "height of raised rib region",
                'slab_w' : "width of slab substrate region",
                'slab_h' : "height of slab substrate region",
                'coat_w' : "horizontal thickness of coating layer",
                'coat_h' : "vertical thickness of coating layer",
                'material_bkg' : "background material",
                'material_a' : "material of raised rib core",
                'material_b' : "material of substrate slab",
                'material_c' : "material of coating layer",
                    }
                )

    def apply_parameters(self):

        subs = [('dx_in_nm = 100;', 'dx_in_nm = %f;', self.get_param('domain_x'))]
        subs.append(('dy_in_nm = 50;', 'dy_in_nm = %f;', self.get_param('domain_y')))
        subs.append(('a1 = 20;', 'a1 = %f;', self.get_param('rib_w')))
        subs.append(('a1y = 10;', 'a1y = %f;', self.get_param('rib_h')))
        subs.append(('slabx = 80;', 'slabx = %f;', self.get_param('slab_w')))
        subs.append(('slaby = 10;', 'slaby = %f;', self.get_param('slab_h')))
        subs.append(('coatx = 2;', 'coatx = %f;', self.get_param('coat_w')))
        subs.append(('coaty = 2;', 'coaty = %f;', self.get_param('coat_h')))
        subs.append(('lc = 0.1;', 'lc = %f;', self.get_param('lc_bkg')))
        subs.append( ('lc_refine_1 = lc/1;', 'lc_refine_1 = lc/%f;', self.get_param('lc_refine_1
')))
        subs.append( ('lc_refine_2 = lc/1;', 'lc_refine_2 = lc/%f;', self.get_param('lc_refine_2
')))
        subs.append( ('lc_refine_3 = lc/1;', 'lc_refine_3 = lc/%f;', self.get_param('lc_refine_3
')))

        return subs
```

```
class RibDoubleCoated(UserGeometryBase):

    def init_geometry(self):
        desc = '''A NumBAT geometry template for a double coated rib waveguide. '''
        self.set_properties('rib_double_coated', 6, False, desc)

        nt=5
        self.set_required_parameters(['rib_w', 'rib_h', 'slab_w', 'slab_h',
                                      'coat_w', 'coat_h', 'coat2_w', 'coat2_h' ],  nu
m_mats=nt)
        self.set_allowed_parameters(['lc_bkg', 'lc_refine_1', 'lc_refine_2', 'lc_refine_3',
                                     'lc_refine_4', 'lc_refine_5' ],  num_allowed_mats
=nt)
        self.set_parameter_help(
                {
                'rib_w' : "width of raised rib region",
                'rib_h' : "height of raised rib region",
                'slab_w' : "width of slab substrate region",
                'slab_h' : "height of slab substrate region",
                'coat_w' : "horizontal thickness of inner coating layer",
                'coat_h' : "vertical thickness of inner coating layer",
                'coat2_w' : "horizontal thickness of outer coating layer",
                'coat2_h' : "vertical thickness of outer coating layer",
                'material_bkg' : "background material",
                'material_a' : "material of raised rib core",
                'material_b' : "material of substrate slab",
                'material_c' : "material of inner coating layer",
                'material_d' : "material of outer coating layer",
                    }
                )

    def apply_parameters(self):

        # msh_name = self._make_mesh_name(self._mesh_name,
        #                                  (self.get_param('domain_x, self.get_pa
ram('domain_y,
        #                                   self.get_param('inc_a_x, self.get_par
am('inc_a_y,
        #                                   self.get_param('coat_x, self.get_para
m('coat_y,
        #                                   self.get_param('coat2_y, self.get_par
am('slab_a_x,
        #                                   self.get_param('slab_a_y, self.get_pa
ram('slab_b_y')))

        subs = [('dx_in_nm = 100;', 'dx_in_nm = %f;', self.get_param('domain_x'))]
        subs.append(('dy_in_nm = 50;', 'dy_in_nm = %f;', self.get_param('domain_y')))
        subs.append(('a1 = 20;', 'a1 = %f;', self.get_param('rib_w')))
        subs.append(('a1y = 10;', 'a1y = %f;', self.get_param('rib_h')))
        subs.append(('slabx = 80;', 'slabx = %f;', self.get_param('slab_w')))
        subs.append(('slaby = 10;', 'slaby = %f;', self.get_param('slab_h')))
        subs.append(('coatx = 2;', 'coatx = %f;', self.get_param('coat_w')))
        subs.append(('coaty = 2;', 'coaty = %f;', self.get_param('coat_h')))

        subs.append(('slab2y = 5;', 'slab2y = %f;', self.get_param('slab_b_y')))
        subs.append(('coat2x = 4;', 'coat2x = %f;', self.get_param('coat2_w')))
        subs.append(('coat2y = 4;', 'coat2y = %f;', self.get_param('coat2_h')))

        subs.append(('lc = 0.1;', 'lc = %f;', self.get_param('lc_bkg')))
        subs.append(
            ('lc_refine_1 = lc/1;', 'lc_refine_1 = lc/%f;', self.get_param('lc_refine_1')))
        subs.append(
            ('lc_refine_2 = lc/1;', 'lc_refine_2 = lc/%f;', self.get_param('lc_refine_2')))
        subs.append(
            ('lc_refine_3 = lc/1;', 'lc_refine_3 = lc/%f;', self.get_param('lc_refine_3')))
        subs.append(
            ('lc_refine_4 = lc/1;', 'lc_refine_4 = lc/%f;', self.get_param('lc_refine_4')))
        subs.append(
            ('lc_refine_5 = lc/1;', 'lc_refine_5 = lc/%f;', self.get_param('lc_refine_5')))

        return subs


class Slot(UserGeometryBase):

    def init_geometry(self):

        desc = '''A NumBAT geometry template for a slot waveguide. '''
        self.set_properties('slot', 4, False, desc)

        nt=4
        self.set_required_parameters(['rib_w', 'rib_h', 'slab_w', 'slab_h', 'slot_w' ],  num_mats=nt)
        self.set_allowed_parameters(['lc_bkg', 'lc_refine_1', 'lc_refine_2' ],  num_allowed_mats=nt)
        self.set_parameter_help(
                {
                'rib_w': "width of raised ribs",
                'rib_h': "height of raised ribs",
                'slot_w': "width of slot between ribs",
                'slab_w': "width of slab substrate region",
                'slab_h': "height of slab substrate region",
                'material_bkg': "background material",
                'material_a': "material of slot",
                'material_b': "material of substrate slab",
                'material_c': "material of ribs",
                'lc_refine_1': "refine factor for slot and ribs",
                'lc_refine_2': "refine factor for slab and ribs",

                    }
                )

    def apply_parameters(self):

        subs = [('dx_in_nm = 1000;', 'dx_in_nm = %f;', self.get_param('domain_x'))]
        subs.append(('dy_in_nm = 500;', 'dy_in_nm = %f;', self.get_param('domain_y')))
        subs.append(('slot_w = 200;', 'slot_w = %f;', self.get_param('slot_w')))
        subs.append(('rib_h = 100;', 'rib_h = %f;', self.get_param('rib_h')))
        subs.append(('rib_w = 200;', 'rib_w = %f;', self.get_param('rib_w')))
        subs.append(('slab_w = 800;', 'slab_w = %f;', self.get_param('slab_w')))
        subs.append(('slab_h = 100;', 'slab_h = %f;', self.get_param('slab_h')))
        subs.append(('lc_bkg = 0.1;', 'lc_bkg = %f;', self.get_param('lc_bkg')))
        subs.append(('lc_refine_1 = lc_bkg/1;', 'lc_refine_1 = lc_bkg/%f;', self.get_param('lc_refine_1')))
        subs.append(('lc_refine_2 = lc_bkg/1;', 'lc_refine_2 = lc_bkg/%f;', self.get_param('lc_refine_2')))

        return subs

    def check_dimensions(self):
        dom_x = self.get_param('domain_x')
        dom_y = self.get_param('domain_y')
        slot_w = self.get_param('slot_w')
        rib_w = self.get_param('rib_w')
```

```
        rib_h = self.get_param('rib_h')
        slab_w = self.get_param('slab_w')
        slab_h = self.get_param('slab_h')

        msg= ''

        if slab_w >= dom_x: msg += 'Slab width (slab_w) is larger than domain width (domain_x).\n'
        if slab_h >= dom_y/2: msg += 'Slab height (slab_h) is larger than half the domain height (domain_y).\n'
        if rib_h >= dom_y/2: msg += 'Rib height (rib_h) is larger than half the domain height (domain_y).\n'

        if slot_w+2*rib_w >= dom_x: msg += 'Slot and ribs are together wider than domain width (domain_x).\n'

        dims_ok = not len(msg)

        return dims_ok, msg

class SlotCoated(UserGeometryBase):

    def init_geometry(self):

        desc = '''A NumBAT geometry template for a slot waveguide. '''
        self.set_properties('slot_coated', 6, False, desc)

        nt=5
        self.set_required_parameters(['rib_w', 'rib_h', 'slab_w', 'slab_h', 'slot_w', 'coat_t' ],  num_mats=nt)
        self.set_allowed_parameters(['lc_bkg', 'lc_refine_1', 'lc_refine_2', 'lc_refine_3' ],
                                    num_allowed_mats=nt)
        self.set_parameter_help(
                {
                'rib_w': "width of raised ribs",
                'rib_h': "height of raised ribs",
                'slot_w': "width of slot between ribs",
                'slab_w': "width of slab substrate region",
                'slab_h': "height of slab substrate region",
                'coat_t': "thickness of coating layer",
                'material_bkg': "background material",
                'material_a': "material of slot",
                'material_b': "material of substrate slab",
                'material_c': "material of ribs",
                'material_d': "material of coating",
                'lc_refine_1': "refine factor for slot and ribs",
                'lc_refine_2': "refine factor for slab",
                'lc_refine_2': "refine factor for coating",

                    }
                )

    def apply_parameters(self):

        subs = [('dx_in_nm = 1000;', 'dx_in_nm = %f;', self.get_param('domain_x'))]
        subs.append(('dy_in_nm = 500;', 'dy_in_nm = %f;', self.get_param('domain_y')))

        subs.append(('slot_w = 200;', 'slot_w = %f;', self.get_param('slot_w')))
        subs.append(('rib_h = 100;', 'rib_h = %f;', self.get_param('rib_h')))
        subs.append(('rib_w = 200;', 'rib_w = %f;', self.get_param('rib_w')))
        subs.append(('slab_w = 800;', 'slab_w = %f;', self.get_param('slab_w')))
        subs.append(('slab_h = 100;', 'slab_h = %f;', self.get_param('slab_h')))
        subs.append(('coat_y = 100;', 'coat_y = %f;', self.get_param('coat_t')))

        subs.append(('lc_bkg = 0.1;', 'lc_bkg = %f;', self.get_param('lc_bkg')))
        subs.append(('lc_refine_1 = lc_bkg/1;', 'lc_refine_1 = lc_bkg/%f;', self.get_param('lc_refine_1')))
        subs.append(('lc_refine_2 = lc_bkg/1;', 'lc_refine_2 = lc_bkg/%f;', self.get_param('lc_refine_2')))
        subs.append(('lc_refine_3 = lc_bkg/1;', 'lc_refine_3 = lc_bkg/%f;', self.get_param('lc_refine_3')))

        return subs

    def check_dimensions(self):
        dom_x = self.get_param('domain_x')
        dom_y = self.get_param('domain_y')
        slot_w = self.get_param('slot_w')
        rib_w = self.get_param('rib_w')
        rib_h = self.get_param('rib_h')
        slab_w = self.get_param('slab_w')
        slab_h = self.get_param('slab_h')
        coat_t = self.get_param('coat_t')

        msg= ''

        if slab_w >= dom_x: msg += 'Slab width (slab_w) is larger than domain width (domain_x).\n'
        if slab_h >= dom_y/2: msg += 'Slab height (slab_h) is larger than half the domain height (domain_y).\n'
        if rib_h >= dom_y/2: msg += 'Rib height (rib_h) is larger than half the domain height (domain_y).\n'
        if rib_h+coat_t >= dom_y/2: msg += 'Rib height (rib_h) + coat thickness (coat_t) are together larger than half the domain height (domain_y).\n'

        if slot_w+2*rib_w >= dom_x: msg += 'Slot and ribs are together wider than domain width (domain_x).\n'

        dims_ok = not len(msg)

        return dims_ok, msg
```